CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Simulating an Ecosystem

Implementing and Analysing Virtual Ecosystems in Real-time Using the Unity Game Engine

Bachelor's thesis in Computer science and engineering

Mattias Heinl
Sebastian Holm
Edwin Holst
Albin Johansson
Oliver Karmetun
Karl Öqvist

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

Bachelor's thesis 2021

Simulating an Ecosystem

Implementing and Analysing Virtual Ecosystems in Real-time Using the Unity Game Engine

Mattias Heinl
Sebastian Holm
Edwin Holst
Albin Johansson
Oliver Karmetun
Karl Öqvist

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 and Analysing Virtual Ecosystems in Real-time Using the Unity Game Engine
Mattias Heinl Sebastian Holm Edwin Holst Albin Johansson Oliver Karmetun Karl Öqvist

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: Screenshot from the main menu of the simulation, depicting the four animals used in the simulations.

Typeset in LaTeX
Gothenburg, Sweden 2021

Simulating an Ecosystem
Implementing and Analysing Virtual Ecosystems in Real-time Using the Unity Game Engine

Mattias Heinl, Sebastian Holm, Edwin Holst, Albin Johansson, Oliver Karmetun, Karl Öqvist

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

Abstract

Many modern video games feature large virtual ecosystems inhabited by diverse animals with realistic behaviours. This project utilises the Unity game engine to simulate different configurations of virtual ecosystems, allowing the user to analyse the results in real-time as well as through visualisations of compiled simulation data. The simulated animals feature genome with genes that support mutation which affect different traits and behaviours, such as speed and vision. Ecological aspects such as evolution and carrying capacity along with miscellaneous trends in gene values are investigated in simulation results. Performance aspects are also discussed, e.g. through experimentation with data-oriented solutions such as ECS (Entity Component System) architecture. The results from numerous simulation iterations showed that simulated ecosystems are complex and hard to assess. Whilst some results presented in the report mimic aspects of real world ecosystems, it is difficult to draw conclusions from the results due to the many contributing factors. Overall, the final application serves as a simple means of investigating the effects of genomes and different initial conditions on the longevity and balance of small ecosystems over time.

Source code: `https://github.com/albin-johansson/ecosystem`

Sammandrag

Många moderna datorspel har omfattande virtuella ekosystem som innehåller flera olika arter av djur med realistiska beteenden. Detta projekt använder spelmotorn Unity för att simulera olika konfigurationer av virtuella ecosystem, där användaren kan analysera resultat i realtid såväl som genom visualiseringar av sammanställd simulationsdata. De simulerade djuren har genom med gener med stöd för mutation som påverkar olika attribut och beteenden, som hastighet och syn. Ekologiska aspekter som evolution och miljöns bärkraft tillsammans med diverse trender i genvärden undersöks i resultat från simulationerna. Prestanda diskuteras också, till exempel genom experimenterande med dataorienterade lösningar som ECS-arkitektur (Entity Component System). Resultaten från flera iterationer av simuleringar visade på att simulerade ekosystem är komplexa och svåra att bedöma. Medan vissa resultat som presentas i rapporten liknar beteenden i verkligheten så är det svårt att dra slutsatser av dessa resultat, på grund av de många faktorer som påverkar slutresultatet. The slutgiltiga versionen av den utvecklade applikationen utgör ett enkelt sätt att undersöka effekten av genom och initiala förutsättningar på livslängden och balans av små ekosystem över tid.

Källkod: `https://github.com/albin-johansson/ecosystem`

Keywords: simulation, ecosystem, evolution, ecology, real-time, visualisation, unity

Acknowledgements

We would like to thank our project supervisor Marco Fratarcangeli, for his enthusiastic support throughout the course of the project.

Mattias Heinl, Sebastian Holm, Edwin Holst, Albin Johansson, Oliver Karmetun, Karl Öqvist, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

Many modern video games feature large and diverse environments that the player can roam and interact with, such as the Elder Scrolls, Read Dead Redemption and Far Cry series. These environments are often made up of forests, plains, seas, lakes and mountains along with a flora and fauna, which together constitute virtual ecosystems. The immersive ecosystems in these games are often used as major selling points due to the great amount of work put into these aspects.

Virtual ecosystems can also be a source of great learning opportunities to explore natural ecosystems and the relations within it. With the use of modern computer graphics and technologies, these experiences are becoming increasingly realistic as technology progresses. A possible future direction of virtual ecosystems could be to adhere to ecological theories and laws.

Furthermore, virtual ecosystems present opportunities to serve as interactive learning tools, due to the possibilities of interaction and real-time visual feedback of various aspects of ecosystems. Visualisations of problems and making complex problems repeatable and testable is a great way to improve understanding of the intricate natural concepts and correlations [1].

1.1 Related Work

This section introduces related work in the field of simulating ecosystems and provides an overview of the state of virtual ecosystems and their use cases within ecology.

1.1.1 Virtual Ecosystems in Games

The video game *Red Dead Redemption 2* was released in 2018, and quickly became incredibly popular, having sold over 36 million copies [2], as of 2020. It has been praised for its detailed and believable open world. The developer, *Rockstar Games*, have stated that the game features ecosystems inhabited by 200 distinct species [3]. However, there is little documentation on the inner workings of these systems, making it difficult to assess to what extent the virtual ecosystems have been made to accurately simulate real world behaviours.

Another example of a game that tries to model virtual ecosystems is *Equilinox*, an

indie game released by *ThinMatrix* in 2018. The official Equilinox website states that Equilinox is a relaxing nature simulation game which allows you to create and nurture your own living ecosystems [4]. In the game, different species have different life-cycles, behaviours and requirements on habitat and resources. The individuals also carry genes and values for those genes that affect their attributes. Individuals inherit values for their genes that are similar to their parent but with the possibility of mutations. Certain values for the genes allows for the player to evolve an individual into an entirely new species.

1.1.2 Scientific Usage of Simulated Ecosystems

Computer simulations of ecology have been used previously in numerous research settings. One such example, Romanuk, et al. [5] compared simulations of species invasions (species unlike those previously present in the ecosystem), speciation events (species similar to previous in the ecosystem) and their effects on the food webs. Using the data from the simulations and comparing it with empirical data, it was determined that *"Contrary to much competition theory, these findings suggest that evolutionary and other processes that more tightly pack ecological niches contribute more to ecosystem structure and function than previously thought."* [5].

There have been previous simulations of the genetic factors in ecosystems, with one example being the research presented by Alan Grafen in his book *"Behavioural ecology: An evolutionary approach"*. Grafen presents research on how modelling and simulating genes, an often overlooked subject, can be a major telltale for the trends seen in ecology since it is from the genes that traits and behaviours stem from [6].

Other examples of graphical simulations of ecosystems includes a video series titled *Evolution*[7], by *Primer*, in which colourful blobs are progressively given traits and that exhibits different aspects of evolution. Another example is the work of Lassabe et. al. in *Advances in Artificial Reality and Tele-Existence* [8].

Computer simulation has also been used as a learning tool. An article in the scientific journal *Computers in Human Behaviour* goes into depth on how a class is taught the intricacies of factors that affects the environment of a pond using a virtual ecosystem. The program details the growth of algae leading to the death of fish in the pond and the student are subsequently tasked with drawing a "concept map" of the events that lead to this [1].

Another example of simulation used in a pedagogic setting was Agar [9]. Agar is what is called, by the creator Michael D. Travers, an *"Animal Construction Kit"* - a computer system made for students to build their own animals and simulate them in an ecosystem. It was done as part of a bigger project called *Vivarium* at MIT funded by Apple. The simulation was created using a kind of individual based model called an agent based model where each animal was its own entity with its own traits and behaviours. The whole Vivarium project had the objective of making ecology more interactive and engaging to learn with the use of modern computer

technology [10].

1.2 Objective

The objective of this project is to produce a software tool for simulating ecosystems with the use of Unity, a modern game engine. The simulations will to varying extents emulate real world ecosystems. Furthermore, basic concepts related to ecology, such as carrying capacity, natural selection and evolution will be investigated in the virtual ecosystems. It should be possible to analyse different scenarios and evaluate the effects of small changes to both biotic and abiotic factors of the ecosystem. Additionally, a tertiary objective is to make the virtual ecosystems provide real-time information about the state of the ecosystem, to aid the user's insight and understanding of the ecosystem. However, in order to be able to notice and evaluate results with confidence, the number of simulated entities should be relatively large, putting great strain on the hardware of the computer which introduces a need for the simulation to be efficient.

1.3 Scope and Limitations

To narrow the scope of the project, and to make sure that the project is finished in the allocated time frame, certain limitations have been introduced. These include both limitations to the application but also limitations within the virtual ecosystem.

Performance

The virtual ecosystem was developed for stable use on computers with a dedicated graphics card, a newer generation of processors and a large memory capacity. Different hardware specifications have not been considered in the developing process meaning that performance on less powerful computers can be unstable or even make the virtual ecosystem unable to run.

Virtual Ecosystem

The only aspect of the simulated ecosystems considered to be active are the animals. Therefore, plants are only considered as food resources or terrain. Furthermore, the simulated ecosystems are limited to static and local system, i.e. factors such as day/night cycles, seasons, and migration are not considered. As a result, the only dynamic aspects of the simulated ecosystems are the animals and the availability of food and water resources.

Simulation Duration

The simulated ecosystems need to be able to produce results within a relatively short time frame, i.e. a few minutes or up to an hour at the most since the virtual ecosystems are meant to be studied in real-time. Additionally, the idea is that different configurations can be tested and compared in a short period of time. Therefore,

aspects such as hunger and thirst rates, gestation periods and sexual maturity are sped up to match this time frame. This means that certain results will not be directly applicable to real world behaviours. For example, an animal might only eat and drink once during the entire duration of its pregnancy.

Reproduction

The simulated ecosystems only feature mammals, making sexual reproduction the only means of reproduction. The genomes that are present in the animals are simplified down to a handful of genes that in no way emulates any real-world gene. The gene system instead emulates some of the effects that a real-world genome can have on traits of the animals.

Behaviour of Animals

The animals in the virtual ecosystem are developed to have behaviours and traits that are as primitive as possible. The animals have unique eating and hunting behaviours but are apart from that equal in all behaviours. These are needed simplifications that reduces the complexity in creating a balanced ecosystem with reproducible results.

2
Theory

This chapter introduces the main theoretical subjects. The chapter is divided into two parts, where the first part elaborates on technical aspects of the project whilst the second part focuses on ecological and biological aspects.

2.1 Technical Theory

This section presents the concept of an agent-based model, along with important concepts related to the Unity game engine. Additionally, the ECS architectural pattern is introduced.

2.1.1 Agent-based Model

The virtual ecosystem in this project is built around an agent-based model (ABM). ABM is a fitting model to work from when dealing with what is called a complex adaptive system (CAS) that an ecosystem can be viewed as. With a CAS means a system that from a micro perspective is governed by simple laws and where events in the system are predictable but that from a macro perspective can give rise to very complex trends and pattern because of the shear number of different events happening in the system. The agent-based approach for modelling a CAS is a straightforward approach where all entities in the system are agents. The agents are independent but they have relations between them that govern how they respond to each other. The relations between two agents can be stochastic, meaning that the behaviour between the two agents is random and impossible to predict. The relations can however also be deterministic meaning that the behaviour between the agents is programmed in advance and can therefore be predicted knowing all factors that determine the behaviour [11].

The possibilities and use cases for agent-based models are many since the agents can model any entity. The ABM is also convenient in the cases where it can be challenging to describe a whole system with for example one equation. In those cases, it might be easier to describe each part of the system as agents and their relation to each other [11]. Simulations in ecology and ecosystems have been shown to benefit from this because the trends that can be seen from a macro perspective depend more on the individuals than the system as a whole. Agent based models also keeps the information local to the individual making it only act on that. It can otherwise be tempting or hard to not have individuals share global information of

the system and use that in the simulation [12].

2.1.2 Unity Engine

The Unity game engine, or simply *Unity*, is the game engine used in this project as the main developing tool for the virtual ecosystem. Game engines, including Unity, are aimed to ease and speed up the development of video games and other graphical applications. Unity includes support for performant 3D graphics, physics simulation, animations, user input and much more. As a result, a lot of common "boilerplate" aspects of game development can be reused in multiple projects. Furthermore, game engines can also be used for producing media content such as animated movies or even architectural demonstrations.

Game Object

Development in Unity is centred around *game objects* that represent different objects in the game world. In Unity game levels are referred to as *scenes*, and are built from a hierarchy of game objects. Furthermore, each game object can have different *components* (also referred to as *scripts*) attached to them, that are updated each frame of the game. This is how custom behaviours, abilities, visual characteristics and other attributes are assigned to game objects in Unity. A component is implemented as a C# class, that inherits from the `MonoBehaviour` class, which is a part of the Unity API. In general, all game objects feature a *transform* component (a position, rotation and scale) as a minimum [13].

Colliders and Triggers

Collisions are a key component of the developed ecosystem because it is used to simulate vision and contact between animals and resources. The collision detection in Unity is primarily handled using *colliders*, which are standard components that can be attached to game objects to let the game objects collide and react to each other. There are many different colliders with different shapes, such as spheres, boxes and capsules. Furthermore, colliders can be configured as *triggers*, which causes them to emit collision events instead of actually causing physics collisions between game objects [14].

NavMesh

The navigation within the ecosystem that game objects perform are handled with the use of a *NavMesh*, a data structure designed to enable navigation and path finding. The NavMesh contains data on which areas of the map are walkable and are structured to provide fast path finding. All navigating game objects have a `NavMeshAgent` component attached to them and it is through that component and the large set of functions that work on the NavMesh data that paths can be calculated. The developer can choose to have all agents be aware of each other and all possible obstacles making the path calculations dynamic. Meaning that if an

agent or any other obstacle obstructs the current path of an agent the path gets recalculated to avoid collision and provide stable movement [15].

2.1.3 Shader

A shader is a program that runs on the graphics card that takes the graphical information of an object as input and outputs a generated image. The shader mathematically defines how light should behave on the object and can as such produce numerous visual effects. Shaders are being used in this project to aid in visibility and distinction of entities in the scene.

2.1.4 Ambient Occlusion

Ambient occlusion is a rendering/shading effect that creates soft shadows based on scene geometry. It functions by estimating the amount of ambient light that should be interacting with a given point, in order to approximate shadows caused by real lighting. The virtual ecosystems developed in this project make use of ambient occlusion to improve the overall graphical quality, but also to increase the contrast and aid the visibility of entities in the scene.

2.1.5 Entity Component System

Entity Component System (ECS), is a data-oriented architectural design pattern that is occasionally utilised in games, due to performance advantages from being designed with data locality in mind. The idea is that the CPU can better utilise its caches and spend less time jumping around in memory. In an ECS architecture, there are three core concepts: entities, components, and systems, hence the name. Entities represent distinct "objects", and are usually implemented as simple integer identifiers. Subsequently, components are simply plain data containers, often implemented as structs with only public data. The components represent different aspects or abilities that an entity might have. For example, movable entities in a game might feature a `Movable` component, which simply contains a position and velocity vector. Unlike object-oriented game objects, components are not stored with entities. Instead, the different components are stored in separate contiguous arrays, enabling very fast iteration speeds. Components are then "attached" to entities, without having the entities store the components themselves. These components are then handled by systems, which are the pieces of code that contain all the logic, iterating entities with different components. For example, a game might feature movement system, that iterates all entities with the aforementioned `Movable` component, updating the positions according to the current velocities.

2.2 Biological Theory

This section introduces relevant concepts from the field of ecology, such as carrying capacity and evolution. Furthermore, concepts related to genetics are also presented, such as genes, mutations and phenotype.

2.2.1 Ecology

Ecology is branch in biology that concerns the relations between all entities that constitute nature. This means relations between biotic factors (living components and once living components) as well as the relations between these biotic factors and the abiotic factors (physical conditions and non-living components) that they come in contact with. The main part of ecology is about changes in population sizes of different animal and plant species and how these correlate to each other. Ecology also takes into account the presence of microbial factors such as bacteria and viruses. [16]. In ecology the most fundamental approach are ecosystems which is a figurative encapsulation of nature. The encapsulation can be of any size meaning that it could include the whole world or only a small cubic centimetre of soil where the contents of the ecosystems are considered to be all abiotic and biotic factors inside the encapsulation. Ecosystems make it possible to study nature without the full complexity since it is limited in size and number of entities in it [17].

Animal Distribution in Ecosystems

Biomass is a term in ecology to describe the mass of biotic factors in an ecosystem. The total amount of biomass in a system gives insight into not only the distribution of for example plants, animals, fungi and microbial life but also distribution of different species of aforementioned groups when divided among their own total share of biomass. In this project, research results from studies of biomass were used to get a basic real-world distribution of population sizes of the animals used. Research show that the amount of prey biomass outnumbers the predators' biomass by a magnitude of at least a hundred, but it can vary depending on region and a plethora of other factors. Biomass is however a weight unit and does not correlate to the number of entities, for example bears occupy a much larger biomass than rabbits because of their larger weight. The metric however still show that the population size of prey outnumbers that of the predators [18].

Carrying Capacity

In ecology carrying capacity is defined as the maximum population size of a species or group of species that can be sustained by the ecosystem. The causes for changes in carrying capacity are many: changes in population of predators, availability of food or water and even the spread of diseases. When the carrying capacity of an ecosystem is surpassed, the population sizes will eventually decrease to a level that can be sustained or collapse [19].

Studying population size changes while plotting them in a diagram gives a good representation on where the carrying capacity lies since population in most cases tend to approach the carrying capacity or fluctuate around it. Population growth and carrying capacity are of great importance in ecology because they are strong indicators of patterns in populations and ecosystems [20]. In this report carrying capacity and population growth diagrams are used to determine and discuss the stability and the success of populations. It also gives a clear metric to be used in

comparisons between different simulations.

2.2.2 Genes, Genomes, and Phenotype

Genome is a term describing the complete set of genes that are present in a living entity and can be constituted by upwards of 31,000 different genes [21]. The genome is what governs how the creature appears and behaves which can be summarised in the concept *phenotype* meaning the collection of traits and behaviours that are observable in a biotic creature [22].

The biggest changes to the genome occur through a process called mutation where the DNA inside of the gene gets altered and are the most common source of genetic variation, playing a big role in the evolution of life [23]. The changes can be brought about for numerous reasons, what is noteworthy is the two types of mutations: hereditary and acquired mutations. Hereditary mutation is a mutation that gets passed on through the DNA from an individual's forefather and will therefore be present in the entirety of the individual's life. The individual can continue passing this mutated gene unto its offspring as well. The other type of mutation is acquired during the life of the individual. This type of mutation can occur because of environmental factors such as radiation or an error in the copying of DNA in the process of cell division. Acquired mutations cannot be passed on to offspring unless the mutated genes are present in the gametes, the cells that are present in the eggs or sperm [24].

2.2.3 Evolution Through Natural Selection

Evolution through natural selection is the theory that is the most well accepted in the scientific community and as such the theory that the virtual ecosystem has been developed with in mind. The theory explains how animals and plants have changed since the first basic life forms emerged on earth, and how these changes have resulted in new species [25].

The theory of evolution through natural selection was based on four facts: phenotype vary within a species, parts of the phenotype get passed on to offspring, the phenotype contribute to the life span and the success of reproduction of the species, and more animals are born than can survive. Evolution theory can get summarised to that the individuals that are adapted well enough to their environment (phenotype that is beneficial enough) survives and have a greater chance of passing along their genes. Meanwhile, poorly adapted individuals have a lesser chance of doing so making the population as a whole diverging towards having a phenotype that is better adapted to their environment [26].

3
Method

This chapter presents in-depth explanations of the methods used during the design and implementation of the simulation and describes the development process. Furthermore, the methodology of the various tests conducted related to genes, carrying capacity and performance is also presented.

3.1 Simulation Design and Implementation

This section discusses the design choices and approaches used in the designing and development of the virtual ecosystem.

3.1.1 Software

The simulation was developed using the Unity game engine, which was chosen due to its relative simplicity and widespread adoption. Subsequently, all source code related to the simulation was written in the C# programming language. Additionally, scripts for visualising the obtained simulation data were written in the Python programming language, using the open-source library *matplotlib* [27].

3.1.2 Data Collection and Visualisation

The simulation continuously monitors the state of the ecosystem, such as food consumption, food generation, animal deaths and animal reproductions. This was implemented using C# events, which are emitted by the various scripts responsible for these different aspects. A centralised logging manager subscribes to all of the simulation events, which manages an in-memory representation of the events throughout the entire duration of the simulation. Upon termination of the simulation, the recorded data is stored as a JavaScript Object Notation (JSON) file, using a custom format. Subsequently, the generated JSON files can be processed with Python visualisation scripts, that generate a collection of diagrams of the simulation data, to aid the analysis of the simulations of the virtual ecosystem and their parameters.

3.1.3 Simulation Model

The simulation developed in this project follows an agent-based model where every animal in the ecosystem is an agent that is governed by its basic traits, behaviours, and relations to other animals. The relations are deterministic and so are

the behaviours, though some behaviours have some stochastic elements to them (e.g. direction in wandering behaviour explained in section 3.1.13).

3.1.4 Configuration

Through an initial main menu, different aspects of the simulation can be configured, such as graphical fidelity and simulation behaviour. Post-processing effects such as anti-aliasing and ambient occlusion can also be configured through the same menu. The anti-aliasing can either be turned off, or assigned as one of SMAA (sub-pixel morphological anti-aliasing), FXAA (fast approximate anti-aliasing) or TAA (temporal anti-aliasing). Similarly, ambient occlusion can be either be turned off, or enabled with adjustable intensity, specified as a percentage. The main purpose of the different anti-aliasing and ambient occlusion options is to enable improved performance on less performant computers sacrificing visual quality.

In addition to graphical settings the user can change how the genomes are configured and inherited in the simulation. For each species it is possible to decide whether the initial genome should be the same for all individuals or if they should be partitioned into sets. The user can also choose whether to allow mutations, or if the gene values should stay in the sets instead. If the user intention is to investigate animal behaviour and run the simulation with an objective to examine carrying capacity or different animal distribution setups, the gene sets can be turned off to enhance the stability and repeatability of the setup. On the other hand, if the intention is to run longer simulations investigating what genes are most advantageous to the environment, the option of having sets active can be beneficial.

3.1.5 Scenes

There are a total of four different scenes available in the simulation. However, one of these scenes (the "Testing Ground" scene) was not intended to be used to simulate ecosystems, it is as the name suggests a testing ground primarily used for development purposes. The other scenes are the forest, dynamic and ECS scenes. The forest and dynamic scenes feature the complete simulation aspects, both using the conventional object-oriented Unity framework. Lastly, the ECS scene makes use of the Unity ECS framework, although it does not feature nearly as complex logic as the non-ECS scenes, since it was developed as a demonstration of the performance potential of the ECS framework (see 3.3.2).

3.1.6 Animal Types

All animals were split into three categories: wolves are carnivores, deer and rabbits are herbivores, and bears are omnivores. The animals differ in many of their behaviours, but they share the same core functionality: wandering, drinking and mating. The main differences lie in eating behaviours, where herbivores flee upon encountering predators, whilst carnivores and omnivores will try to chase down any herbivores they encounter (if they are sufficiently hungry). There is no aggression

between predators or between prey, i.e. two individuals of the same species will not attack each other.

3.1.7 Genome

Each animal has a set of genes, called a genome, where each gene corresponds to some functional value for the animal. The phenotype is strictly tied to the genes meaning that there is no element of chance in the creation of an animal's phenotype except for the case where mutations have been chosen to be allowed, then small changes can occur when new individuals get birthed. The genomes in the animals within the virtual ecosystem are constituted of the following genes: hunger rate, thirst rate, hunger threshold, thirst threshold, vision, speed, gestation period, and sexual maturity age.

The hunger rate gene affect the metabolism, which is how fast the hunger increases. Likewise, the thirst rate gene, affects the rate of water consumption. Threshold genes determine at which degree of hunger or thirst at which the animals should start looking for the associated resource. How far an animal sees and how fast it can move around in the terrain is directly correlated to the vision and speed genes. Since a fast animal with great vision can seem objectively better than one with lover speed and vision, these genes are balanced by making the animal's metabolism dependent on them, together with the hunger rate gene, in a multiplicative relationship. The gestation period gene determines the duration for which a female is pregnant. When pregnant the hunger and thirst increase faster, nurturing the child so that it will be birthed with lesser need to eat and drink in comparison to individuals with a shorter gestation period. Lastly the sexual maturity age gene affects the time from birth to the time when the animal can mate. All animals are considered to be immature before they reach their age of sexual maturity. This means that a lower maturity age makes it possible for animals to get pregnant/impregnate faster in comparison to animals with a higher age of maturity.

Since certain aspects of an animal could be dependent on more than one gene the genome has composite factors. A genome contains exactly one such composite factor, namely metabolism which is dependent on hunger rate, speed, and vision. As such there is a trade-off to having higher speed or longer vision, being higher food needs.

3.1.8 Acquisition of Genes

Each animal acquires its genome either during its reproduction or by the initialisation (if it belongs to the first generation of the simulation). The first generation is provided with a value for each gene or if gene sets are enabled by randomly selecting from a set of values. In the case of reproduction, the new genome is generated from the two parent genomes. For each gene, one is selected at random (with even 50/50 odds) from either of the parents. When reproducing, mutation adds a chance for each gene to take a new random value within the allowed range of the gene. Both gene sets and mutation can be enabled or disabled for each species in the menu of

the simulation.

3.1.9 Food and Water

The animals can drink from designated water sources, implemented as invisible game objects, that are placed along the shorelines of lakes and rivers. The animals have a thirst that is always increasing and when surpassing a threshold will trigger the need for water. The thirst is gradually quenched when colliding with a water source. Similar to thirst, all animals experience hunger, which will cause animals to look for food when they get sufficiently hungry. Wolves hunt and eat deer and rabbits, rabbits eat berries and carrots, bears are omnivores and can consume rabbits and deer as well as berries and carrots, and deer consume grass which they find everywhere on grassy terrain.

3.1.10 Food Generation and Decay

Carrots are placed at random position in the scenes. Berries regenerate, but are always placed on berry bushes at stationary positions. Additionally, whenever an animal dies, it results in a carcass (represented by a piece of meat) being placed at the position of the deceased animal, which can subsequently be consumed by carnivores. However, carcasses will only remain for a certain duration after which they will disappear. The hunger value that the berries, carrots and meat replenish varies. As a result, different animals require different amounts of food to be fully replenished.

3.1.11 Mating

Animals are through their genome defined as either male or female, and become sexually mature after a certain amount of time after their birth. Mature animals actively search for mates, as long as their thirst and hunger are under control (and as long they are not chased by predators). After a mating has taken place, the female animal is pregnant for a duration of time that is determined by their genome. During pregnancy, the female will have increased hunger and thirst needs. The longer the gestation period, the less hungry and thirsty the child will be when entering the world.

3.1.12 Memory

All animals feature a limited memory of locations of water sources. When an animal discovers a water source the location of the resource is stored in the memory, regardless of whether the animal consumes the resource. This means that animals can recall previous positions of water sources, reducing their need to constantly explore the terrain when thirsty.

3.1.13 Navigation

The animals wander in the terrain looking for food, water, or a mate. The wandering works by setting a destination at the edge of its sensory range and repeating the process when the destination is reached. If the animals detect food, water, or a predator detects a prey it will change target and go to the new target instantly. If a prey detects a predator it will try to flee in a direction opposite of the predator. If the opposite direction is blocked, the animal will try to flee in one of seven predefined angles away from the predator. The fleeing direction is updated regularly, making the hunted animal very responsive to changes in the trajectory of the predator. When hunting or being hunted the animals have the ability to sprint at 150% of its walking speed at the cost of stamina. The animal will return to walking when the stamina has been depleted. Stamina is subsequently regenerated as long as the animal is not sprinting.

3.1.14 Collisions

The interactions between game objects in the simulation are implemented through the use of collision detection in the physics engine in Unity. Each animal feature a sphere collider and a box collider. The sphere collider represents sensory range and is used so that the animals can detect other animals and resources. The box collider, roughly the size of the animal, is used to represent the collision of the actual animal. For example, the collision between the box collider of a wolf and the box collider of a rabbit being chased by said wolf indicates that the wolf has caught up with the rabbit and can attack.

Layer Collision Matrix

Different collision layers were used in order to minimise the number of collisions handled by the state machine, using the unity built-in layer collision matrix. The matrix provides an overview of all layer combinations and whether or not they should collide. Each animal makes use of two distinct collision layers, one for its large sphere collider, that represents its field of view, and one for the hit-box based on the body of the animal.

3.1.15 State Machines

The animals in the simulation feature a state machine, which determines the behaviour of each animal. This separates logic related to specific states into distinct classes, reducing the complexity of implementing the behaviours. As seen in the table 3.1, several animals implement the same states. This is due to the fact that the animals share most of its fundamental behaviours, such as drinking and eating. However, in some of these cases the implementation is slightly different, e.g. the `LookingForFood` state has the same general purpose for all animals, but since different animal species have different food preferences, they all have a unique implementation of said class. Below, it is briefly described what the purpose of each state is and what the requirements to enter that state are.

State	Rabbit	Deer	Wolf	Bear
LookingForWater	✓	✓	✓	✓
LookingForFood	✓	✓	✓	✓
Idle	✓	✓	✓	✓
Fleeing	✓	✓		
Drinking	✓	✓	✓	✓
RunningTowardsWater	✓	✓	✓	✓
RunningTowardsFood	✓		✓	✓
ChasingPrey			✓	✓
LookingForMate	✓	✓	✓	✓
Attacking			✓	✓
Eating	✓	✓		✓

Table 3.1: Overview of all available states and which states the different animals support.

`LookingForWater`

The `LookingForWater` state can be entered when an animal becomes sufficiently thirsty. In this state the animals will try to recall previous locations of water sources from their memory. If the animals do not know of any previous water sources, they will roam the terrain in search of new water sources.

`LookingForFood`

When the hunger of an animal surpasses a certain threshold, it can enter the `LookingForFood` state. The state serves the purpose of wandering around and reacting to collisions with a preferred food source, such as berry bushes for rabbits and bears or meat for wolves and bears. Predators will also react to live prey.

`Idle`

The `Idle` state is the initial state of all animals when the simulation starts, the animals leave this state when maturing and goes to the `LookingForMate` state. The animal will then only return if they are pregnant and all other needs are fulfilled.

`Fleeing`

The `Fleeing` state is only used by prey animals, i.e. rabbits and deer. This state can be entered from all other states, whenever a prey detects that a predator is nearby. Animals that are fleeing will actively steer away from the threat, and deplete their stamina by sprinting, in order to evade their pursuer.

`GoingToWater`

When an animal in the `LookingForWater` state recalls or detects a water source, it enters the `GoingToWater` state and start moving towards the source in order to start drinking.

`Drinking`

The `Drinking` state can be entered when an animal is in the `GoingToWater` state and collides with a water source. Whilst inside the `Drinking` state, the animal quench their thirst gradually.

`GoingToFood`

When a hungry animal recalls or detects a possible food source, e.g. a carcass or a carrot, it enters the `GoingToFood` state. However, this state should not be confused with `ChasingPrey`, which is used by hungry predators when they detect *live* prey, whilst the `GoingToFood` only incorporates food that is stationary.

`ChasingPrey`

The `ChasingPrey` state can be enabled when a hungry predator detects a live prey. The predators will try to catch the prey by sprinting, which gradually depletes their stamina.

`LookingForMate`

When animals are not hungry nor thirsty (and are not chased by predators, in the case of prey), they may look for a mate by entering the `LookingForMate` state. This state can only be entered by animals that are sexually mature, to prevent newborn animals from immediately looking for mates.

`Attacking`

When predators that are chasing prey get withing striking distance, they will enter the `Attacking` state. By the time this state is entered, the prey that was chased has been killed, and a carcass will be spawned at the location of the deceased prey.

`Eating`

The `Eating` state can be entered whenever an animal is eating from a stationary food source such as berry bushes. Note, the reason that bears support this state whilst wolves do not, is because bears are able to consume berries, since they are omnivores.

Switching Between States

Since only one state can be active at a time for each animal, and several requirements for entering the states can be fulfilled at the same time a priority system was implemented. The priorities were chosen based on what seemed to be most urgent in terms of surviving. This led to the following priorities: `Fleeing`, `GoingToFood` and `GoingToWater`, `LookingForFood` and `LookingForWater` (which ever resource has the larger deficit), `LookingForMating` and lastly `Idle`. In the bears' chase, this resulted in the state diagram in Figure 3.1. For all state diagrams including the transition conditions, see the figures in appendix A.

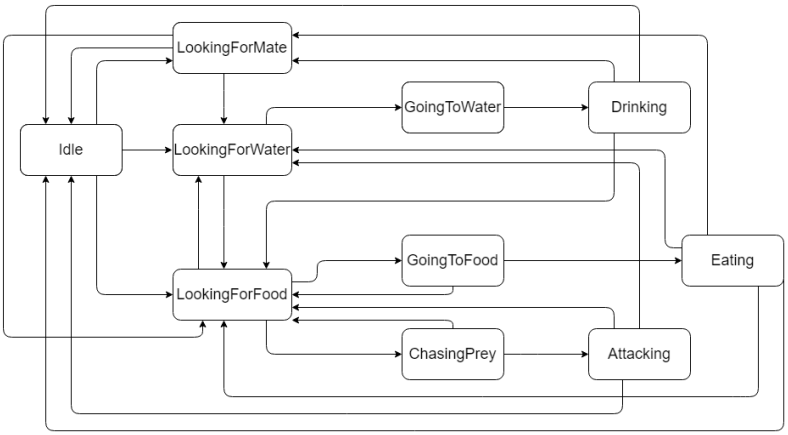


Figure 3.1: The figure displays the state diagram of the bear state machine, displaying the different states with associated transitions.

3.2 Graphics

This section explains in detail how the graphics for the simulations were developed and what technologies were used to enhance the aesthetics and usability of the virtual ecosystem.

3.2.1 Third-Party Packages

The graphical assets used in the simulation stem from a third party package created by *Polyperfect*. This package included all the animal materials, textures and animations as well as terrain assets such as trees, rocks, and plants that were also used. There were some modifications done to the package to better suit the simulation, e.g. animator controllers and animal materials were improved and/or replaced entirely.

3.2.2 Animations

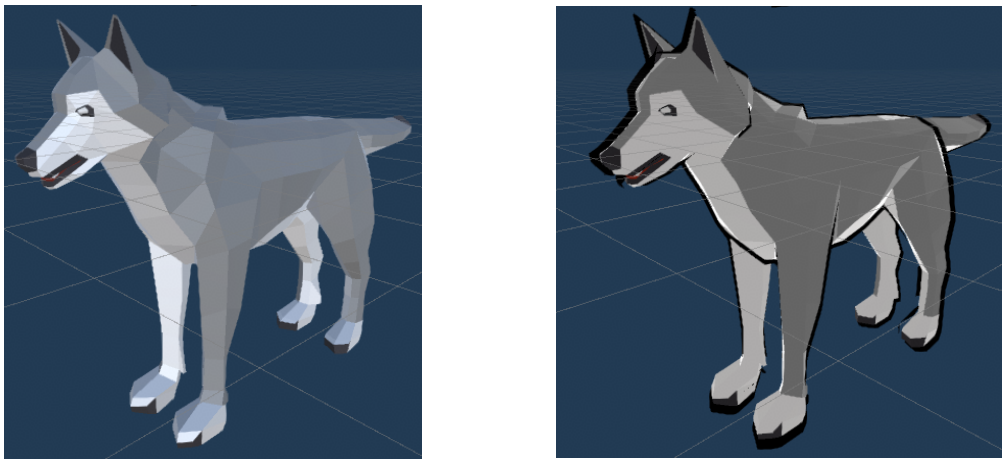
All animals are fully animated for each possible state. As such, all animals have idle, walking, running, attacking and dying animations. The animations are originally from a third-party package (from Polyperfect). However, their associated animation controllers were rewritten to better suit the custom state machines used in the project. The different states are able to enable different transition animations by simply setting Boolean values in their associated animation controller.

3.2.3 Terrain

The terrains in the simulation consists of a grass, trees, boulders, and rivers and lakes with shorelines. The terrain in the scenes were constructed with the Unity terrain tool and is static making it a more stable platform for doing multiple consecutive simulations and comparing results. The particular shape and content of the terrain varies between different scenes.

3.2.4 Toon Shader

A family of shaders known as toon shaders are used in the simulation on various objects to primarily aid the visibility of different objects at a distance. Toon shaders (also referred to as *cel* shaders), mimic the visual style of cartoons. For instance, by making objects appear with black outlines, a paper-like texture and flat shading. An experimental toon shader provided by Unity [28] was used. See Figure 3.2 for an example of the effects of the toon shader when applied to the wolf model used in the simulation.



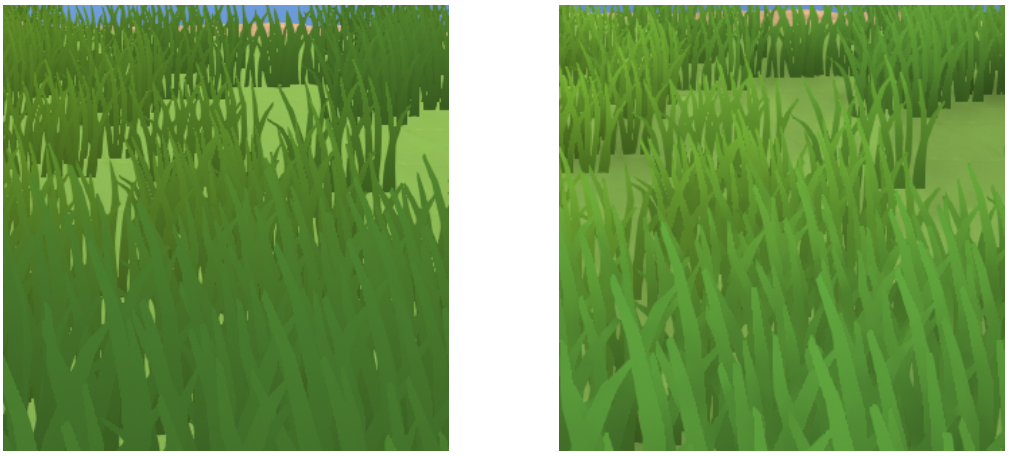
(a) Wolf without toon shader.

(b) Wolf with toon shader.

Figure 3.2: Wolf with and without toon shader applied. Note the addition of a black outline, reduced reflections and increased uniformity of colour when the toon shader is enabled.

3.2.5 Ambient Occlusion

Ambient occlusion is utilised in the project to improve the overall visual quality of the simulation, by producing soft shadows. Furthermore, ambient occlusion aids with increasing the contrast between animals and the terrain, improving visibility. The ambient occlusion is partially "baked" into scenes when the lighting data for the different scenes is generated. However, the most noticeable ambient occlusion effects come from post-processing. See Figure 3.3 for an illustrative example of the effects of ambient occlusion on the grass in the main menu of the simulation.



(a) (b)

Figure 3.3: Grass without any ambient occlusion (a) and with strong ambient occlusion (b). The grass in image (a) is hard to distinguish and there is not much depth in comparison to image (b), where each single straw of grass is easier to distinguish.

3.3 Performance

This section discusses the optimisations that were performed to improve the overall performance of the virtual ecosystem.

3.3.1 Object Pooling

The simulations are meant to run over many generations, to be able to see general trends in the statistics. This entails dynamic allocation and destruction of a large quantity of game objects, such as when food items are consumed or when animals are born, etc. However, dynamic memory allocation is costly, and the animals are constructed of several game objects with a considerable number of attached components, which gives them a non-trivial memory footprint. As a result, object pools are used by the simulation to reuse game objects, reducing the overall number of dynamic allocations. In general, there are distinct object pools for each animal and food resource. This works by letting objects that would normally be destroyed to be inactivated, and later reused when needed. This optimisation likely results in increased overall memory usage, in favour of improved execution speed due to fewer dynamic allocations. Additionally, start-up times are also slightly negatively affected, since the pools are allocated immediately when the simulation starts.

3.3.2 Entity Component System

To be able to get more accurate results the number of simulated entities must be large and a goal was for the ecosystem to have population sizes in the magnitude of a few thousand animals. However, this turned out to be very difficult using the conventional game object workflow, due to inherent performance limitations caused by poor data locality and cache misses. To address this issue, a more data-oriented

approach was deemed necessary. As a result, the plausibility of utilising an ECS architecture to improve the general performance of the simulation was investigated. By the time of this realisation, a considerable amount of code had been written for the conventional Unity framework. As such, a proof of concept was developed in parallel with further development of the simulation in the conventional object-oriented style. This work resulted in the "ECS Demo" scene, accessible through the main menu of the application, which consists of a simplified version of the main simulation that is implemented using an ECS architecture. Due to the fact that the group did not commit to the ECS architecture for the entire simulation, it was important to ensure that scenes developed without ECS support would continue to work as expected. For that reason, the ECS systems are dynamically enabled or disabled depending on the choice of scene that is run. This means that the simulation does not unnecessarily run ECS logic in non-ECS scenes, and vice versa.

3.4 Process

This section gives a chronological report on the development process of the virtual ecosystem.

3.4.1 Prototyping

The initial prototyping took place in a small scene with flat terrain. Basic behaviours such as roaming, eating and drinking were the focused on at this stage. Subsequently, interactions between animals were experimented with using triggers. Most of the early work consisted of trying out different approaches in order to figure out how to utilise Unity in the best way. Only rabbits and wolves were available for a considerable amount of time during the prototyping phase, with similar behaviours. The initial prototyping scene was never intended to be used for actual simulations, but it is included in the final version of the application (called "Testing ground").

3.4.2 Crafting Interesting Ecosystems

The visual aspects of the simulation were considered important. As such, the scenes needed to be visually interesting and appealing. This led to the introduction of the forest scene, the first attempt at crafting interesting ecosystems, featuring trees, lakes, rivers along with miscellaneous decorations. Whilst it turned out quite nicely, it was not configurable by the user and was still quite small. The forest scene was the main scene used during the early stages of the project, when more and more features were added.

3.4.3 Implementing Complex Behaviours

As more complex behaviours were introduced, the need for a better way to separate and implement different behaviours grew apparent. As a result, state machines were introduced, where each animal had distinct implementations of different states. This simplified development and reduced the complexity of animal prefabs. Eventually,

deer and bears were introduced when the state machine architecture was in place. However, these animals did not initially differ in any aspect except visual compared to rabbits and wolves, respectively.

3.4.4 Configurable Scenes

In order to be able to compare simulated ecosystems, initial aspects such as population sizes needed to be configurable. For this reason, the dynamic scene was introduced. The dynamic scene was designed along with a configuration menu that is shown before the simulation starts, which enables the user to specify the initial number of animals and food. This led to the introduction of the main menu, which lets the user choose which scene that will be run. Additionally, graphical options such as anti-aliasing and ambient occlusion were also made configurable to improve performance.

3.4.5 Balancing Attributes

The animals in the ecosystem feature numerous attributes that affect their chances of survival, e.g. the maximum amount of hunger, thirst and stamina along with size and the nutritional value of carcasses. In order to balance these attributes, aspects such as hunger, thirst and stamina were initially set to the same values for all animals. Other aspects such as size were configured to achieve realistic proportions, except for rabbits that were slightly enlarged in order to make them more visually distinctive. The nutritional value of carcasses was determined in a similar manner. Subsequently, the different animals were slowly given more distinctive gene values through a process of empirical trials to make sure that no animals grew too powerful.

3.5 Testing Carrying Capacity

A set of guidelines were introduced in order to test the carrying capacity of ecosystems. These included a process of gradually increasing the amount of different animal species for each test, i.e. the test began with rabbits as the sole species present in the simulation, and subsequently introducing wolves, deer and bears. All carrying capacity tests were conducted with immutable genomes. That is, the genes were assigned values at the start of the simulation and values would remain the same during the entire simulation, with no mutations at all. The dynamic scene was chosen as the scene to use when conducting carrying capacity tests, where each simulation run lasted for 1,000 seconds.

3.5.1 Food Availability

The food availability was considered a key limiting factor for the survival of the animals. This was tested by running simulations with different initial ratios of animals and food resources. Three configurations were used to test the effects of food availability. The first configuration used 500 rabbits and 2,000 berry bushes and ran for 400 seconds, whilst the second configuration used the same amount

of rabbits and berry bushes, but ran for 1,000 seconds in order to investigate the effects of letting the simulation run longer. The third and last configuration used 500 rabbits and 400 berry bushes and ran for 1,000 seconds, emulating more scarce food conditions.

3.5.2 Impact of Predators

The impact of predators was tested using two different configurations. The first used 500 rabbits, 50 wolves and 400 berry bushes, whilst the second configuration increased the amount of berry bushes to 2,000. The rationale for this was to see how predators affect the carrying capacity and if an increased food supply could counteract the effects of the wolves. The ratio of rabbits to wolves were based on the research presented in section 2.2.1.

3.5.3 Simulating All Animals

Two configurations were used when testing carrying capacity in ecosystems with all animals and features enabled. The first configuration used 200 instances of each animal type along with 500 berry bushes. Subsequently, the second configuration used a distribution of animals based on the research presented in 2.2.1, with 400 rabbits, 100 deer, 20 bears, 50 wolves and 500 berry bushes.

3.6 Testing Evolution

In order to verify that reasonable conclusions regarding evolution in the simulated ecosystems could be made, a number of tests simulations were conducted. The tests were intended to determine if and how genes changed over time. Mutations were disabled during these tests, and genes which differed between simulations were compared, as judged by the logging and visualisation scripts.

Three tests were constructed, each attempting to conclude the effects of certain variables on the gene populations of animals over extended periods by comparing the results of two simulations with different starting configurations. The first test examined the effects of food density on prey (rabbits) in a predator free environment. In both simulations 500 rabbits were placed with differing amounts of berry bushes, with 75 berry bushes in the sparse scenario and 1000 berry bushes in the abundant scenario. In the second test, the effects of predators hunting prey were examined. In both simulations 750 rabbits and 750 berry bushes are placed. In the predator scenario 500 wolves are added, whilst in the control no wolves were added. In the third test, the effects of the two different predators were compared. Since bears are omnivores and the wolves are carnivores, the rabbits might therefore respond differently. In both simulations 500 rabbits, 400 berry bushes, and either 100 wolves or 100 bears were placed.

Finding that changes to the simulations produce meaningful changes to the genomes of the animal populations would be considered a success in indicating that there

is an amount of evolution occurring in the simulation. Due to this definition of success, each test is considered done once the population number in the simulation has stabilised.

3.7 Performance Benchmarks

The performance results were obtained using two different computers, one of them being a desktop computer and the other being a laptop, with desktop having more powerful components than the laptop. The exact specifications are listed in Table 3.2. The desktop and laptop computers will hereby be referred to as configuration A and configuration B, respectively.

	Configuration A (Desktop)	Configuration B (Laptop)
CPU	Intel i7-9700K @ 3.6 GHz	Intel i7-8550U @ 1.8 GHz
GPU	Nvidia GeForce RTX 2070 Super	Nvidia GeForce MX150
RAM	16 GB DDR4 @ 3,000 MHz	16 GB DDR3 @ 1,867 MHz
Video memory	8 GB GDDR6	2 GB GDDR5
Operating system	Windows 10 20H2	Windows 10 20H2
Monitor refresh rate	240 Hz	60 Hz
Unity version	2020.2.1f1	2020.2.6f1

Table 3.2: Hardware and software specifications of the computers used for benchmarking the simulation.

All frame rate data was obtained by running the relevant scene for 45 seconds. The minimum, maximum and average frame rates were based on frame rate data that was updated three times per second, where only the last 100 values were used, in order to avoid anomalies in association with the initialisation of the scenes (especially with the ECS scene). Furthermore, all benchmarks were run using the default graphics settings, i.e. SMAA was used for anti-aliasing and ambient occlusion was enabled at 25% intensity.

4

Results

This chapter presents the results of the project, including an overview of the visual aspects of the virtual ecosystems, investigations into ecological aspects and performance benchmarks of simulation runs.

4.1 Visuals

This section provides an overview of the graphical user interface (GUI) and the scenes that represent different ecosystems in the final version of the application.

4.1.1 Main Menu

When running the simulation, a main menu lets the user select which simulation scene that will be run, as well as configure graphical quality and genomes. See Figure 4.1 for a screenshot of the main menu. Certain scenes, such as the dynamic and ECS scenes, will let the user specify the initial number of animals and resources such as food and water.

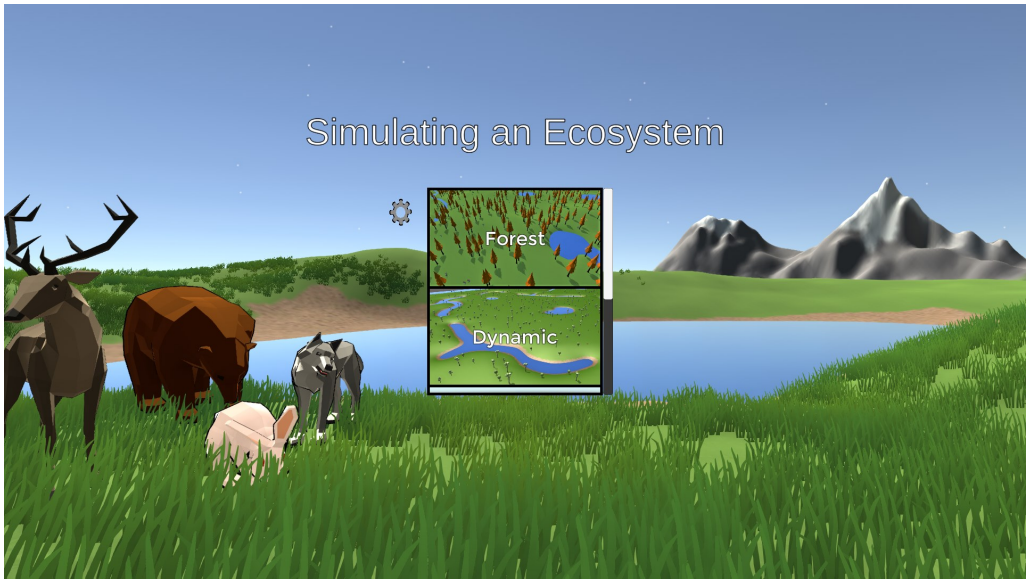


Figure 4.1: The main menu of the simulation. The cogwheel button will navigate to the settings menu which enables the user to change the graphics as well as how the genomes work for the different animals. Furthermore, the central "cards" can be scrolled, and let the user choose the scene that will be simulated.

4.1.2 Graphics Settings

The initial settings menu provides graphical options for anti-aliasing and ambient occlusion. Both anti-aliasing and ambient occlusion can be disabled entirely for improved performance. The genome settings are accessed by pressing the "Gene settings" label. See Figure 4.2 for a screenshot of the settings menu.

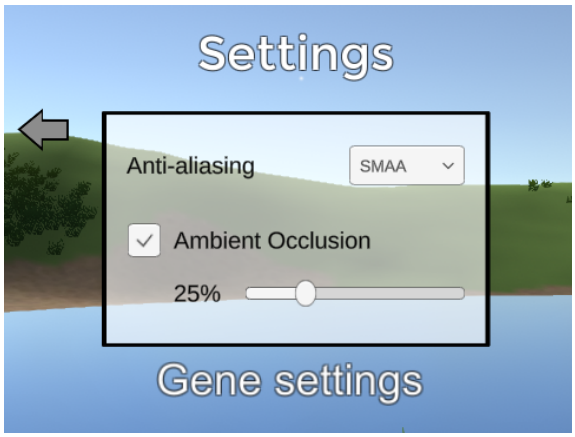


Figure 4.2: The graphics settings available in the simulation.

4.1.3 Real-time Animal Information

To aid understanding and insight into the behaviour of the animals in the simulation, all animals feature graphical indicators that provide information about the state, gender, hunger, thirst and stamina of the animal. Additionally, pregnant animals feature purple gender icons. See Figure 4.3 for an example of real-time information provided about animals.

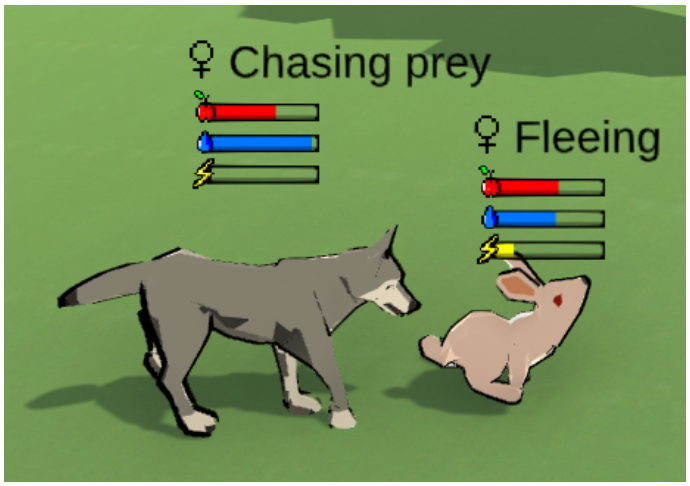


Figure 4.3: A wolf chasing a rabbit in the forest scene. Note the state indicators, the wolf is chasing a prey, i.e. the rabbit, whilst the rabbit is actively fleeing from the wolf. Furthermore, both animals are low on stamina (indicated by the "lighting" bars), due to the chase.

4.1.4 Logging Information

In addition to recording simulation events and storing them in log files, some information is displayed in real-time when running the simulation. This information includes the current frame rate, the duration of the simulation, the number of animals, the number of births, the number of deaths, the amount of food resources, the amount of mating and the amount of consumed prey. This is displayed in the upper left corner of the screen. Overall, this information is useful to quickly make assessments about simulations without having to look at the generated log files. See Figure 4.4 for a screenshot of how the information is presented in the simulation.

Figure 4.4: The real-time information provided about the current simulation.

4.1.5 Carcasses

When an animal dies in the simulation, either from being hunted, starvation or thirst, a piece of meat is placed at the position where the animal died, to emulate a carcass. This can subsequently be consumed by both omnivores and carnivores, i.e. bears and wolves. This is represented by a piece of meat with a bone, see Figure 4.5 for a screenshot of how it looks in the simulation.

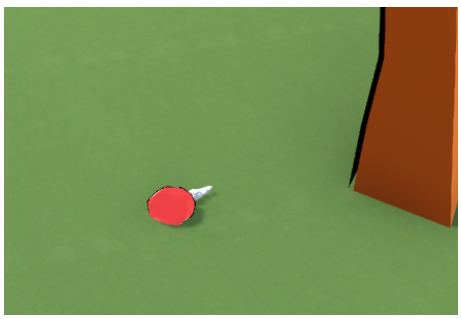


Figure 4.5: A carcass in the forest scene, as the result of an animal dying.

4.1.6 Forest Scene

The forest scene is a small non-ECS scene, having the dimensions 200×200. It was the first scene added to the simulation, and serves as a small-scale testing ground.

The animal and food amounts are not configurable in the forest scene, instead the scene always starts off with a total of 104 animals. See Figure 4.6 for a screenshot of the forest scene.

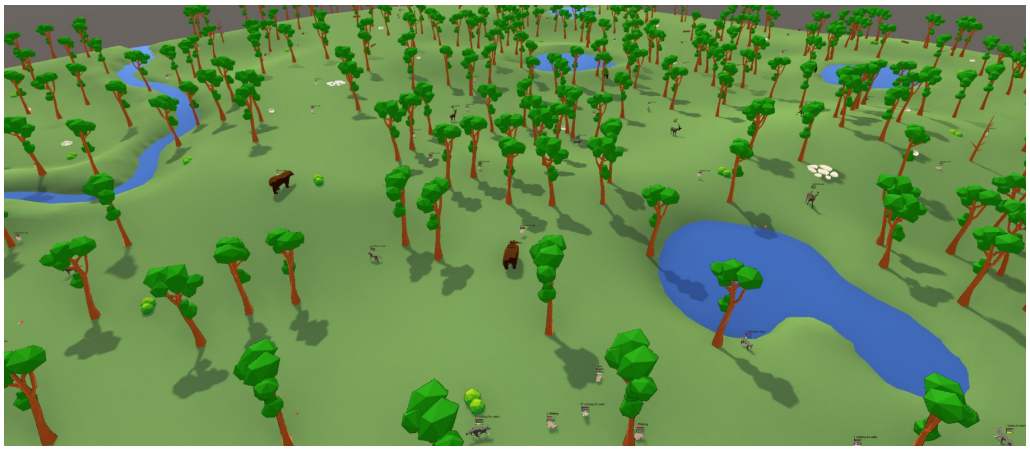


Figure 4.6: Screenshot of the forest scene.

4.1.7 Dynamic Scene

The dynamic scene is the largest non-ECS scene provided in the simulation (with the dimensions 750×750 in Unity). It features open and flat terrain, with numerous lakes and rivers. The number of animals and food is configurable in the dynamic scene (hence the *dynamic* in the name). See Figure 4.7 for a screenshot of the dynamic scene.

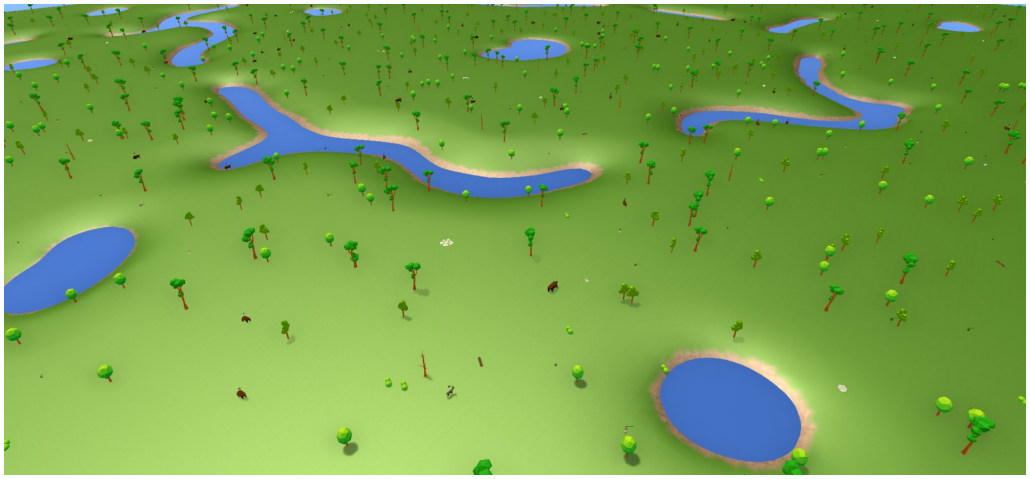


Figure 4.7: Screenshot of the dynamic scene, with its numerous water sources and open terrain.

By default, the dynamic scene is configured to spawn 100 rabbits, 100 deer, 100 wolves, 100 bears, 100 carrots and 100 berry bushes. The amount of any animal of food resource is limited to the interval $[0;\ 9,999]$. See Figure 4.8 for a screenshot of the configuration menu for the dynamic scene.

Figure 4.8: The configuration menu for the dynamic scene, with its default values.

4.1.8 ECS Scene

The ECS scene is by far the largest scene, with the dimensions 3,000×3,000. Due to limited NavMesh support using the ECS framework, the terrain is completely flat. For the same reason, the animals in the scene feature no animations. However, the scene does provide insight into the performance potential of the ECS framework in Unity. See Figure 4.9 for a screenshot of the ECS scene.

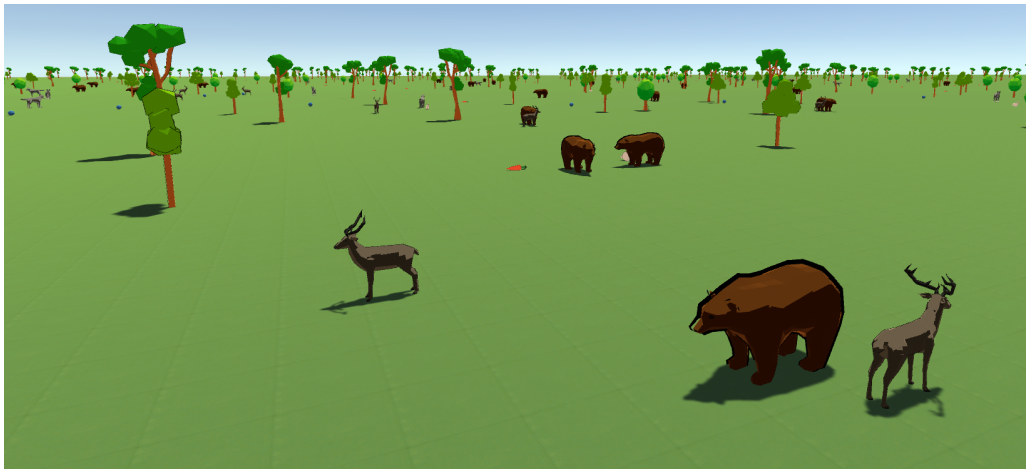


Figure 4.9: Screenshot of the ECS scene. Note the completely flat terrain and lack of animations (all animals are always in their default "poses").

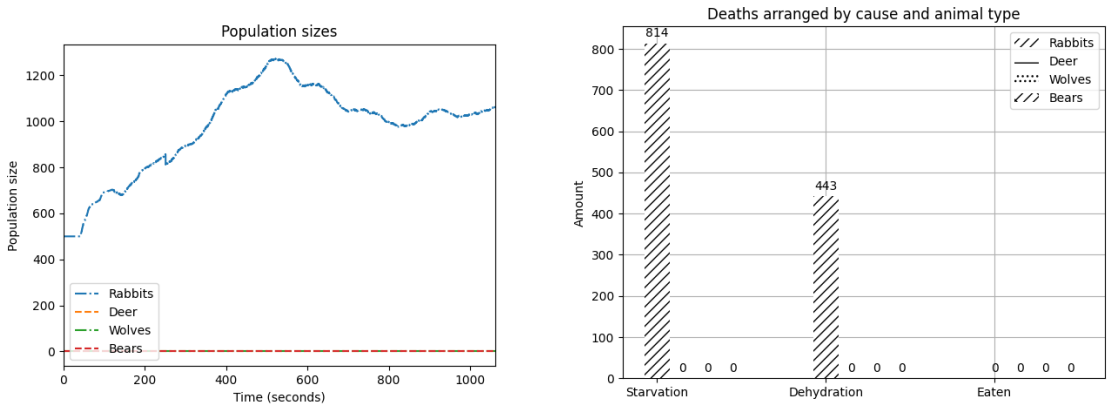
4.2 Carrying Capacity

The simulated ecosystems feature many factors that affects the overall balance and carrying capacity of the different species. This section presents limiting factors related to the carrying capacity. Section 3.5 explains the methods used for performing the simulations used in this section.

4.2.1 Near Unlimited Food Supply Without Predators

Given near unlimited availability of food and no predators, the rabbit population will grow steadily and then seems to stabilise around a carrying capacity (see Figure 4.10). Whilst most rabbits still succumb due to starvation, the amount of deaths by thirst is still a bigger share of the total then compared to 4.11 (b). This is likely due to the fact that water sources are relatively scarce compared to food resources.

The simulation shows that it is the availability of berries on the berry bushes that can be found by the rabbits, possibly close to water sources, that are the true limiting factor to carrying capacity.

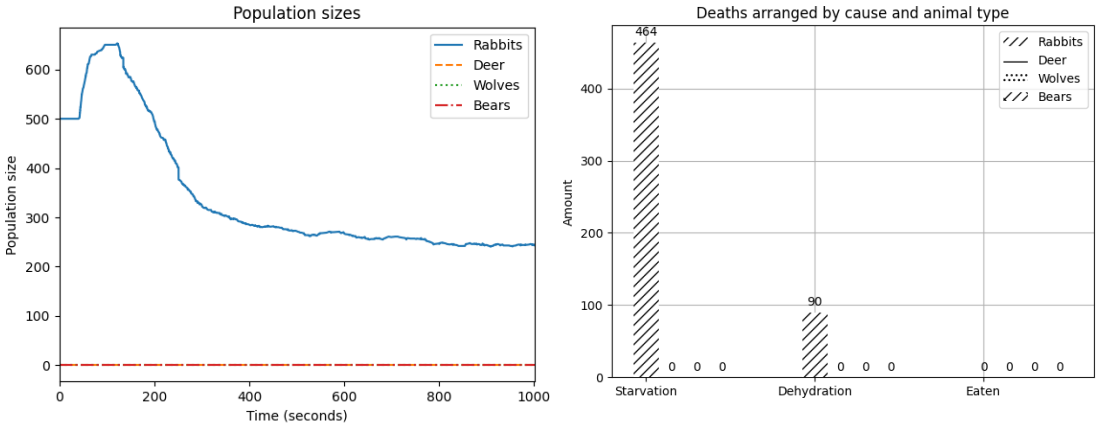


(a) (b)

Figure 4.10: Population data (a) and cause of death (b) from simulation of 500 rabbits and 2,000 berry bushes running for 1,000 seconds.

4.2.2 Limiting Food Supply

If food availability is lower the population size will stabilise around a carrying capacity for the current ecosystem. The left diagram in Figure 4.11 shows the population where the amount of food has been lowered. The population starts with a rapid growth but as time goes on there appears to be a struggle for food and the population size decreases rapidly. The right diagram in the same figure shows the cause of death where dehydration still accounts for a similar number of deaths but starvation has become a more prominent cause.



(a) (b)

Figure 4.11: Population data (a) and cause of death (b) from simulation of 500 rabbits and a smaller food supply.

However, despite the fact that the amount of food increases steadily, as can be seen in Figure 4.12, some rabbits still continue to die of starvation. A likely cause for this is that lowering the number of bushes makes it harder for the rabbits to actually find the food even though there is food available in the ecosystem.

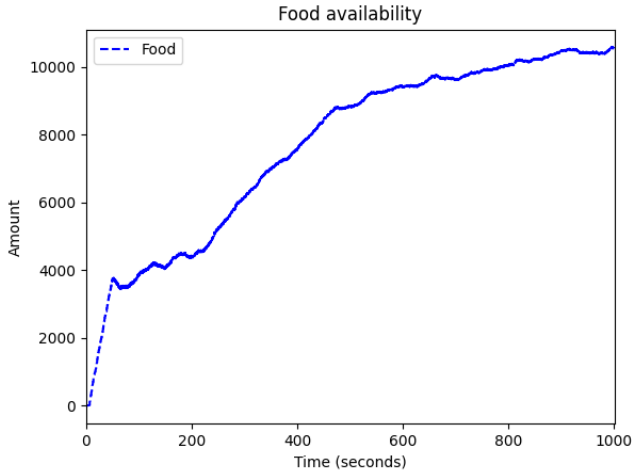
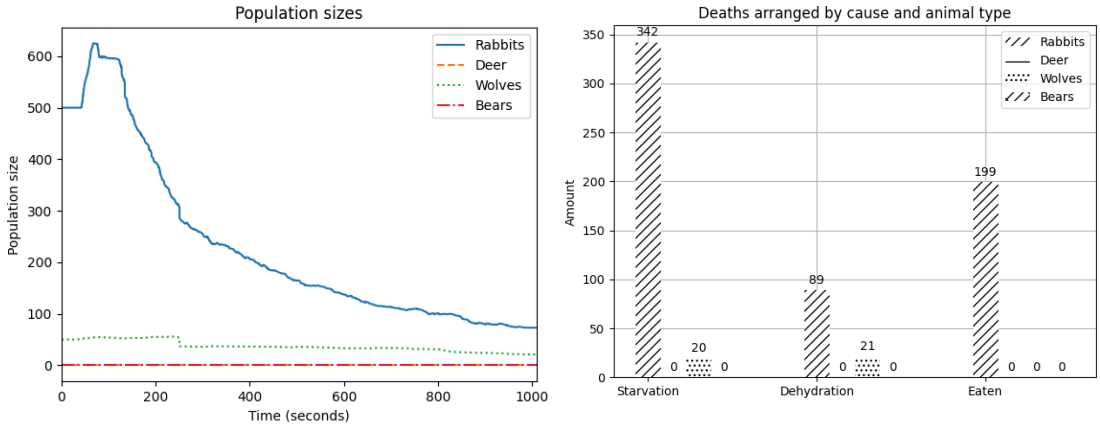


Figure 4.12: Amount of available food in simulation of 500 rabbits and 400 berry bushes in dynamic scene.

4.2.3 Introducing Predators

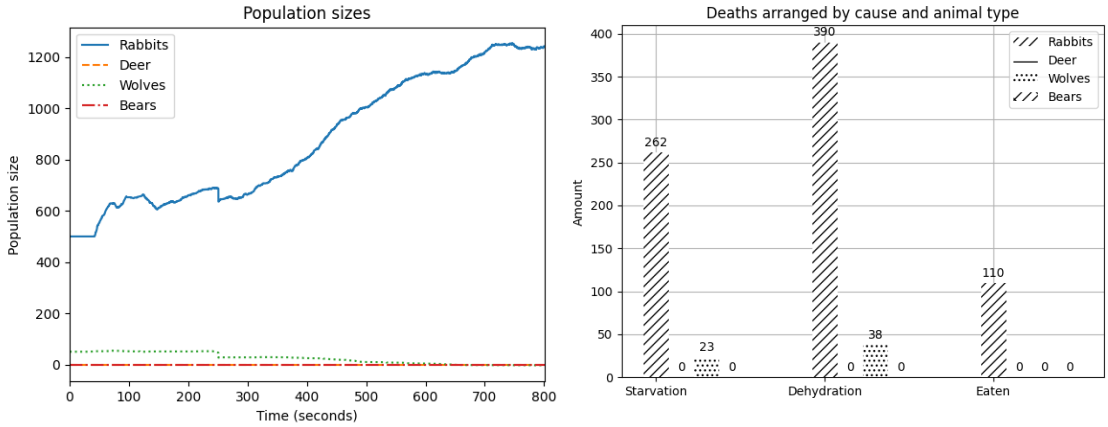
The most important limiting factor in the simulated ecosystems for prey are predators. Introducing a few predators will lower the carrying capacity for rabbits significantly. Figure 4.13 depicts the population curve and cause of death when a few wolves have been introduced into the ecosystem.



(a) (b)

Figure 4.13: Population data (a) and cause of death (b) from simulation of 500 rabbits, 400 berry bushes and 50 wolves

When simulating the ecosystem and increasing the amount of berry bushes a new trend emerges that can be seen in the left population diagram in Figure 4.14. The wolves go extinct and the rabbits thrive, likely due to the fact that are fewer rabbit carcasses for the wolves to eat. This forces the wolves to survive of hunting, which is difficult.



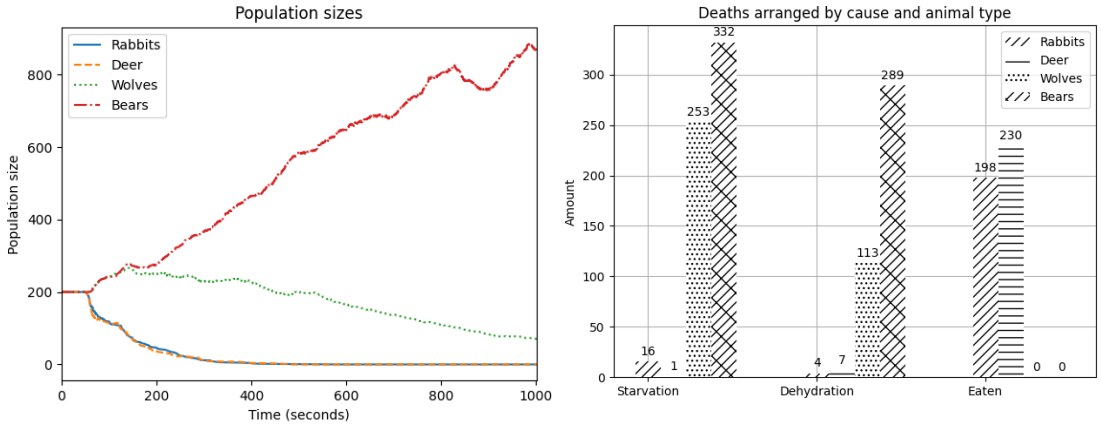
(a) (b)

Figure 4.14: Population data (a) and cause of death (b) from simulation of 500 rabbits, 2,000 berry bushes and 50 wolves.

4.2.4 Adding all animals

Simulating ecosystems containing all animals using the same amount of each species leads to unstable systems, where the population sizes of rabbits, deer and wolves decrease rapidly whilst the amount of bears steadily increase. The wolves mainly succumb to starvation when a large amount of prey have hunted and subsequently

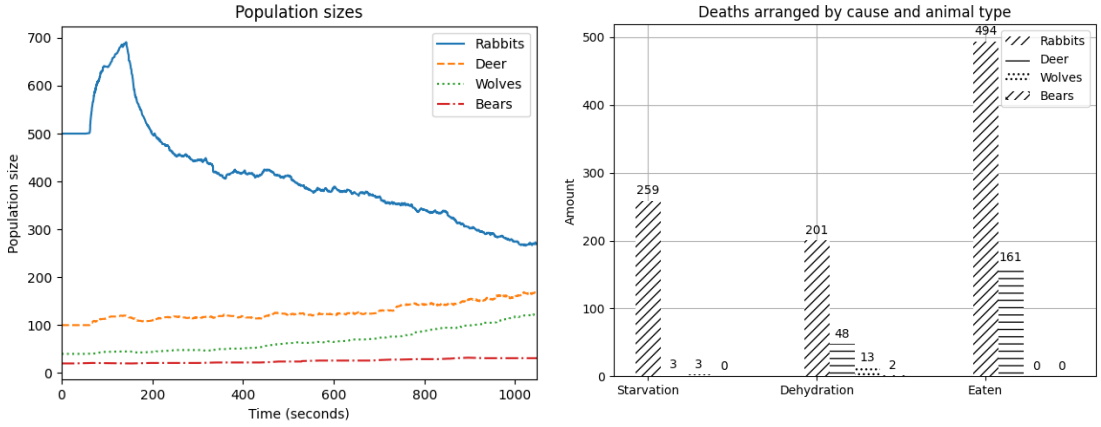
killed. Figure 4.15 depicts population sizes and cause of death in a simulation where the population sizes are equal in the beginning.



(a) (b)

Figure 4.15: Population data (a) and cause of death (b) from simulation of 200 rabbits, 200 deer, 200 wolves, 200 bears and 500 berry bushes.

With some balancing, more stable ecosystems can be simulated. This involves having fewer predators and larger prey populations. Results from using more balanced population sizes is shown in Figure 4.16.



(a) (b)

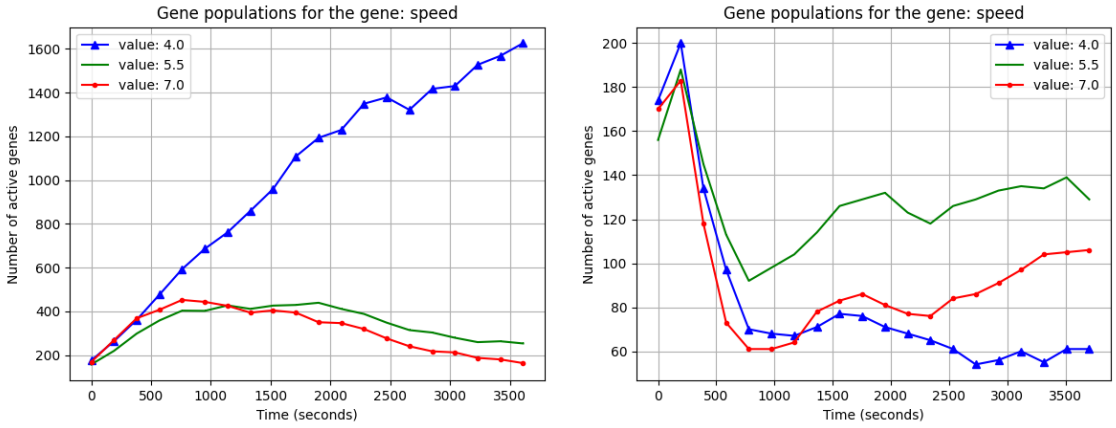
Figure 4.16: Population data (a) and cause of death (b) from simulation of 500 rabbits, 100 deer, 50 wolves, 20 bears and 400 berry bushes and predators.

4.3 Genes and Evolution

In this section, the results of the tests described in section 3.6 are presented.

4.3.1 Food Density

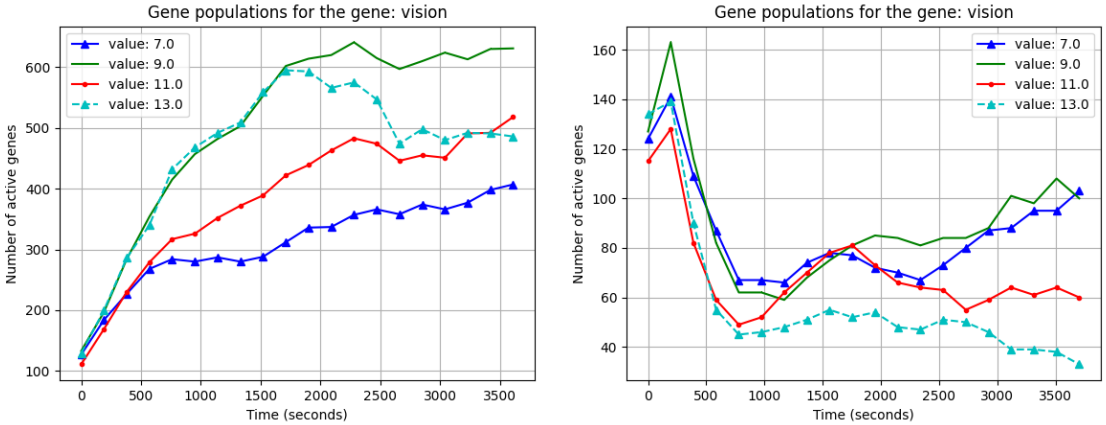
As specified in 3.6, sparse versus abundant food is compared in a scenario containing only rabbits (no predators, only prey). Comparing the results of both simulations (each taking approximately 3,600 seconds), it was found that speed, vision, and gestation period differ. As can be seen in Figure 4.18 short vision (value 7.0) is successful in the sparse environment. Low speed is significantly more prevalent in the plentiful scenario.



(a) Speed (Abundant food)

(b) Speed (Sparse food)

Figure 4.17: Speed gene value populations for 500 initial rabbits with (a) abundant (1000 berry bushes) and (b) sparse (75 berry bushes) food.

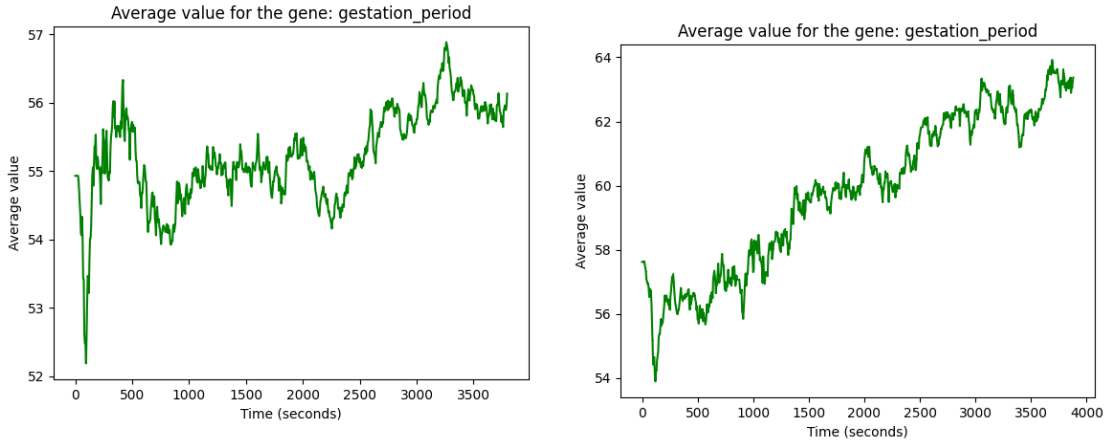


(a) Vision (Abundant food)

(b) Vision (Sparse food)

Figure 4.18: Vision gene value populations for 500 initial rabbits with (a) abundant (1000 berry bushes) and (b) sparse (75 berry bushes) food.

When comparing the gene populations (total number of present genes sorted by value) for gestation period, no clear change could be found. However, a noticeable difference is apparent upon examination of the average values instead. As seen in Figure 4.19, higher values for the time spent in gestation are favoured when food is sparse.



(a) Average gestation period (abundant food).

(b) Average gestation period (sparse food).

Figure 4.19: Average values of the gestation period gene of 500 initial rabbits for (a) abundant (1000 berry bushes) and (b) sparse (75 berry bushes) food.

4.3.2 Presence of Predators

Due to wolves being ineffective predators, in this case due to the fact that they go extinct early in the simulation, no information can be extracted from this test. There are no meaningful results produced from this result, see Figure 4.20.

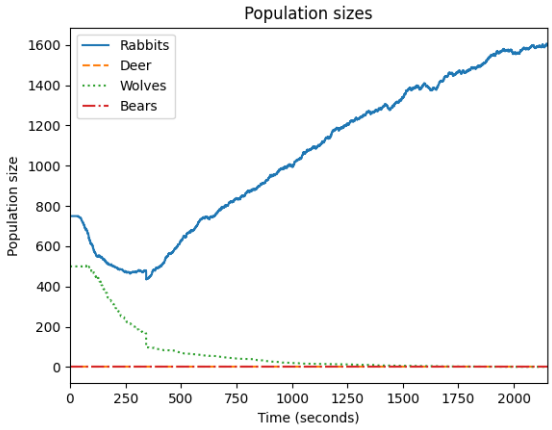
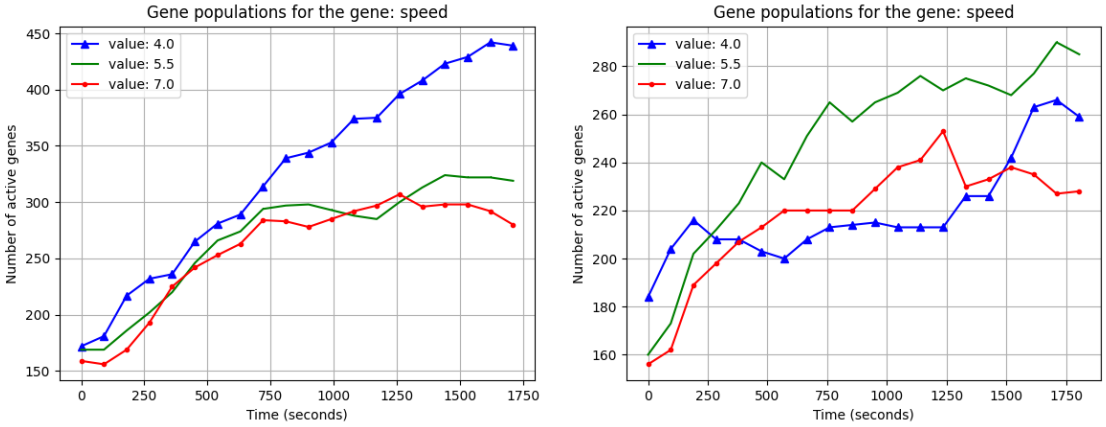


Figure 4.20: Changes in population size for 750 initial rabbits, 500 initial wolves, and 750 berry bushes over 2000 seconds. Note the extinction of wolves after approximately 1000 seconds. Note the increase in (b).

4.3.3 Different Predators

When comparing the effects of wolves and bears a similar issue to the previous section 4.3.2 is found. Since the wolves go extinct early in the simulation, this test is more akin to a predator versus no predator test. Each simulation was run for approximately 1,750 seconds. As shown in Figure 4.21, rabbits favour lower speeds when wolves are present compared to without bears.



(a) Rabbit speed gene values with 100 wolves.

(b) Rabbit speed gene values with 100 bears.

Figure 4.21: Speed gene populations changes for 500 initial rabbits with either (a) 100 initial wolves or (b) 100 initial bears.

4.4 Performance

This section discusses the overall performance of the simulation, primarily regarding frame rate, measured in frames per second (FPS). The results in this section were obtained according to section 3.7.

4.4.1 Conventional Game Object Approach

The main scene for the simulation is the dynamic scene, which features much more intricate logic and behaviours compared to the ECS scene. However, it makes use of the conventional object-oriented game object workflow in Unity, which inherently limits its performance potential. The dynamic scene is by default configured spawn a total of 600 game objects, of which 400 are "active" objects. See Table 4.1 in the appendix for the performance of the dynamic scene using the default distribution.

	Minimum FPS	Maximum FPS	Average FPS
Configuration A	25.5	39.1	32.6
Configuration B	15.0	30.1	22.8

Table 4.1: Minimum, maximum and average frame rates obtained when running the dynamic scene with the default distribution of entities.

Performance Degradation

The performance degradation in the dynamic scene is shown in Figure 4.22. The amount of game objects was evenly distributed for each benchmark. Overall, the performance degradation appears to be relatively linear. For instance, the relative decrease in average FPS from 1,000 to 2,000 game objects is 52.7 %, which is close to 50 %. See Table B.1 in the appendix, for the exact frame rate data.

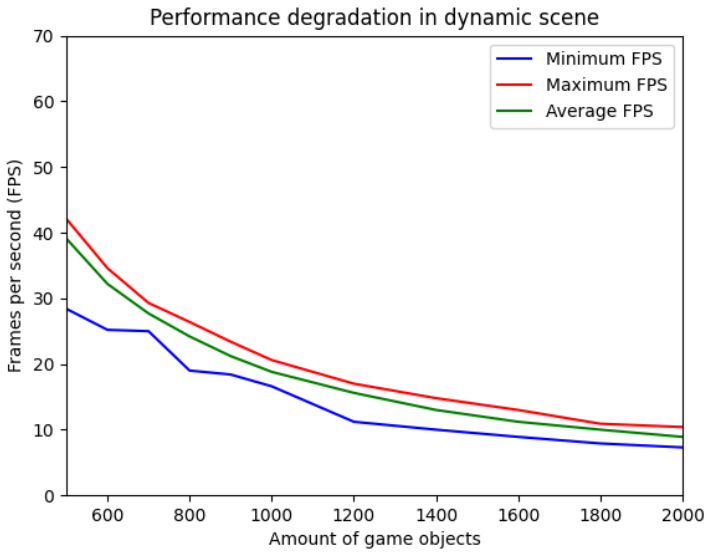


Figure 4.22: Plot of the performance degradation in the dynamic scene. Configuration A was used to obtain these results.

4.4.2 ECS Framework

The "ECS Demo" demo scene utilises the Unity ECS framework. However, it does not feature the same level of detail as other scenes that do not use the ECS framework. Nevertheless, the scene does provide an insight into the potential of ECS in relation to more traditional object-oriented approaches. By default, the ECS scene is configured to spawn 1,500 rabbits, 700 deer, 450 wolves, 250 bears, 2,000 carrots and 2,000 water objects. These add up to a total of 6,900 entities. This number of entities is effectively impossible to simulate in the other non-ECS scenes.

	Minimum FPS	Maximum FPS	Average FPS
Configuration A	32.8	240.0	74.9
Configuration B	12.0	60.0	26.1

Table 4.2: Minimum, maximum and average frame rates obtained when running the ECS scene with the default distribution of entities.

Performance Degradation

Figure 4.23 shows how the performance degrades in the ECS scene. See Table B.2 for the underlying frame rate data. A distribution of 50 % rabbits and 50 % wolves

was used for the benchmarks. The maximum FPS is capped at 240 FPS due to the fact that the frame rate cannot exceed the refresh rate of the monitor, so the potential maximum FPS might be slightly higher. Compared to the dynamic scene, the difference between the maximum and minimum FPS is greater using the ECS framework, even if the overall performance is much better. An apparent result when comparing the frame rate data from the ECS and dynamic scene is that the performance of the ECS scene seems to improve when the entities are more evenly distributed. Since the performance of the ECS scene using the default distribution (using configuration A), which uses a total of 6,900 entities, is comparable to the performance of running 3,000 entities, where 50 % are rabbits and 50 % are wolves.

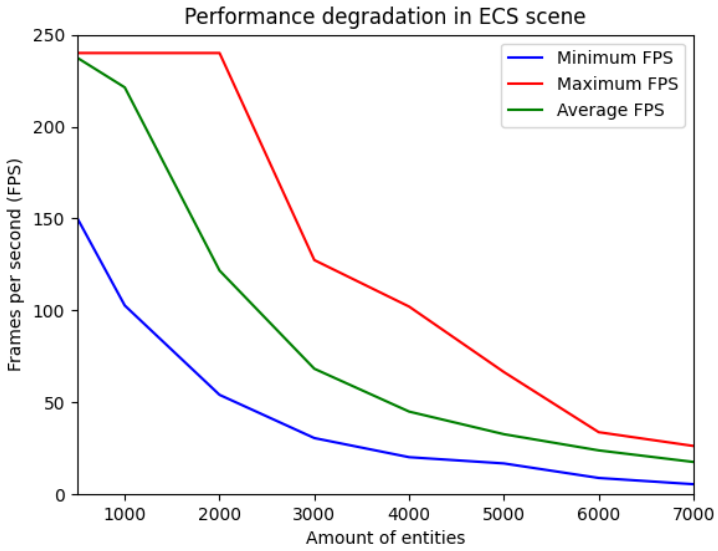


Figure 4.23: Plot of the performance degradation in the ECS scene. Configuration A was used to obtain these results.

5
Discussion

This chapter discusses the final state of the simulation, possible improvements and other results. Additionally, societal and ethical aspects of the project are discussed.

5.1 Thoughts on the Simulation Results

When evaluating the results of the carrying capacity tests and evolution test, we find that the simulation works adequately. The system reaches the expected level of advancement for the scope of the project. It does however leave room for improvement.

5.1.1 Factors Affecting Carrying Capacity

Through testing several setups some interesting patterns emerged. Many patterns and results were briefly discussed in 4.2 but will be expanded upon here.

Unlimited Food

Tests with ecosystems that provided near unlimited food showed that, when rabbits were not required to struggle with finding food and could spend more time mating, the population skyrocketed. However, according to Figure 4.10, there appears to be a population ceiling. This might be caused by the size of the ecosystem, which could limit the amount of rabbits the scene can support since many rabbits succumbed to thirst as they failed to locate water sources. Another factor was the system's ability to run a large amount of entities, one limitation with the unity navigation system is its limit on how many entities can compute their paths at the same time. This meant that in our case, numerous entities froze while waiting for their turn to calculate paths making it even harder for them to traverse the map and find resources to survive.

Stability

It became clear that the graph obtained when running the simulation for 1000 seconds didn't resemble a stable system, at least not when predators were introduced. This could be improved upon by tweaking the stats of the wolf and rabbit more and to obtain stability when all animals are present even more testing needed to be done. But with the outset to have the genes be the same for all animals, the problem of

stability had to be solved by adjusting the starting population. But overtime this tended to not be stable.

5.1.2 Evolution of Genes

As can be seen in the results of the tests 4.3.1 and 4.3.3, changes in the initial conditions cause changes in the evolution of the animals within the simulation. As such, we can consider there to be good indication that evolution occurs in the system.

As mentioned in 5.1.1, one possible source of error is that because of the large population of entities due to the overabundance of food, the navigation system is not able to keep track of all the rabbits at the same time and the rabbits therefore spend a lot of time in place. This might have an effect on the evolution. For example, one possible reason why low values of the speed gene were favoured might be that the increased hunger rate that high speed values entail was too detrimental when the animals spent much time frozen in place. However, this should not have affected the results of the third test in 4.3.3, since in this case the population numbers remained similar throughout the simulation.

5.1.3 Maturity of Unity ECS Framework

Several problems were encountered when using the Unity ECS framework. The framework was, at the time of the development of the simulation, in a preview state. As a result, many features, such as the NavMesh navigation system, were not usable in the context of the ECS framework. Therefore, a third-party library [29] was required to get the basic navigation to work. Furthermore, due to the preview nature of the framework, many aspects of the typical Unity workflow changed dramatically, increasing complexity and made it difficult to port old code. Overall, the performance gains were promising, especially with regards to parallelisation, due to great support in the Unity ECS framework to easily and safely run systems in parallel. However, the difficulties with getting even basic Unity features to work with the ECS framework meant that the group chose to not use it other than in a "demo" scene.

5.2 Possible Improvements

There are many opportunities for improvements to the simulation, both with regards to the quality of the simulated behaviours but also with regards to the user experience of running the simulation.

5.2.1 Simulation Quality

One area for potential improvements could be increasing the quality of the simulation, either by increasing the reliability of the simulation or by introducing more complex and different aspects to simulate.

Behavioural Genes

Genomes could be given greater influence over the behaviour of animals. In the current simulation, genes only marginally affect the *traits*, e.g. speed or size, of animals, but they do not add or remove any behaviours. For example, a gene could be made to correspond to a specific strategy for hunting or fleeing, or perhaps to give animals the possibility to give birth to a litter of children, instead of only ever giving births to a single animal.

Increase Simulation Size

Expanding the total size of the simulation and the number of entities in it would be an improvement. It would give more stable results when simulating ecological theories since it would reduce the impact of error sources in the statistics. For this to be possible it would in essence involve enhancing the performance of the system, either by optimising the current implementation or by transitioning to a different engine/type of implementation. The latter being what the group has found as the preferable alternative, namely transitioning to full ECS. As seen in the ECS demo scene, the number of animals that can be supported far outperforms the current implementation making it unlikely that any amount of optimisation on the current implementation would achieve the same performance. This transition would almost certainly involve changing the engine to one that supports it or by waiting for Unity to build support for it.

An additional important factor related to the scalability of the simulation is the choice of programming language. A more low-level language, such as C++, would result in more control over the general performance of the simulations. The performance advantage from choosing a lower level language might come at the cost of a slower overall development pace, which might be worth it if the goal is to simulate larger ecosystems. Future projects aiming to simulate ecosystems should at least consider the scalability and performance issues during the early stages of development. For example, there are popular game engines that use C++, such as the Unreal engine, which could be used instead of Unity for simulating ecosystems.

5.2.2 User Experience

This section discusses aspects related to user experience that could be improved, such as support for multiple simulation runs without having to restart the simulation or providing visualisation of simulation results directly in the application.

Multiple Simulation Runs Per Session

Due to certain assumptions in the implementation, it is not currently possible to simulate more than one ecosystem configuration per run of the simulation application. The overall usability would benefit from letting users simulate multiple ecosystems with different configurations without having to restart the entire application.

Improve Visualisation of Data

In order to produce visualisations of the data obtained from simulating an ecosystem, the user is required to manually navigate to the generated log file and run the visualisation scripts. This experience could be improved significantly if it was also possible to generate the visualisations from within the application, since it would make the results more accessible.

Adding support for visualising the differences between multiple runs could improve what kind of conclusions could be drawn from the data. Additionally, different types of graphs could be included, such as heat maps of animal activity and death to look at hunting behaviours, genealogy trees to find exceptionally successful individuals, and generally more complex graphs.

5.3 Real World Comparisons

There is little to no possibility of computer simulations being able to fully simulate the real world with the same complexity. The strive for this project was never to achieve a virtual ecosystem that behaves like an ecosystem would in reality but for it to be a plausible approximation of the real world. With this said, even though simulation and models seldom are fully correct there is still real-world conclusions to be had, with this project being no exception. The developed ecosystem has the potential to be a learning tool and source of information into how complex the real world can be. Showing that animals that have a handful of variables to them can display a complex system while the user also gets tangible experience with ecology.

5.3.1 Balancing

A major challenge in working with a simplified model is how to balance the variables in a way that resembles the real world enough as to enable drawing some conclusions about the principles of ecosystems and natural selection. Furthermore, they need to be balanced in a way that makes said conclusions possible to draw within a suitable time frame and meaningful changes should happen within this time frame. Attributes such as hunger and thirst rate thus need to vary enough so that some animals die from being less adapted to the environment than others. Moreover, because the simulation is running for a relatively short time, they need to vary enough to counteract the randomness of the system, such as whether animals will cross path with food or not.

5.3.2 Realism of Attributes

Because of the limit on the number of entities discussed earlier, some values need to be made more significant than they would be in real life. For example, the vision attribute of animals allows some animals to have a field of view twice the size of others. Unfortunately, this strays from reality quite a bit, and runs the risk of leaving the survivability of different phenotypes up to chance by too much. Nevertheless,

while longer simulation times and more entities would allow for more accurate balancing, the current system still works as a tool for exploring the principles of natural selection and ecosystems.

5.4 Societal and Ethical Aspects

The benefits of conducting this project included improving our understanding of the underlying mechanisms of evolution and natural selection. Our goal was to create a simulation tool that is interesting, pedagogical and provides insights to people outside of our project group. This can help to improve the understanding of these natural processes for people other than ourselves. Furthermore, in a time when many animal species are in danger of going extinct [30], it could be interesting to see if we could, for example, create a simulation that demonstrates how animal populations can change as a result of only small changes to their initial sizes, to encourage greater care of our environment.

5.4.1 Risk of Incorrect Conclusions

There is almost always a risk of making incorrect conclusions from a set of data. This is something which were kept in mind and monitored during the evaluation of results of the simulation, since the simulation consisted of a simplified model whose results cannot be directly applied to the real world, even if they are useful for understanding the underlying concepts. For example, if we find that a population is stable at a size of 50 animals in our simulation, that cannot be interpreted as real-life animal populations are fine with only 50 living examples. Likewise, if the genes of the animals converge around certain values through natural selection, it does not necessarily mean that those are the values that would work best or at all for the attributes of animals in the real world.

5.4.2 Risk of Damages

We find that there are no direct risks related of material or psychological damages associated with the project, since no humans, animals or ecosystems were put in harm's way through the execution of the project. Furthermore, the simulation did not make use of photo-realistic graphics and did not depict any graphic or gruesome violence, so it should not be any issues related to people using the simulation and watching the results unfold.

5.4.3 Privacy

The simulation does not collect user credentials or information of any kind, so privacy is not an issue.

5.4.4 Fairness

The simulation depicts different animals with attributes assigned to fit their "role" in a real ecosystem. For example, a bear hunts rabbits and are aggressive. However, the attributes assigned to different animals are arbitrary and/or chosen to measure different outcomes. As a result, there might be worries that some animals are "misrepresented", but as explained earlier, the simulation is not to be taken too literally, since the underlying processes and mechanisms are the interesting aspects.

6
Conclusion

It is possible to utilise the application to simulate different configurations of ecosystems and evaluate results of different factors. Caution should be exercised before drawing conclusions from the obtained results in relation to real ecosystems. Simulating ecosystems is complicated and there are many subtle details that affect the end result. However, the simulation software is capable of providing a visually interesting experience that provides some basic insights into how ecosystems can be affected by tweaking different initial factors.

As presented in section 4.4, the performance of the conventional object-oriented Unity frameworks limited the potential scale of the simulated ecosystems. The Unity ECS framework did show potential to mitigate the performance limitations, but it caused problems due to the fact that many features, such as NavMesh agents or triggers, were not officially supported at the time of development. Future projects that aim to simulate large scale ecosystems, i.e. ecosystems with a few thousand entities, should consider performance early during the development cycle. Using an ECS architecture is recommended, but utilising more low-level programming languages, such as C++, could simplify the optimisation process significantly. Since the group spent quite some time trying to avoid some of the innate performance deficits associated with languages such as C#.

Overall the results meet the goals (to at least sufficient levels) within the constraints of the project. The results of the evolution and carrying capacity tests, while not perfect, were satisfactory. Simulating biological aspects such as genomes and mutations can be challenging especially when some conclusions are desired from the system. There is room for improvement in many aspects of the system, as described in section 5.2, such as reliability and complexity. However, as a prototype, or first iteration, project, the overall outcome can be considered a success.

Bibliography

[1] S. J. Metcalf, J. M. Reilly, A. M. Kamarainen, J. King, T. A. Grotzer, and C. Dede, "Supports for deeper learning of inquiry-based ecosystem science in virtual environments-comparing virtual and physical concept mapping," *Computers in Human Behavior*, vol. 87, pp. 459–469, 2018.

[2] K. Knezevic. (2021). "GTA 5 has sold more than 140 million copies," [Online]. Available: `https://www.gamespot.com/articles/gta-5-has-sold-more-than-140-million-copies/1100-6487267/` (visited on 04/20/2021).

[3] U. Ali. (2018). "Red dead redemption 2: Virtual ecology is making game worlds eerily like our own," [Online]. Available: `https://theconversation.com/red-dead-redemption-2-virtual-ecology-is-making-game-worlds-eerily-like-our-own-107068` (visited on 04/20/2021).

[4] Rami Ismail. (2018). "Equilinox presskit," [Online]. Available: `https://equilinox.com/presskit/sheet.php?p=equilinox#factsheet` (visited on 05/13/2021).

[5] T. N. Romanuk, A. Binzer, N. Loeuille, W. M. A. Carscallen, and N. D. Martinez, "Simulated evolution assembles more realistic food webs with more functionally similar species than invasion," *Scientific Reports*, vol. 9, no. 1, Dec. 2019. DOI: `10.1038/s41598-019-54443-0`. [Online]. Available: `https://doi.org/10.1038/s41598-019-54443-0`.

[6] A. Grafen, "Modelling in behavioural ecology," *Behavioural ecology: an evolutionary approach*, vol. 3, pp. 5–85, 1991.

[7] Primer. (2018). "Why do things exist? setting the stage for evolution.," [Online]. Available: `https://www.youtube.com/watch?v=oDvzbBRiNlA&list=PLKortajF2dPBWMIS6KF4RLtQiG6KQrTdB` (visited on 02/22/2021).

[8] N. Lassabe, H. Luga, and Y. Duthen, "Evolving creatures in virtual ecosystems," in *Advances in Artificial Reality and Tele-Existence*, Springer Berlin Heidelberg, 2006, pp. 11–20. DOI: `10.1007/11941354_2`. [Online]. Available: `https://doi.org/10.1007/11941354_2`.

[9] M. D. Travers, "Agar–an animal construction kit," Ph.D. dissertation, Massachusetts Institute of Technology, 1988.

[10] ——, "Programming with agents new metaphors for thinking about computation," Ph.D. dissertation, Massachusetts Institute of Technology, 1996.

[11] H. V. D. Parunak, R. Savit, and R. L. Riolo, "Agent-based modeling vs. equation-based modeling: A case study and users' guide," in *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, Springer, 1998, pp. 10–25.

[12] P. Hogeweg and B. Hesper, "Individual-oriented modelling in ecology," *Mathematical and Computer Modelling*, vol. 13, no. 6, pp. 83–90, 1990.

[13] Unity Technologies. (2021). "Important Classes - GameObjects," [Online]. Available: https://docs.unity3d.com/Manual/class-GameObject.html (visited on 05/10/2021).
[14] ——, (2020). "Colliders," [Online]. Available: https://docs.unity3d.com/Manual/CollidersOverview.html (visited on 04/05/2021).
[15] ——, (2021). "Inner workings of the navigation system," [Online]. Available: https://docs.unity3d.com/Manual/nav-InnerWorkings.html (visited on 03/03/2021).
[16] Y. Avissar, J. Choi, J. DeSaix, V. Jurukovski, R. Wise, C. Rye, *et al.*, "Biology: Openstax," 2018.
[17] T. E. o. E. Britannica. (2020). "Ecosystem," [Online]. Available: https://www.britannica.com/science/ecosystem (visited on 03/03/2021).
[18] I. A. Hatton, K. S. McCann, J. M. Fryxell, T. J. Davies, M. Smerlak, A. R. Sinclair, and M. Loreau, "The predator-prey power law: Biomass scaling across terrestrial and aquatic biomes," *Science*, vol. 349, no. 6252, 2015.
[19] T. E. o. E. Britannica. (2020). "Carrying capacity," [Online]. Available: https://www.britannica.com/science/carrying-capacity (visited on 05/14/2021).
[20] L. learning. (2016). "Environmental limits to population growth," [Online]. Available: https://courses.lumenlearning.com/boundless-biology/chapter/environmental-limits-to-population-growth/ (visited on 04/07/2021).
[21] N. S. Foundation. (2011). "The most genes in an animal? tiny crustacean holds the record," [Online]. Available: https://www.nsf.gov/news/news_summ.jsp?cntn_id=118530 (visited on 03/02/2021).
[22] M. Mahner and M. Kary, "What exactly are genomes, genotypes and phenotypes? and what about phenomes?" *Journal of theoretical biology*, vol. 186, no. 1, pp. 55–63, 1997.
[23] F. J. Ayala and M. Coluzzi, "Chromosome speciation: Humans, drosophila, and mosquitoes," *Proceedings of the National Academy of Sciences*, vol. 102, no. suppl 1, pp. 6535–6542, 2005.
[24] MedlinePlus. (2020). "What is a gene mutation and how do mutations occur?" [Online]. Available: https://medlineplus.gov/genetics/understanding/mutationsanddisorders/genemutation/ (visited on 03/03/2021).
[25] J. M. Smith and S. J. Maynard, *The theory of evolution*. Cambridge University Press, 1993, p. 26.
[26] R. C. Lewontin, "The units of selection," *Annual review of ecology and systematics*, pp. 1–18, 1970.
[27] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.
[28] Unity Technologies. (2021). "Unity Toon Shader," [Online]. Available: https://github.com/Unity-Technologies/com.unity.toonshader (visited on 05/05/2021).
[29] R. Schultz. (2020). "ReeseUnityDemos," [Online]. Available: https://github.com/reeseschultz/ReeseUnityDemos (visited on 04/19/2021).

[30] WWF. (2021). “Species list,” [Online]. Available: https://www.worldwildlife.org/species/directory?direction=desc&sort=extinction_status (visited on 02/22/2021).

A

State Diagrams

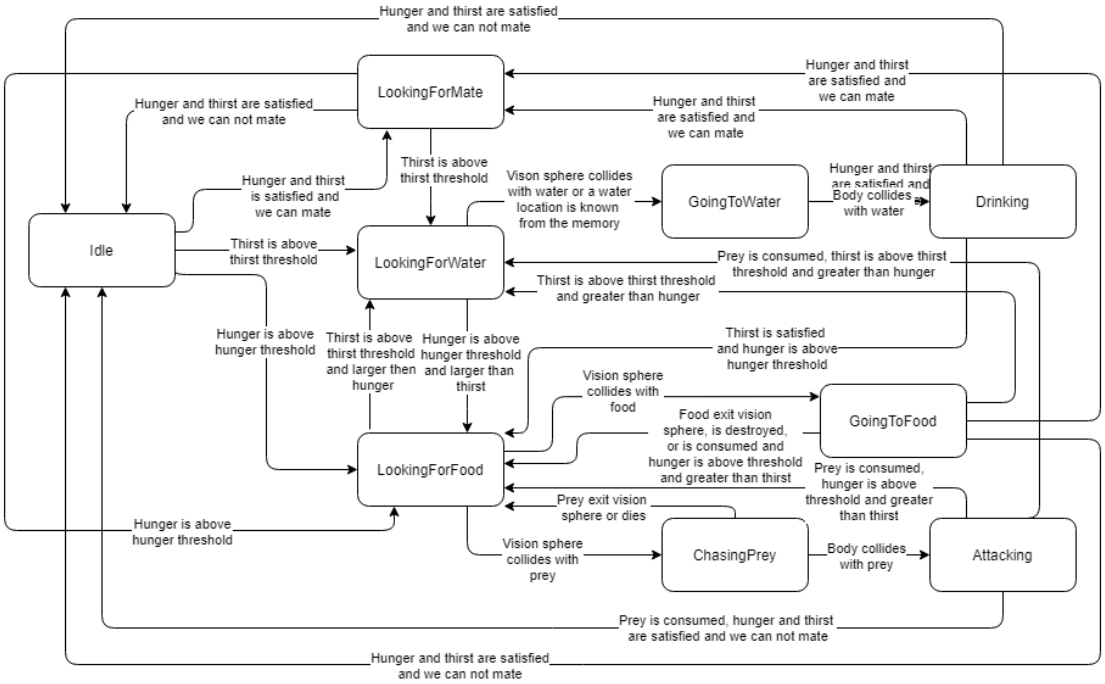


Figure A.1: A state diagram describing the state machine used by wolves. The diagram features the states of the state machine and the state transitions with associated transition conditions.

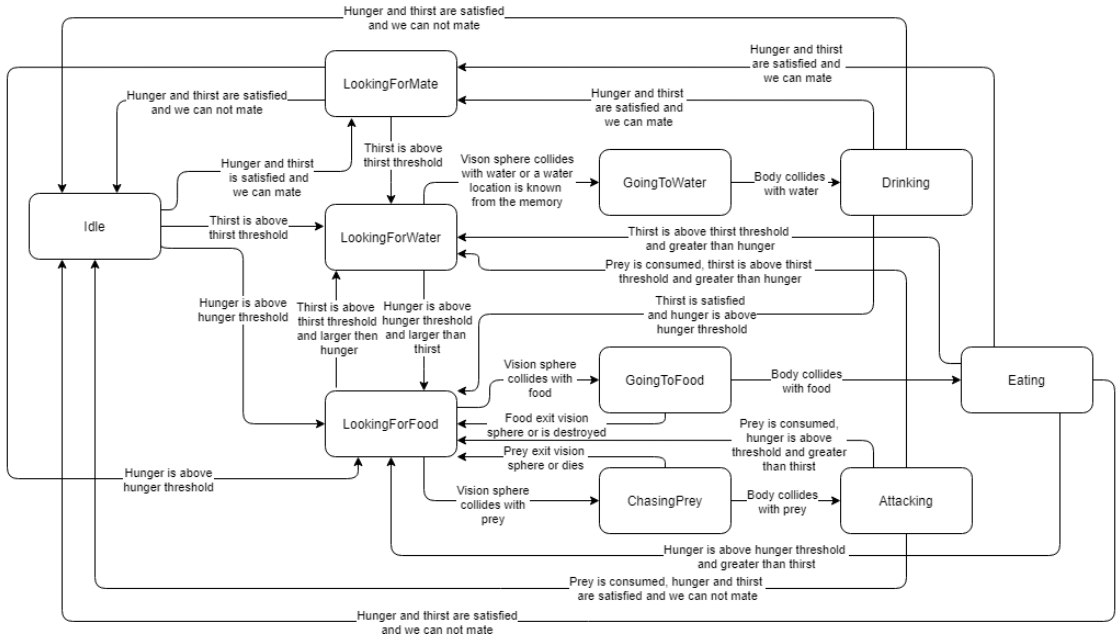


Figure A.2: A state diagram describing the state machine used by bears. The diagram features the states of the state machine and the state transitions with associated transition conditions.

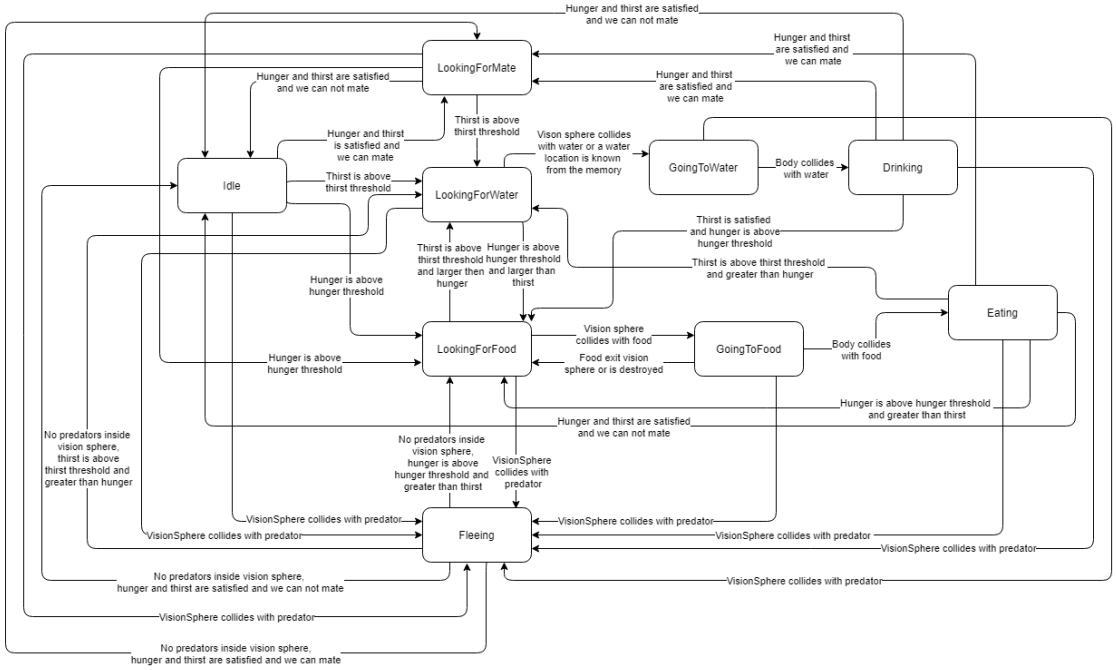


Figure A.3: A state diagram describing the state machine used by rabbits. The diagram features the states of the state machine and the state transitions with associated transition conditions.

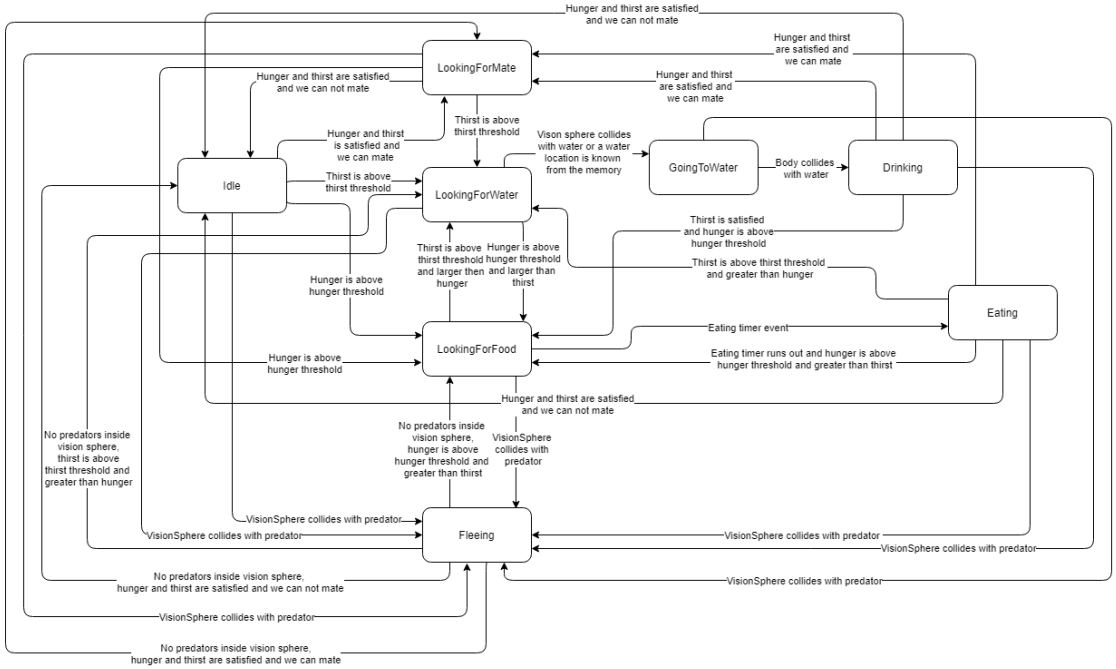


Figure A.4: A state diagram describing the state machine used by deer. The diagram features the states of the state machine and the state transitions with associated transition conditions.

B

Performance Benchmarks

Amount of game objects	Minimum FPS	Maximum FPS	Average FPS
500	28.4	42.1	39.1
600	25.2	34.6	32.2
700	25.0	29.3	27.7
800	19.0	26.4	24.2
900	18.4	23.4	21.2
1,000	16.6	20.6	18.8
1,200	11.2	17.0	15.6
1,400	10.0	14.8	13.0
1,600	8.9	13.0	11.2
1,800	7.9	10.9	10.0
2,000	7.3	10.4	8.9

Table B.1: Performance degradation in the dynamic scene as a result of increasing the amount of game objects. Configuration A was used to obtain these results.

Amount of entities	Minimum FPS	Maximum FPS	Average FPS
500	150.1	240.0	237.4
1,000	102.7	240.0	221.3
2,000	54.1	240.0	121.7
3,000	30.6	127.4	68.3
4,000	20.2	102.1	45.0
5,000	16.8	66.5	32.7
6,000	8.9	33.8	23.9
7,000	5.5	26.3	17.6

Table B.2: Performance degradation in the ECS scene as a result of increasing the amount of entities. Configuration A was used to obtain these results.