Navigation in Three-Dimensional Ecosystem Models

Master's thesis in Computer Science and Engineering

Rasmus Lindgren

Simon Olsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Navigation in Three-Dimensional Ecosystem Models

RASMUS LINDGREN
SIMON OLSSON

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2022

Navigation in three-dimensional ecosystem models
RASMUS LINDGREN
SIMON OLSSON

Supervisor: Claes Strannegård, Department of Computer Science and Engineering
Advisors: Niklas Engsner, Ecotwin, Simon Ulfsbäcker, Ecotwin
Examiner: Ulf Assarsson, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A snapshot of one of the resulting Unity environments of the project

Typeset in LaTeX
Gothenburg, Sweden 2022

Navigation in three-dimensional ecosystem models
RASMUS LINDGREN
SIMON OLSSON
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

As the area of deep learning advances, it has become possible to predict and simulate areas of the real world in a digital setting. One emerging field is the research regarding virtual twins. Virtual twins aim to replicate aspects of the real world so that actions can be analysed digitally without needing physical access to the location. This thesis aims to research if it is possible to model reinforcement learning agents to traverse such three dimensional environments that are based on real world data.

By using a combination of land-cover data and heightmaps, the ability to generate three-dimensional environments in the game engine Unity was explored. The agents were modelled as three-dimensional animals which had the possibility to consume resources, run and rotate. These agents were trained in one of the aforementioned generated three-dimensional Unity environments, using a simple image containing multiple channels to represent the state around it. It was also these channels that the agents got as their input, essentially acting as their perceptions. Results show that the agents were able to learn and survive in the environment. Simple population tests also showed that agents were able to move to new areas to find food if there were too many agents in close proximity to them, which would lead to depleted resources. Furthermore, other tests related to the environment were done, such as how an increase in sea level could impact the environment and the population of the agents.

Keywords: Navigation, Digital ecosystems, Reinforcement learning, Unity, Satellite data, Land-cover data.

Acknowledgements

We want to express our thanks to our supervisor Claes Strannegård for all your support and guidance throughout the project, thanks to Niklas Engsner and Simon Ulfsbäcker for your expertise and experience regarding reinforcement learning and implementation in Unity, respectively, as well as your feedback regarding the project as a whole. We also want to thank Heather Reese for your insight in the satellite data sources, mainly the land-cover data that we ended up using.

Rasmus Lindgren, Gothenburg, June 2022
Simon Olsson, Gothenburg, June 2022

Contents

List of Figures

Glossary

- **Environment** - An environment in the reinforcement learning context is the world in which an agent exists. It usually contains a set of rules which an agent must follow, and usually gives negative and positive rewards depending on what actions the agent takes.
- **Agent** - The machine learning model that is being trained to succeed in the environment. It has a set of actions it can do which can change the state of the environment. The agent's goal is to maximise its reward.
- **Episode** - A set of states which start from the initialisation of the agent until the agent reaches its last state. Currently, the episodes are 3000 steps, during which the agent has to maximise its reward. However, it doesn't have to be a set number of steps, in chess for example the last state would be when there is a winner or a tie.
- **Step** - A change of state in the environment, that happens after an agent has taken an action.
- **View frustum** - An area that can be seen by an entity.

1
Introduction

An ecosystem is generally described as an enclosed geographical area, where living organisms such as animals and plants work in tandem with the climate and other aspects of nature to form a self-sustaining circle of life[1]. In the present day, with climate change and global warming posing major challenges, the ability to study ecosystems, what affects them, and how they are affected, is more important than ever before. The ability to simulate such ecosystems digitally has gained popularity, mostly thanks to the rapid improvement of real-time computer graphics. These improvements should hopefully enable the possibility of generating digital models of real-world locations, to be able to study the real ecosystems that are present in this location.

In the past, ecosystems have been simulated using mathematical models [2]. This has proven to be successful, however, with recent breakthroughs in machine learning and more specifically reinforcement learning, new opportunities have emerged. With the help of reinforcement learning, it could become possible to have the ecosystem, and all its members, learn to exist according to the rules given to them, rather than by predefined equations.

1.1 Aim

This thesis aims to investigate if it is possible to represent *animal navigation* in a simulated ecosystem, while also exploring the possibility of using satellite data from a real-world location to create the three-dimensional environment of the ecosystem.

The meaning of animal navigation is to have animals that have the desire to move towards energy sources, while simultaneously avoiding obstacles such as trees and bodies of water, just like real animals do. The animals should also be able to move toward energy sources that might not be in the animals' immediate field of view, meaning that they should be smart enough to navigate to areas that have high concentrations of food. In other words, the animals in the ecosystem should desirably have some kind of brain that assists them in deciding where to move next.

The thesis will answer the following questions:

- If it is possible to programmatically generate three-dimensional environments in Unity (see Section 2.1 for further explanation about Unity), based on satellite imagery. Since there is no available metric to decide how realistic an

environment might be compared to the real-world location, the criteria for success will rather be if it can be done or not.
- If it is possible to simulate animal behaviour using reinforcement learning in such three-dimensional environments.

1.2 Delimitations

Due to the extensive possibilities of this project, and the risk of adding more and more features, several delimitations were made.

- There will not be any modelling of buildings or other man-made constructions.
- There will be no day and night cycles or other environmental changes, this could be a possible addition in future versions.
- There will be a geographical restriction to the country of Sweden, regarding which locations can be modelled, the main reason for this is covered in Chapter 3.
- The input and action space is limited to what can be found in Unity.
- The success of the reinforcement learning model will largely control how realistic the animals will be in the three-dimensional environment.

1.3 Ecotwin

The Ecotwin project [3] is an open-source ecosystem simulator in the early stages of development. Currently, it can run simple simulations in small and predetermined three-dimensional environments. This thesis will build upon the Ecotwin project by using part of its codebase in the development and hopefully integrate the results of this thesis into this project. The codebase is kept and maintained through GitLab [4].

2
Theory

A simulated ecosystem can in theory be used to learn about animal behaviour and to see how changes in the environment impact the native wildlife. For example, a simulated ecosystem can be used to model a population of fish in an area where fishing is being conducted. Another example would be an ecosystem being used to monitor populations of species where urbanisation is taking place, e.g. what are the consequences to the wildlife if a bridge or a building is built in a specific location.

Simulated ecosystems could also make a great addition to software, such as video games[5], where they can act as environments for players to observe and interact with. If implemented successfully, this could potentially make certain games more enjoyable, since one can imagine that this type of environment can be richer in features and details than most typical video game environments. Similarly, this kind of game could also provide a new feature to the gameplay; the actions that players perform have consequences for the environment, for instance killing too many animals of a certain species might lead to the extinction of this species.

2.1 Unity and Three-Dimensional Modelling

A popular tool to model three-dimensional environments is the game engine Unity[6], which is a real-time rendering application. Unity is widely used to render graphics, and create animations and other visualisations, most notably in the video game industry. However, it has also started to gain traction within the field of three-dimensional simulation, for example simulating the process of vehicle testing, to reduce the amount of physical testing needed[7].

To achieve an accurate representation of real environments, different methods such as manual modelling and procedural generation are frequently used[8]. However, manual modelling is too time-consuming while procedural generation is mostly used to model fictional environments, and is therefore not useful for environments that are intended to mimic real-world locations.

For a three-dimensional environment, adding trees, water, mud, and landscapes with heights such as mountains and valleys all add another aspect to consider; a height value. These objects and landscapes will not only act as obstacles to the animals; they will also reduce their view frustums. In other words, these obstacles should influence what the animals see, and also what decisions they make for their next

movements.

2.2 Reinforcement Learning

Reinforcement learning is a subsection of machine learning in which the model, usually called an agent, has to learn to maximise its cumulative reward in a certain environment. What differentiates it from other types of machine learning is that the data is collected through observations taken from the environment, by the agent, rather than receiving it from the user. This data is then subsequently used by the agent to learn. This is convenient since there is no need for large datasets but it also means that the agent can be very biased towards the data it collects.

A reinforcement learning algorithm is comprised of a set of states, a set of actions, and a finite reward. The state space and action space is most often discrete, and thus also finite in size which is the case for this thesis, but can also be continuous, and thus infinite. For each step in this algorithm, a model takes an action and observes a reward and a next state. This reward is then used cumulatively over many steps to teach the model what behaviour is considered good or bad. This behaviour that the model learns is called the model's policy. An example of a reinforcement learning loop can be found in Figure 2.1, where an agent performs an action which impacts the environment, collects observations about the new environment state, and finally gets its reward.

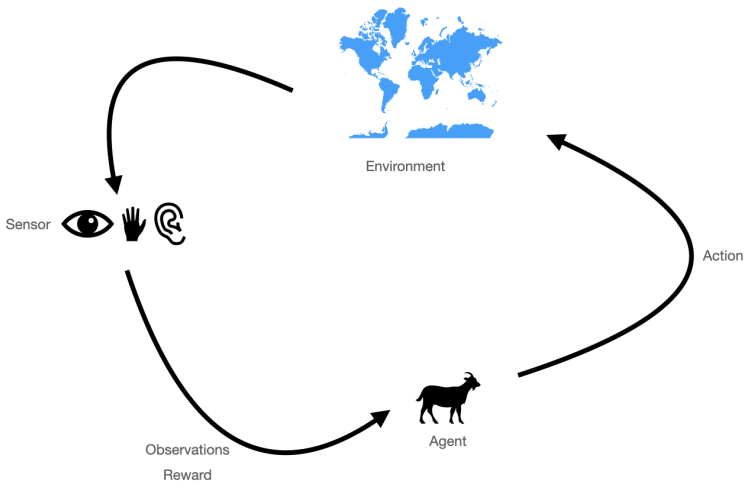

Figure 2.1: Simple visualisation of the reinforcement learning loop

Earlier work has shown that reinforcement learning can be used to make agents learn complex behaviour, one example being OpenAI's *OpenAI Five*, which was the

first AI team to defeat the world champions in the video game Dota 2[9]. Reinforcement learning has also been successfully used to model animal behaviour, one example being DeepMind showing a flocking behaviour that emerges with animals in a simulated ecosystem[10]. What was worth noting was that the flocking behaviour emerged without any input on how the agents should behave, which was different from previous research, which required explicit flocking behaviour to be successful.

Another example of reinforcement learning successfully being implemented for complex behaviour, in Unity, is a publication by Ubisoft[11]. In this study, deep reinforcement learning is successfully used to implement NPC[1] navigation in several complex environments.

2.2.1 Proximal Policy Optimization

Reinforcement learning algorithms have proven to be successful in interacting and learning, in an environment. By adding a neural network to traditional reinforcement learning algorithms, such as Q-learning, further advanced behaviour can be learned. However, there is a risk with these deeper reinforcement learning algorithms in that they may find a local minimum and get stuck there. This can make an agent drastically change its policy for the worse in the long run.

The *proximal policy optimization* (PPO)[12, 13] algorithm aims to solve this issue by "clipping" the changes between policy iterations, using the clipped surrogate objective, which can be seen as the loss function. The policy update function[14] for PPO is defined as:

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{E} \left[L(s, a, \theta_k, \theta) \right], \tag{2.1}$$

L is the objective function:

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right) \tag{2.2}$$

s is the state, where a is the action, $\pi_{\theta_k}(a|s)$ is the policy for the parameters Θ after k steps and $\pi_\theta(a|s)$ is the new policy. $A^{\pi_{\theta_k}}(s, a)$ is the advantage for the policy π after k steps, which can also be expressed as $Q(s, a) - V(s)$, the difference between the Q-value for the state and action, and the value function of the state. E is the expected value, and g is defined as:

$$g(\epsilon, A) = \begin{cases} (1 + \epsilon)A & A \geq 0 \\ (1 - \epsilon)A & A < 0. \end{cases}$$

The objective function L (equation 2.2) sets a limit to how much the policy can change between steps since the "min" term chooses the minimum of the reciprocal of the new and old policies, and the g function, which depends on ϵ (both multiplied with the advantage function). Therefore, if the old and new policies differ too much,

[1] NPC is an abbreviation for non-player character

the term that depends on ϵ will be selected, but if the policy change is within the chosen limits of the algorithm, the first term is selected instead.

In practice, this means that if an action should be a lot more probable than previously, the algorithm will limit this increase in probability. This ensures that the algorithm is more stable and does not suddenly make big policy changes.

The algorithm has proven to be successful at training advanced reinforcement learning agents, one example being the previously mentioned OpenAI[9], which used it as their default reinforcement learning algorithm.

2.2.2 Unity ML-agents Toolkit

A convenient way of training reinforcement learning agents in Unity is to use the Unity ML-agents toolkit[15]. This toolkit provides an accessible way to gather observations from the environment, for example in the form of float observations, ray cast observations or camera observations, which allow the agents to get images from the environment as their inputs. Furthermore, it allows for easy use of multiple reinforcement learning algorithms, one of them being the aforementioned PPO.

The Unity ML-agents toolkit has been used to model different complexities of environments such as simple discrete tasks where the goal is to move towards an object, as well as more advanced examples where the agents have to learn to play soccer against each other[16]. Even more complex environments have been tested where an agent had to navigate a complex 3D-environment[17]. Although the agent proved to be worse than human players in this example, it still shows the possibilities of what could be modelled with the toolkit.

2.2.3 Multi Agent Reinforcement Learning

Multi-agent reinforcement learning is the idea of having multiple agents learn in the same environment. By introducing multiple agents, the complexity of the learning can increase significantly. An example of this is that the transition between states is no longer limited to the actions of one agent, but rather a combination of every agent's actions. In an ecosystem, one example of this would be that a resource was consumed by another agent, which changed the state without any input from that specific agent. Another example would be predators having the prey move around independently, which means that the state for predators is dynamic even if it stands still.

The advantage of this idea is as one could imagine; being able to have multiple agents choosing actions simultaneously. This would help improve the realism of a digital ecosystem, especially since most ecosystems have many individuals of the same species present, and many of these species also move in herds, flocks or other forms of groups.

2.2.4 Curriculum Learning

While research shows that reinforcement learning agents have been able to successfully learn and excel in different environments, they often take a lot of time to train and it can be unclear how well they generalise to other environments. An example would be an agent having to die by falling off a cliff several times before it realises that this is a bad action. However, real animals know that falling from heights is dangerous and will therefore avoid such scenarios, while an agent does not have the same concept of the world initially, since it only has experiences it has previously taken. To mitigate this unnecessary behaviour, some ideas have been proposed, one of them being *curriculum learning*.

Curriculum learning is used to introduce new levels of complexity when the agents learn how to handle certain problems[18]. This means that, in this project for example, they first learn how to consume resources, then how to avoid water, then how to navigate the terrain at different heights, transferring previous knowledge to the next level. This can be seen as a variant of transfer learning, which is a common technique in areas such as image recognition and natural language processing. In these areas, it is common to use pre-trained models, which have been trained on large data sets or data sets with similar domain, to then fine-tune them to the specific area/data set that is being researched[19].

2.3 Reinforcement Learning in Ecosystem Modelling

Previous research involving reinforcement learning and animal behaviour has often been focused on modelling a single aspect of animal behaviour such as the predator/prey relationship[20]. While these experiments have been successful in their modelling of the agents, they have often been modelled in very simple environments. This often resulted in the agents learning exceedingly well, but it is unclear how well this would translate into an actual simulation with "real" nature. Similarly, some of these papers only consider two-dimensional environments[20, 21, 22], which further simplifies the problem, and is thus less realistic than a three-dimensional environment would be.

In a study about population dynamics using reinforcement learning, it was shown that reinforcement learning was able to simulate a working population dynamic with both predators and prey, while it also showed the possibility of a large environment and population[22]. Furthermore, research has shown that predators in a simulated ecosystem can learn to cooperate while hunting, to sustain themselves [23]. This research also tested different reinforcement learning algorithms, with PPO showing the most stable results.

Other studies have also evaluated the possibilities of using reinforcement learning to investigate environmental policy changes. One such study explores a reinforcement

learning policy that can properly learn how to optimise fertiliser dosages to minimise environmental impact[24]. Another study investigates if reinforcement learning could help in conservation decisions[25], focusing on over-fishing. Although these examples are limited in execution and very different from this project, they still show the possibilities that are available regarding reinforcement learning modelling ecological challenges.

2.4 Satellite Data

To construct accurate three-dimensional environments of real-world geographical locations, satellite images can be a useful resource, especially to see real environmental changes to an area over time. This also brings another aspect of realism, since the environment portrays a real-world location rather than a fictive one. By combining accurate environment modelling with a simulated ecosystem it would theoretically be possible to show how simulated animals would navigate this terrain.

A famous example of satellite imagery being used to model three-dimensional environments is the software Google Earth[26]. More recently, satellite imagery has also been used to model environments in the previously mentioned game engine Unity[27]. This is quite a novel method and will be the method that is explored in this thesis, and to what extent it can be used for this purpose.

2.4.1 Heightmap Data

A heightmap is a type of data that usually comes in the form of a regular RGB[2] image with 8-bit precision, which means that each pixel is in the range $[0, 2^8 - 1] = [0, 255]$ [28]. Each pixel in this range is then used to form the topology of the environment, where 0 is the lowest possible height and 255 is the highest possible height in the selected area. Different bit-precisions can be used to have more accurate heightmaps, one example would be to use the three individual channels of the RGB component, which could then represent $256^3 = 16,777,216$ different heights. Nevertheless, for the scope of this thesis, 8-bit precision is expected to be sufficient.

A visual example of a heightmap is shown in Figure 2.2. In this figure, it is possible to distinguish lakes, since they make up the darkest colours of the heightmap, as well as other elevations like smaller mountains which are more lightly coloured, and valleys which usually are closer to the grey colour. Note that the grey colour does not guarantee that there is a valley in that location, as some lakes could also be grey if they are situated at a higher elevation than other areas in the heightmap.

2.4.2 Land-cover Data

While a heightmap provides the topology of the area, it doesn't have any other information regarding the area. For this reason, another data source will also be used,

[2]RGB = Red Green Blue, is a standard colour model

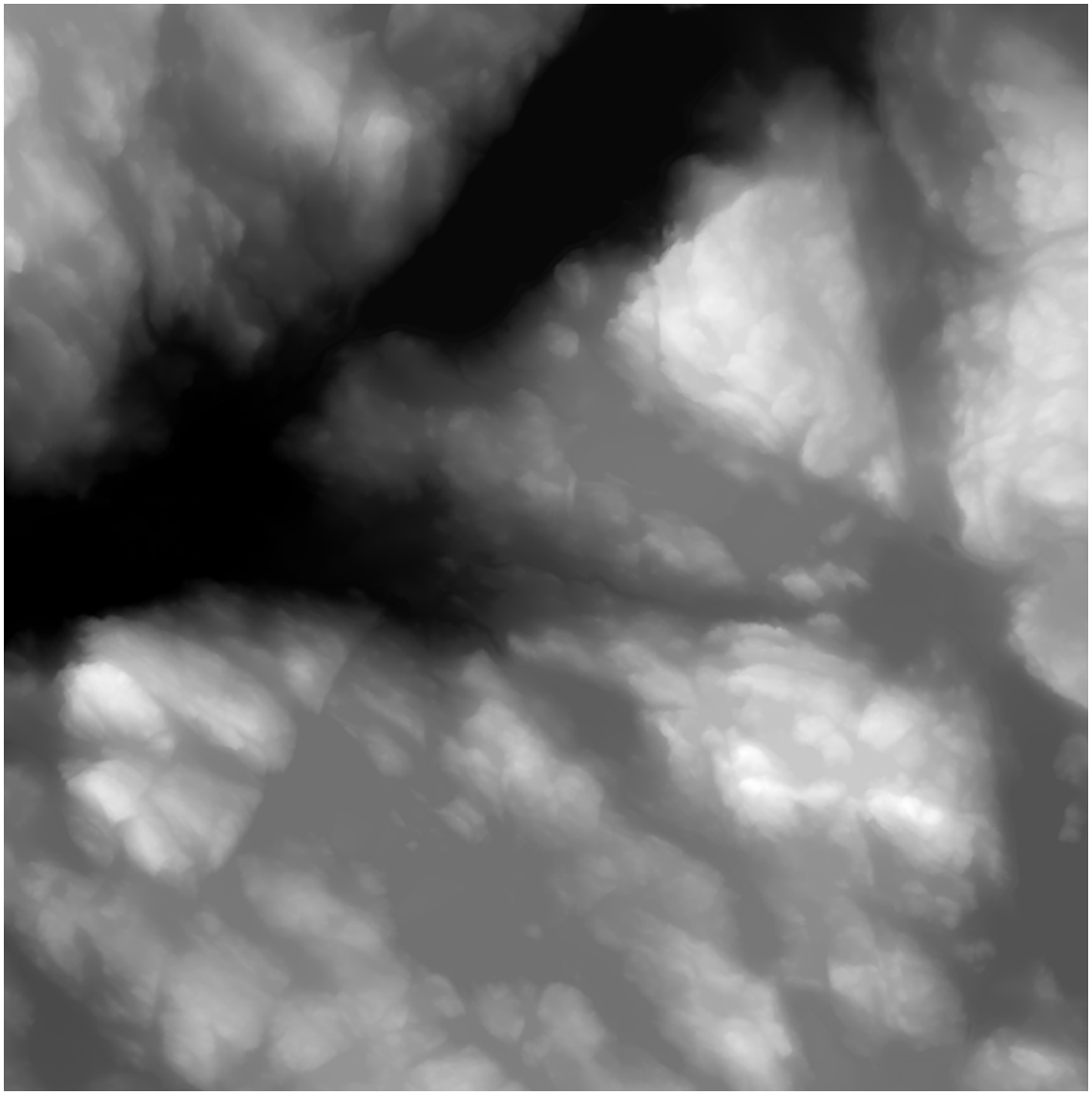

Figure 2.2: Heightmap data over an area close to Uddevalla, Sweden.

which is called land-cover data. Land-cover data is usually created from satellite data and is used to divide geographical areas into classes, examples being forests, grasslands and wetlands. See Figure 2.3 for an example of what land-cover data looks like.

In this figure, there are several of these classes present, specifically many shades of green, which indicates that the area has lots of different types of forests. It is usually the case that these classes are more specific, and divided into what one could call sub-classes. Examples of these divisions are that forests are separated into deciduous forests, pine forests, fir forests, a mix of pine/fir forests, a mix of deciduous and pine/fir forests, etc. Similarly, the class for open grounds (fields, mountains, etc) is often divided into open grounds with and without vegetation, etc.

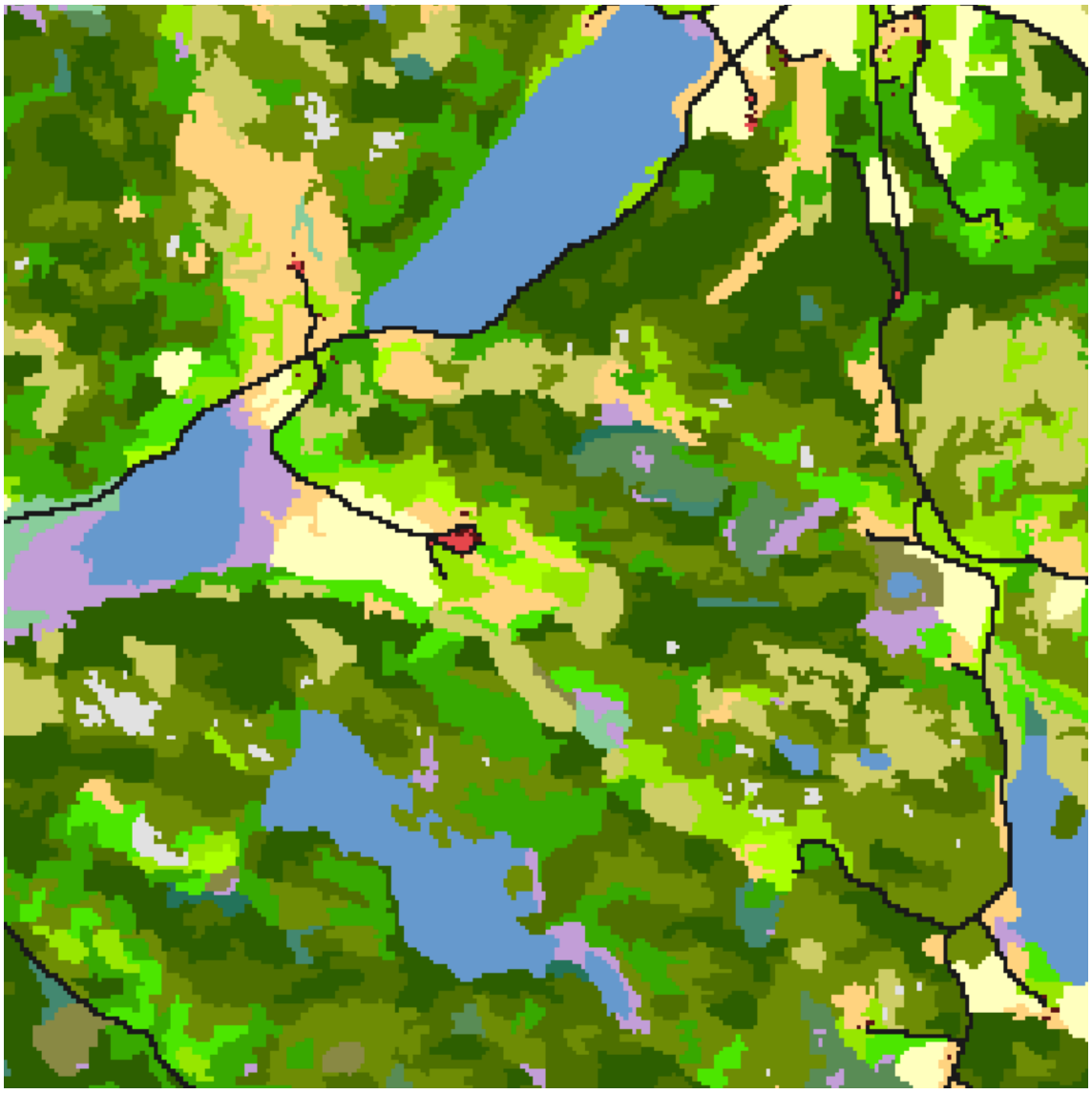

Figure 2.3: Land-cover data over the same area as the heightmap in Figure 2.2.

3
Terrain Generation

This section will cover the methods used to achieve the goal of terrain generation using satellite data.

At a very early stage in the project, an initial idea was to consider the usage of heightmaps to create environments from the real world. The heightmap is a common concept in computer graphics, where it is used to store information about elevation data. Heightmaps can thus be used to generate three-dimensional terrain, usually, in the form of displacement maps, [29] or bump mapping [30].

Initially, heightmap data was retrieved from a GitHub repository [31] where it is possible to extract heightmaps from anywhere in the world. This repository was originally used to generate heightmaps for the city-building simulator called Cities: Skylines [32], but seemed to work nicely for this project as well. This heightmap data was then applied to a terrain object in Unity, to form it accordingly.

One issue that was discovered regarding heightmaps, was that they do not capture information about the depth of lakes, oceans and other bodies of water. To remedy this, an external Python script was developed. This script would lower the height of all areas that were recognised as lakes and oceans, by comparing the heightmap to the land-cover data. The script would then lower each lake relative to its height and set the ocean height to a permanent depth. Finally, the heightmap had to be re-scaled to capture the new minimum value of the image.

For the texturing of our environments, there was an initial plan to use an orthophoto of the specific area. An orthophoto is an aerial photo of an area that has had its objects geometrically adjusted. At this point in the project, the data source from the Swedish University of Agricultural Sciences (SLU) [33] was discovered, where both heightmap data and orthophotos could be gathered. This data source, similar to the previous GitHub repository, allowed for the drawing on a map that selected a part of Sweden to download heightmaps and orthophotos from. See Figure 3.1 for a visualisation of this feature.

This orthophoto produced a reasonable looking texture on the terrain, as is more discussed in Section 5.1, but it was not considered realistic enough for the project, especially on ground level.

After some further research and discussions with our supervisor, the decision was to

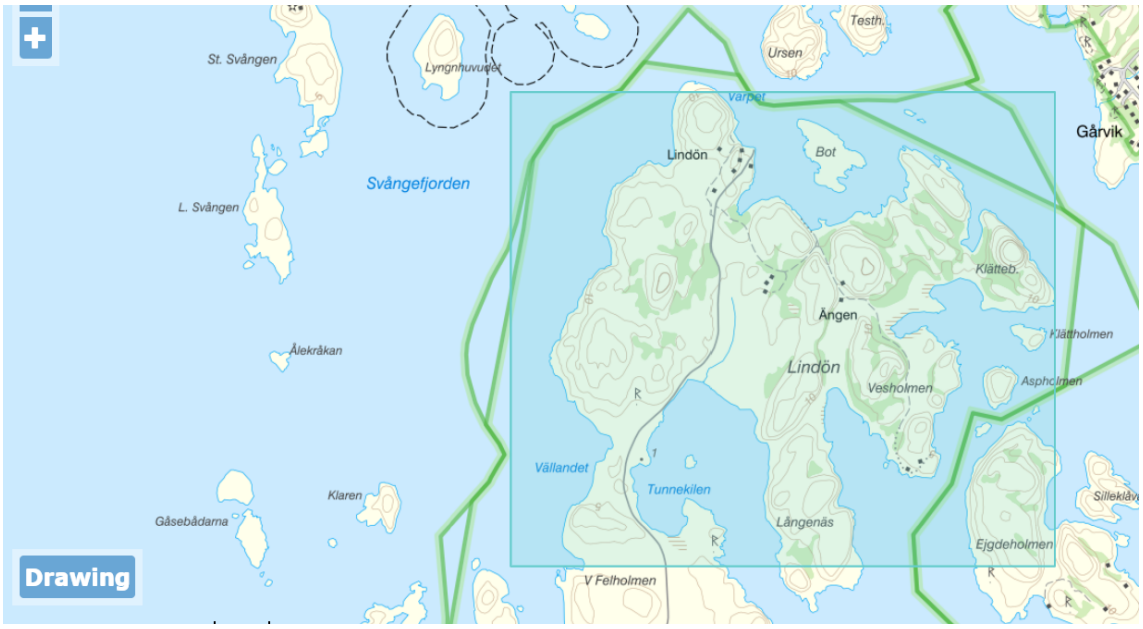

Figure 3.1: The location selection feature from the SLU data source.

use land-cover data from the Swedish environmental protection agency [34]. This was mainly done because of the easy access to the data, but also because the resolution of 10 meters that this data source has, was better than most other land-cover datasets. It was also at this point in the project that it was decided to limit the geographical scope of the satellite data to Sweden, mostly because of the decision to use this data source.

QGIS [35], a tool that handles geographic information, was used to work with the land-cover data. The aforementioned classes of land-cover data, opened with QGIS, can be seen in Figure 3.2, where the different shades of green are forest types, dark blue is a lake, light blue is an ocean, yellow is open grounds with vegetation, grey is open grounds with no vegetation, the brighter yellow is farmland, pink is wetland, etc.

From the SLU data source, the heightmaps were gathered in tiles of 2500x2500 meters, these tiles can be seen in Figure 3.3. Therefore, for desired areas smaller than this, QGIS was used to import the tiles of heightmaps and layer them over the land-cover data. This in turn made it possible to select a specific part of these two data sources by specifying the desired coordinates of two opposite corners of the desired location.

3.1 Land-cover Data Usage

The main idea with the land-cover data is to iterate over each pixel of the image. Thus, depending on the colour of the pixel, this corresponding coordinate in the

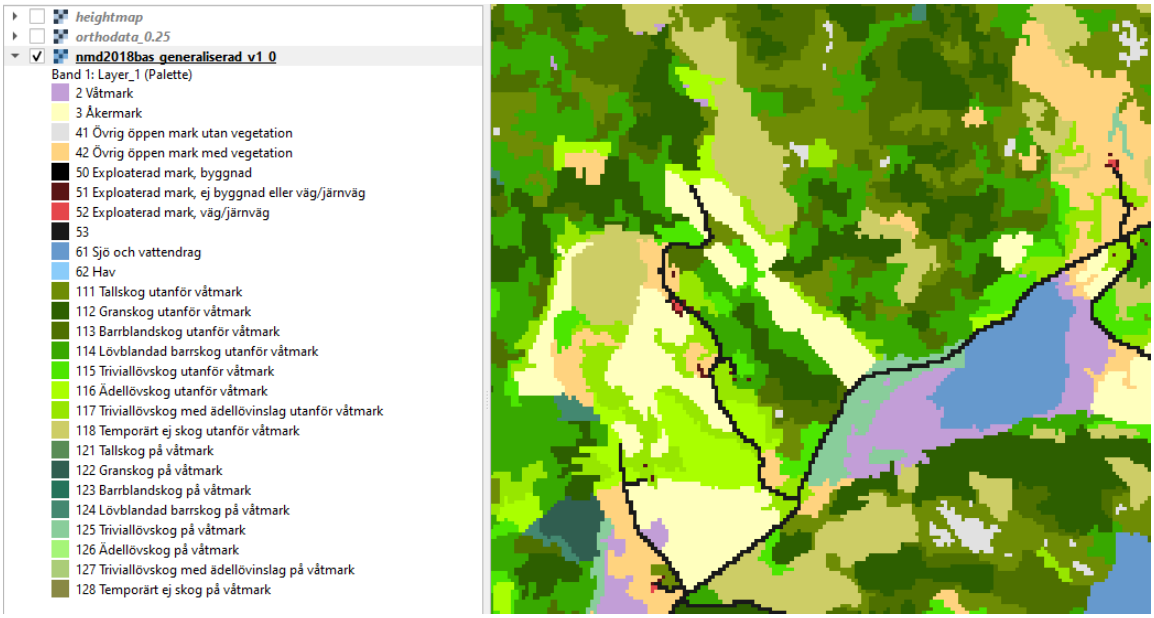

Figure 3.2: The QGIS software with land-cover data from the Swedish Environmental Protection Agency [34].

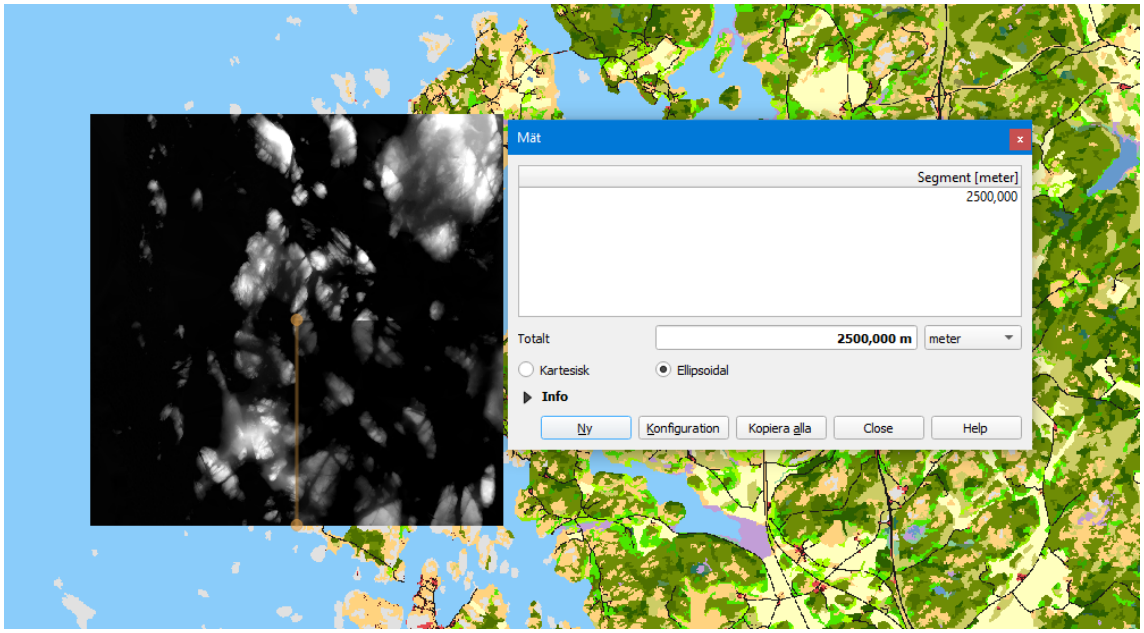

Figure 3.3: The heightmap tiles in QGIS layered on top of the land-cover data.

environment could be textured. This turned out to work almost as expected, apart from one area; edges between water and land. Because the land-cover data has a resolution of 10 meters, the edges between land and water became very sharp and unnatural. Thus, there needed to be some manual work applied, which was done through another Python script.

This Python script worked by first removing all the noisy pixels (pixels that did not belong to a land cover class colour) from the image and assigning them to a land cover class. After this was done, a filter was applied to the image to smoothen it out, by assigning each pixel to the colour that was most common among its surrounding pixels. Thus, after running the script on the image of the land-cover data, a more rounded output was achieved, which resulted in more natural land-to-water transitions in the environment.

Similar to the textures, the trees were also placed according to the land-cover data. This was achieved by checking roughly every ten pixels of the land-cover data, and if the pixel corresponds to a forest type, a tree is placed with a random offset, with the tree type depending on the type of the forest. If the specific forest was of mixed type, for example, a mixed forest of pine trees and fir trees, the spawned tree is selected randomly from an array that contained the appropriate trees for this forest type. The scale of each tree was also randomised to an extent, to add more realism to the forest. Since there was no explicit information available about the coordinates and size of each tree, these solutions seemed good enough to represent realistic forests.

3.2 Automatic Terrain

As a way of further automating the terrain generation, some work was put in to potentially allow an end-user to generate terrain from an arbitrary location in Sweden. This resulted in a script which allowed the user to click a button, which prompted the user to navigate to the path of the desired land-cover data. After this, the terrain was automatically generated, with the textures, trees, vegetation and bodies of water appropriately placed, ready to act as the training ground and ecosystem for the animals.

This was done by creating what is in Unity terms known as an editor script. This script allows automatic changes to be applied to the scene without running the simulation. This essentially lets a user build an ecosystem to completion before running any simulations involving animals.

3.3 First Person Controller

To enable the possibility of walking around in the environment in real-time, a controllable first-person camera was also implemented. This view simply represents a real human walking around in the ecosystem and made it possible to move around

in the simulated three-dimensional world space, to observe the paths that the agents are taking. The only limitation that this addition brings is that the agents do not see this individual in their perception.

To further enhance the realism of the project, several post-processing effects were added to the camera of the first person controller. These effects were implemented using Unity's post-processing package[36]. This package works by adding post-processing layers to the scene, which can then be used to add effects to what is known as *global volumes* and *local volumes*. A local volume means that the effect is only applied to a certain part of the environment while the global volume affected the entire scene. In this project, only a global volume was used. The final effects and their visual impact on the terrain are covered in Section 5.2.1.

4
Animal Navigation

This section will cover the methods used to achieve the goal of animal navigation.

For this goal, the agents needed to learn to do several things:

- To move towards resources such as food and water, to survive.
- To avoid obstacles. One option for this aspect would be to add a pain parameter, which would defer the animals from hitting these obstacles. Depending on the chosen parameters for the animal models, this pain parameter could simply be implemented as a negative reward in the reinforcement learning algorithm.
- To avoid unnatural movement. A few examples would be:
  - They should be unable to scale a steep mountainside.
  - Land animals should be unable to survive in seas/lakes.
  - They should have limited movement speed.

4.1 Unity 3D-model

To cover the visual representations of the agents in the environment, assets from the Unity asset store[37] were used. These assets are known as *prefabs* in Unity and act as blueprints for objects in the environment. These prefabs also came with built-in animations, which helped increase the general realism that was desired for the project.

The asset pack that was used in the project was called *Forest Animals*[38], which is an asset that contains many animals such as foxes, bears, hares, etc. The main focus for this project was the hares, but future work could also make use of the other animal models. The model of this hare can be seen in Figure 4.1.

Unity's physics engine was utilised to simulate the animals walking in the terrain accordingly. This engine can appropriately account for gravity, collisions between multiple prefabs, for instance, a model colliding with a tree or two models colliding with each other, as well as other forces being applied to objects[39].

To further enhance the visual realism of the navigation, another asset called *Ground Fitter* was also used in the project [40]. This asset uses one or more ray-casts from the prefab towards the ground below it, which collectively find the incidence angle of the impact. This angle is then used to properly rotate the prefab, which gives the feeling of a realistic animal walking, especially in rugged terrain that has noticeable

Figure 4.1: The model of a hare from the previously mentioned *Forest Animals* asset pack.

slopes.

4.2 Action Space

The agents started out being able to move discretely in all directions, i.e north, south, west and east. Later on, movement and rotations were introduced. However, this was deemed unnatural looking and it was decided that further realism should be sought after. The resulting idea was to make all the actions into animations, which also ultimately made the action space increase in size. This was because both the angle and movement were no longer considered to be two actions merged but rather a new action.

The animator controller with this action space can be seen in Figure 4.2, where all nodes (grey rectangles) are actions and all possible state transitions are visually represented as directed arrows. The idea behind the controller was that an agent should not be able to transition between all states. For instance, it should start walking before it could run, if it was running it should start walking before it could start eating, etc.

The training with animations as actions was initially done by having an action start an animation, rather than outputting a direction for the agent to move towards. This proved to add lots of difficulties, some of which were:

- Should one action result in one complete animation being played (for instance, one walk animation)?
- Should actions be able to cancel a current animation? For instance, if the

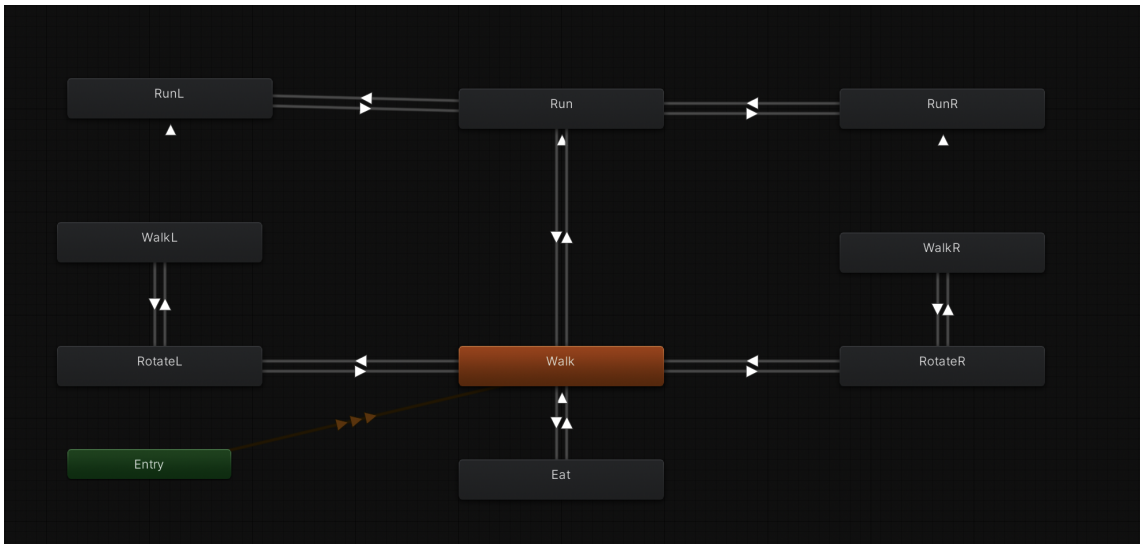

Figure 4.2: Animator controller in Unity. The orange box is the default state of the agent with transitions being possible only being possible between the agents.

agent is walking, and the next action says that the agent should start eating, should the walking animation be cancelled?
- How many animations should be used (i.e how large should the action space be)?

After many sessions of training, the setup which seemed to work the best was a setup where the animations would always play until completion, but there might be several actions taken for this animation.

Having one action per complete animation enforced, i.e. having actions manually called whenever an animation would end, seemed to work less well, the reason likely being that some animations are longer than others, which means that fewer actions are taken when these animations are chosen, ultimately slowing down training.

4.2.1 Action Masking

In every state, every action might not be available. For example, going from sleeping to running would seem like an unnatural state transition. To prevent this behaviour, one way would be to punish the agent for doing unwanted behaviour, i.e giving it a negative reward if it goes from sleeping to running. Another way would be to mask this behaviour, meaning that it is simply not possible for the agent to choose any action it wants. This could make the agent learn faster since all the possible action combinations are reduced, which in turn can make it explore faster. Action masking was used in the earlier stages of the project when it was not possible to go between all possible states, to limit the possible actions the agent could take.

However, it was decided not to implement this idea, since the states do not include transitions that can be deemed as unrealistic (there is no sleeping state, for instance,

so no possible transitions like sleeping to running, etc), and it was thus deemed not needed. Furthermore, there was also a suspicion that action masking slowed down the training since the agent had to learn which actions were masked in which states to avoid them.

Because of this decision, the controller needed to be revamped, to more accurately depict the possible state transitions. Because all possible transitions were now allowed, there needed to be arrows from each state to all other states. To solve this in Unity's animation controller, the so-called *Any State* was implemented. This state is used, as its name suggests, as a state which can be transitioned to by any other state, essentially acting as a middle ground for all state transitions. Thus, with this new controller, the agent could choose freely which state to transition to, regardless of its current state.

The second revamped animation controller can be seen in Figure 4.3. This version contained a more simplified action state, but more available total state transitions, since action masking was no longer used. Worth noticing is that there are only visible arrows from the "Any State" to each other state, but the reversed arrow is implicit, because of the special property of the "Any State".

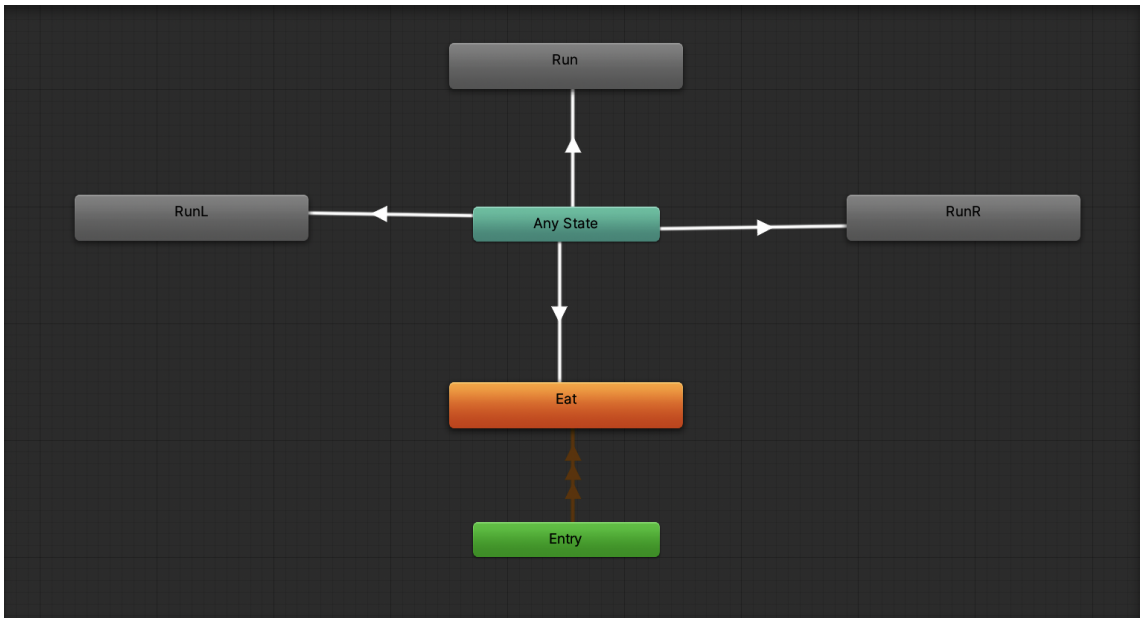

Figure 4.3: Revamped animator controller with simpler transitions.

4.3 Reward Function

The most important factor in a reinforcement learning environment is the reward function/how the agent is rewarded for its behaviour. For the agents in this project, they needed to navigate their terrain naturally, meaning that they would find food and water sources, remember their location and not wander aimlessly. Some of the most obvious rewards that were considered were:

- Positive rewards for consuming resources.
- Negative rewards for entering water bodies.
- Negative rewards for running into obstacles.
- Negative rewards for traversing heights.
- Negative reward if the agent dies before its maximum age. The maximum age was defined as 3000 state updates.

However, should the agent not consume enough resources during its exploration phase, it could lead to the agent finding a local maximum where it doesn't move. To make sure that the agent moves, it was also useful to give it negative rewards for living and moving around.

In this project, these rewards were implemented with a term called *happiness*. These happiness functions could mathematically be defined in many ways and were changed continuously throughout the project, as the agents were trained in more and more complex environments.

4.4 Agent State

Because of the complex environments in which the agents operated, great care was needed to decide what input the agent should receive. The input should ideally be as realistic as possible, but also something that the agent could relatively quickly learn to comprehend and use to maximise its reward. An initial idea was that agent would get information about its surroundings from a camera, possibly located above its current position at all times. After further deliberation, however, this seemed to be too difficult for the agent, and the decision was to go for a solution that had fewer details in the inputs.

Thus, the decision was made to send inputs to the agents through different channels instead. These channels could consist of resource locations, obstacles(such as trees and water bodies), heights, other agents, etc. This in essence meant that the agents received an aerial overview of their surroundings. Although this might not be fully accurate compared to a real animal's perception, it was motivated by the reasoning that smell, sound, and other senses could give the animal more information than what was directly in front of it.

To model this input, a special sensor called the *grid-sensor* was used[41]. The grid-sensor component made it possible to create images with an arbitrary number of channels to be sent to the agent. The agent was given 8 channels and a viewing area of 31 pixels, where one pixel represents 1x1 meters, which meant that the agent could see a radius of 15 meters around its position. The chosen channels, which can be seen in Figure 4.4 and 4.5, were:

- The agent's position and previous positions.
- Food locations.
- Water(resource) locations.
- Heights.

- Land-cover class.
- Obstacles.
- The border of the map.
- Other agents positions.

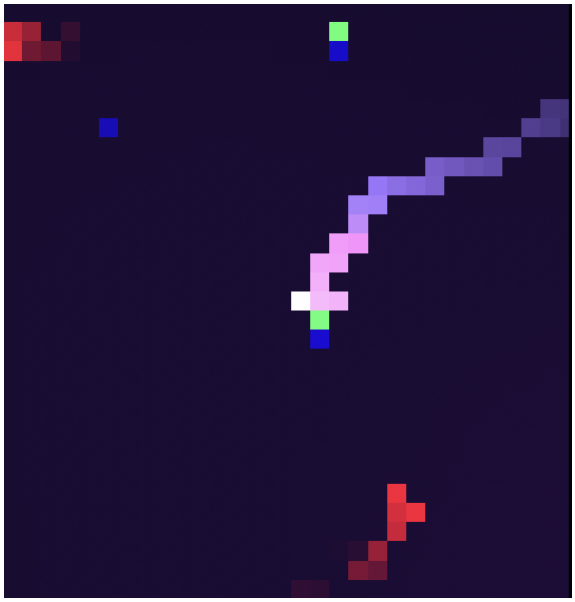

Figure 4.4: The view that the agent has. The central pixel is the agent, the green pixels represent food, blue pixels are water, dark purple/black pixels are the heights, light pink/purple pixels are the agent's previous positions and the red pixels are other agents.

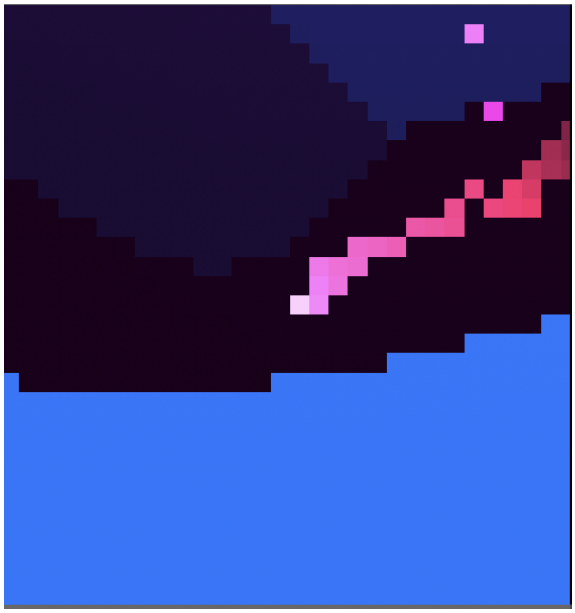

Figure 4.5: Four different land-cover classes (sea, open field, forest and other) as well as obstacles (trees) visible in the forest land-cover class.

The motivation behind encoding the agents' previous positions as a trail was to try

and give the agents a sort of memory since they only received their current state as input. This would allow the agent to potentially learn to not go back to areas where it had just been since there are likely fewer resources there.

Due to the channels only being able to encode pixels, it was not possible to encode the direction of the agent. Therefore, the agent also receives its direction in vector form (also known as the forward vector) to make sure it can correlate its rotation actions with a change in state space. It also received its current energy, hydration and happiness as inputs. The two primary reasons for this were to hopefully have the agent learn why it died when it ran out of energy or hydration, and also potentially have the agent learn to prioritise both resources.

4.5 Initial Navigation

Initially, the idea was to train the agents to move toward food sources that were placed in the environment. These food sources were initially placed at locations in the environment which had a certain class in the land-cover data, which was open fields with vegetation. The agent itself had a random spawn location, but could not be spawned in the water. The rewards for this training session were as follows:

- +1 for eating a food source
- -0.1 for collision with a tree
- -1 for walking into a body of water, which also ended the episode
- -1 for walking into a wall (border of the environment), which also ended the episode

The action space for the agent was simply four discrete directions, north, south, east and west. Initially, the training was not very successful, as the agents simply did not seem to learn anything, and instead moved around randomly.

After some debugging, the main issues seemed to be that the episode length was too long, which meant that the agent did not learn enough from sparse events of eating food, and also the fact that when the agents spawned in locations with no food, it simply did not learn anything during that episode. Simply lowering the episode length by itself had a significant impact, as further training showed improvements to agents that were spawned close to food sources.

To battle the second problem, that of agents not learning because they spawned too far away from food sources, curriculum learning was introduced in the environment. In this case, this meant that they were spawned in areas with many food sources early on in the training, to speed up the initial learning process. A simple analogy of this that one can consider is how certain animals bring prey (i.e. food) for their children to play with while they are still young, to teach them how to hunt.

4.6 Advanced Navigation

Once it was proven that the agent could learn to move towards food sources and avoid water bodies and obstacles, the agent was moved to an even more complex environment. In this new environment, the agent had to manage two resources, food and water, and the agent now died if it did not have enough of either resource.

Due to the need for the agent to balance the two resources, a happiness function that reflected this balance, and thus gave the highest reward when the agent maximised both resources, was desired. Simply giving the agent an immediate positive reward for eating or drinking could lead to an agent that only focused on one resource, which would be sub-optimal. The reward should instead ideally try to model energy consumption, meaning that the agent could get a big reward for consuming a resource and then afterwards gradually get higher and higher negative rewards if its energy deposits keep depleting. This implies that a delta value, the difference between how good the previous state was compared to how good the current state is, would be a good way to model this.

This resulted in the development of a more complex happiness function, which was defined as $\ln(1+10*\mathit{energy})+\ln(1+10*\mathit{hydration})$, where energy was based on how much food it had consumed and hydration was how much of the water resource it had consumed. The amount of these resources that the agents could have were both capped at 5, to remove the possibility of agents consuming an unlimited amount of either resource.

The reward that related to happiness was its delta happiness, which means the current happiness was subtracted by the happiness it had in its previous state. More formally, the reward for time-step t was: $\mathit{reward}_t = \mathit{happiness}_t - \mathit{happiness}_{t-1}$.

For the initial action space, which was discussed in Section 4.2 and visualised in Figure 4.2, the energy and hydration were affected (per time-step) by the following:

- -0.001 for being alive
- -0.001 for walking
- -0.002 for running
- -1 for running into an obstacle
- energy += 0.1 if on food resource
- hydration += 0.1 if on water resource

After the removal of action masking, and the reduction in action space, as discussed in Section 4.2.1 and visualised in Figure 4.3, the energy and hydration depletion per time-step were altered a bit, and now looked like this:

- -0.001 for being alive
- -0.001 for running
- -0.001 for eating if there is no food
- -1 for running into an obstacle
- energy += 0.1 if on food resource

- hydration += 0.1 if on water resource

Of note is that for every step, the agent would take the same action twice. This was done to speed up training since it would force the agent to go bigger distances at each action. However, it also meant that it would get the rewards twice. For each fully consumed resource, the agent could live at most 250 ($\frac{1}{(0.001+0.001)*2}$) steps more assuming it walked on a flat field and had enough of the other resource to survive. At the start of each episode, the agents had two of each resource.

4.7 Curriculum Learning

In total, the agent was trained in four different iterations with each iteration building on the previous one, and all of the iterations were trained on the generated environment of Lilla Amundön. For all iterations, the resources had an approximate 60%/40% split, where 60% were on open fields with vegetation and 40% were in forests and paths. The difference between the iterations was the following:

- Iteration 1: 60 agents, 4000 of each resource and all the agents being spawned on open fields.
- Iteration 2: 60 agents, 2500 of each resource and the agents are spawned all over the environment except on open fields with no vegetation.
- Iteration 3: 60 agents, 1000 of each resource, otherwise same as iteration 2.
- Iteration 4: 120 agents, 1200 of each resource and the agent can also spawn on open fields without vegetation.

The motivation for the different iterations was to introduce the learning concepts in stages, much like curriculum learning is intended to do.

The first iteration had more food and water sources to make sure the agents would more easily find them. The second iteration spawned the agents in more locations and introduced them to obstacles. The third iteration was introduced to teach the agents to really prioritise the resources, since there are now a smaller number of them.

Finally, the fourth iteration was introduced when it was observed that the agent would never cross areas with no vegetation. This meant that it would sometimes miss areas with a lot of resources due to having learnt that there was never any resource on open fields with no vegetation, so there was no reason to cross these areas. Furthermore, the number of agents was doubled to help the agents to learn to avoid other agents since more agents meant more competition for resources.

4.8 Population Dynamics

To verify that the animals behaved realistically, multiple population tests were conducted, that all aimed to demonstrate just a few of the possible applications of the

project. These population tests were also useful to verify that the agents were able to survive and seek out resources in the environment. The resources were divided the same as the training. Important to note is that for these tests, the agents were running in inference mode, rather than training.

Inference, which is a machine learning term, simply means that there is no training being performed; the agents do not gather any data, it is rather just a simulation using the trained agents. For these population tests, the agents had a percentage chance to reproduce, which was defined as

$$\frac{1 + (energy - 3.5) + (hydration - 3.5)}{1000}$$

at each step, while simultaneously requiring the agent to have above 3.5 of each resource. If this was successful, the parent would give some of its energy and hydration to the offspring and then continue as normal. The offspring was also forced to stand still during its first 200 steps to give the parent time and not make them fight over the same resources.

The first test was simply conducted to explore what the *carrying capacity* of the selected environment was. Carrying capacity is a term in environmental biology that simply means the maximum population size that can sustain itself in a certain environment [42], which in this case was the island of Lilla Amundön. The way this was done was by doing tests with two different setups:

- One setup with **60** agents initially placed randomly across the terrain.
- One setup with **five** agents initially placed in close proximity to one another, in one of the open fields with vegetation in the terrain.

These tests were done, as mentioned above, to explore what the carrying capacity was, but also to see if the same carrying capacity was reached regardless of both the agents' initial positions and the initial population size. The results of these tests are covered in Section 5.3.3.

4.8.1 Human Intervention

The second test that was performed was to simulate the possible consequences of hunting. This was done by first letting the simulation run for 7500 steps, with the same setup as the previous test that had **five** initial agents, with the same aspects of reproduction also included. After that, 5% of the agents would randomly get picked to be killed (as a consequence of hunting), during the subsequent 7500 steps. This percentage was linearly increased for each subsequent 7500 steps as well, i.e. 10% at 15000 steps, 15% at 22500 steps, etc.

The third test that was done was to simulate a possible consequence of climate change, that of the rising sea levels. To do this, the simulation first ran until it reached a carrying capacity, after 25000 steps. After that, the sea level was in-

creased by one meter, which would of course change the look of the land-cover data.

To implement this, the heightmap data was used to track which areas of the terrain would be below the new sea level, to change the look of the land-cover data to accurately depict this change. The visual effect of the first three changes in sea level can be seen in Figure 4.6.

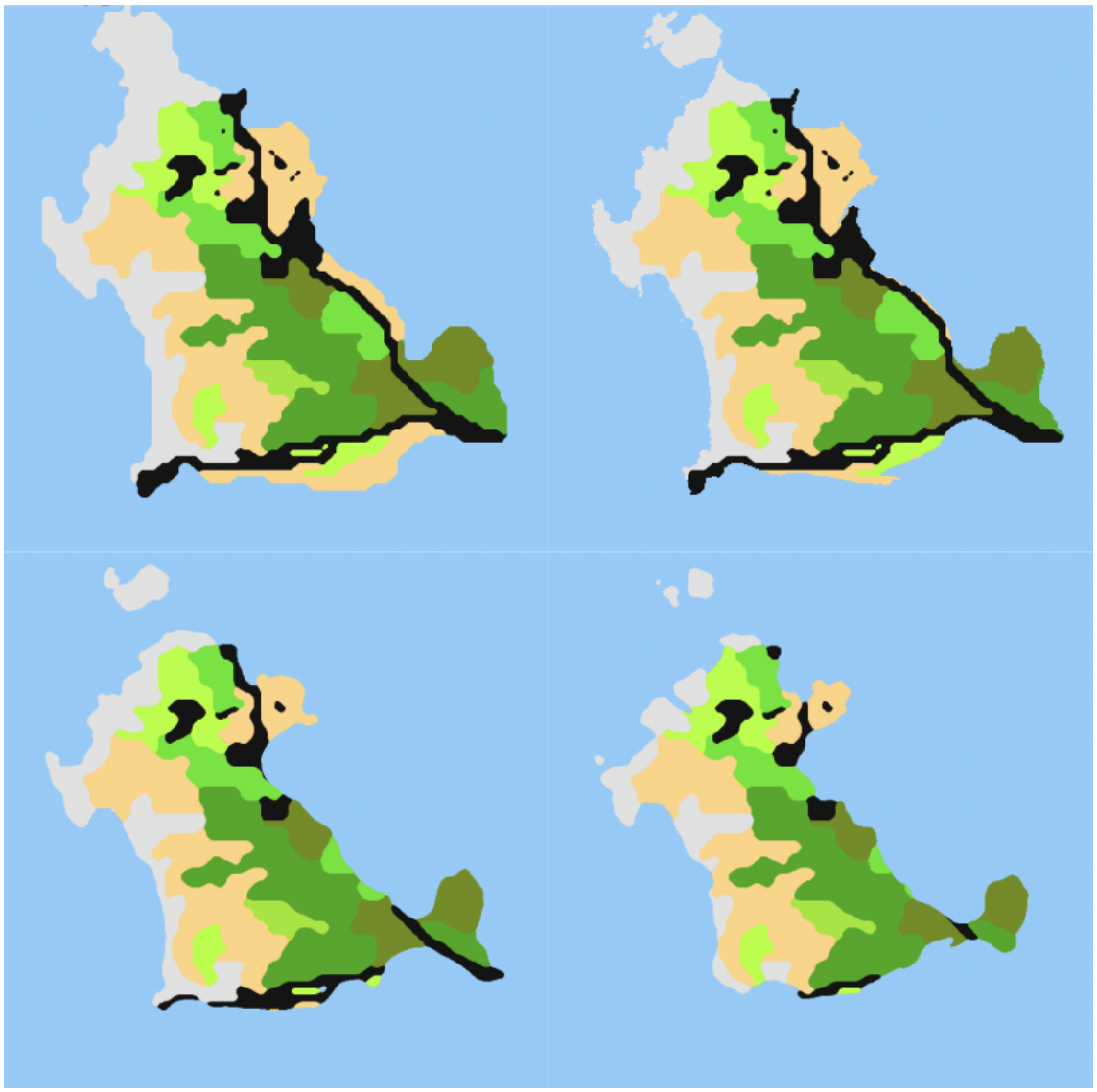

Figure 4.6: Land-cover images of Lilla Amundön. Top left has no sea level increase, top right has a one meter sea level increase, bottom left has a two meter sea level increase, bottom right has a three meter sea level increase

In this figure, one can observe that the field in the south of the island disappears underwater already after an increase of one meter, as well as the narrow field in the east and parts of the rocky area in the north. The same procedure was done for each increase in sea level, for four meters, five meters, etc, until the animals become extinct.

The fourth test was done to explore the possible addition of infrastructure to an ecosystem; in this case, a road being built across the terrain. In Figure 4.7, the land-cover data with the added road can be seen.

This test was done by first letting the agents reach the carrying capacity like in previous tests, with the percent chance of dying when walking on the road being 0%. After a certain amount of time, this percentage increased linearly, until it became 100%, i.e. all agents that touched the road died.

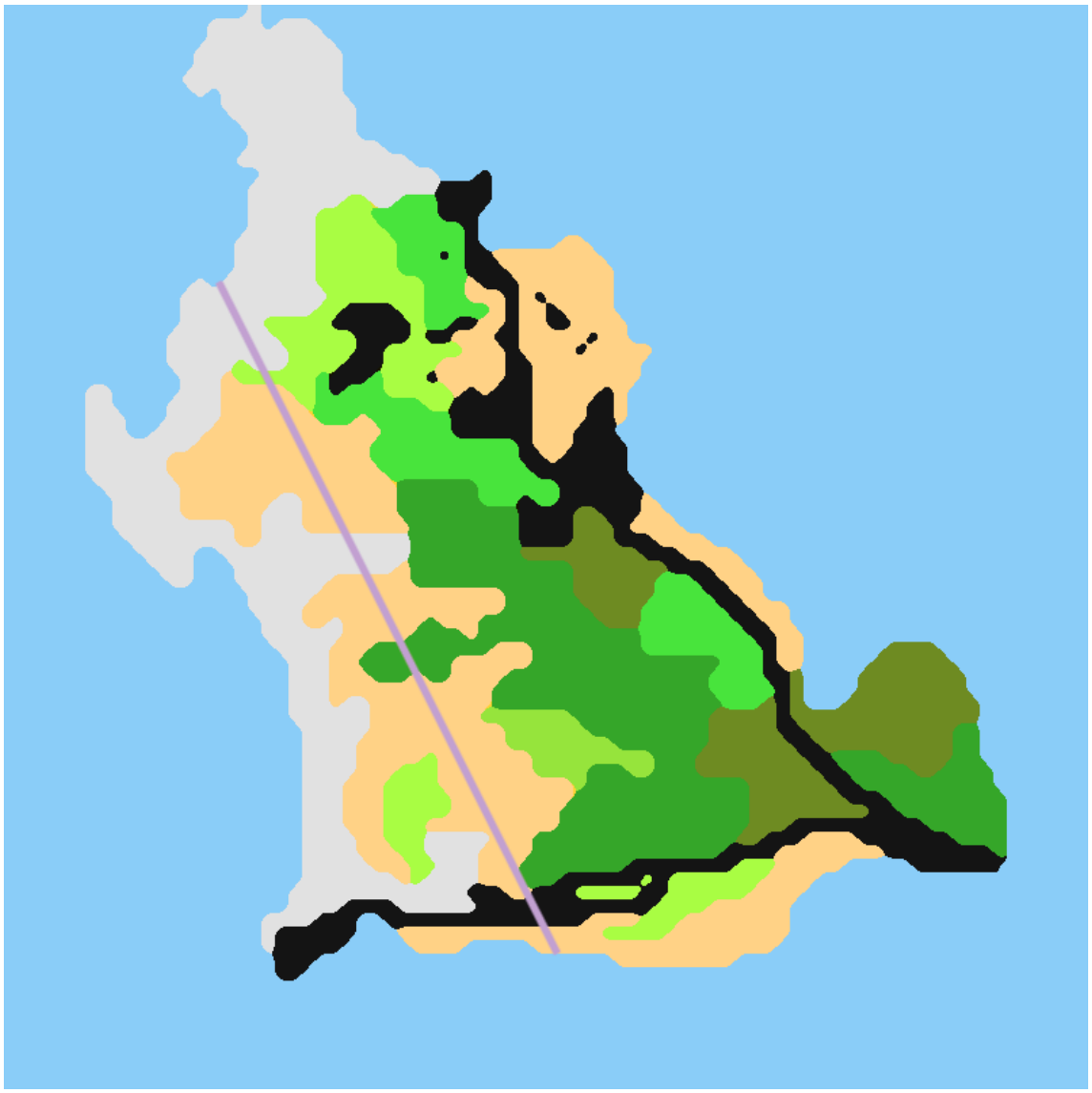

Figure 4.7: The land-cover data after a road has been added.

5
Results

In this section, the results for the conversion of satellite data to three-dimensional environments and animal navigation with reinforcement learning will be covered.

5.1 Initial Terrains

The first resulting terrain of the project can be seen in Figure 5.1, where the heightmap data was taken over an area in the mountains of Switzerland. This was done before the addition of the land-cover data, which is why the geographical restriction of Sweden was not in place yet.

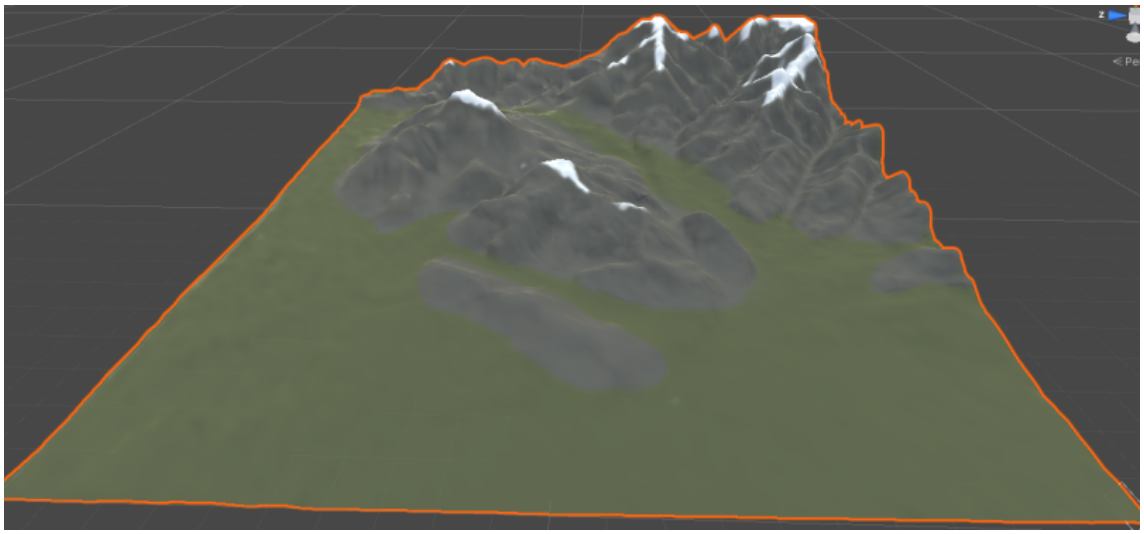

Figure 5.1: The first terrain of the project generated by heightmap data from a location in the Switzerland mountains.

This was done mainly as a backup, in case future ventures with the satellite data would not suffice, and also to get acquainted with Unity's environment and its large toolset, most notably its integration with heightmaps. It is worth noting that the only automatically generated aspect of this terrain was the topology; the textures were manually placed, as any other objects like trees and rocks also would have to be.

The second resulting terrain of the project can be seen in Figure 5.2, which is another example of an environment created from a location in the real world, this time

in Sweden. This was the first result of using the orthophoto for texturing purposes.

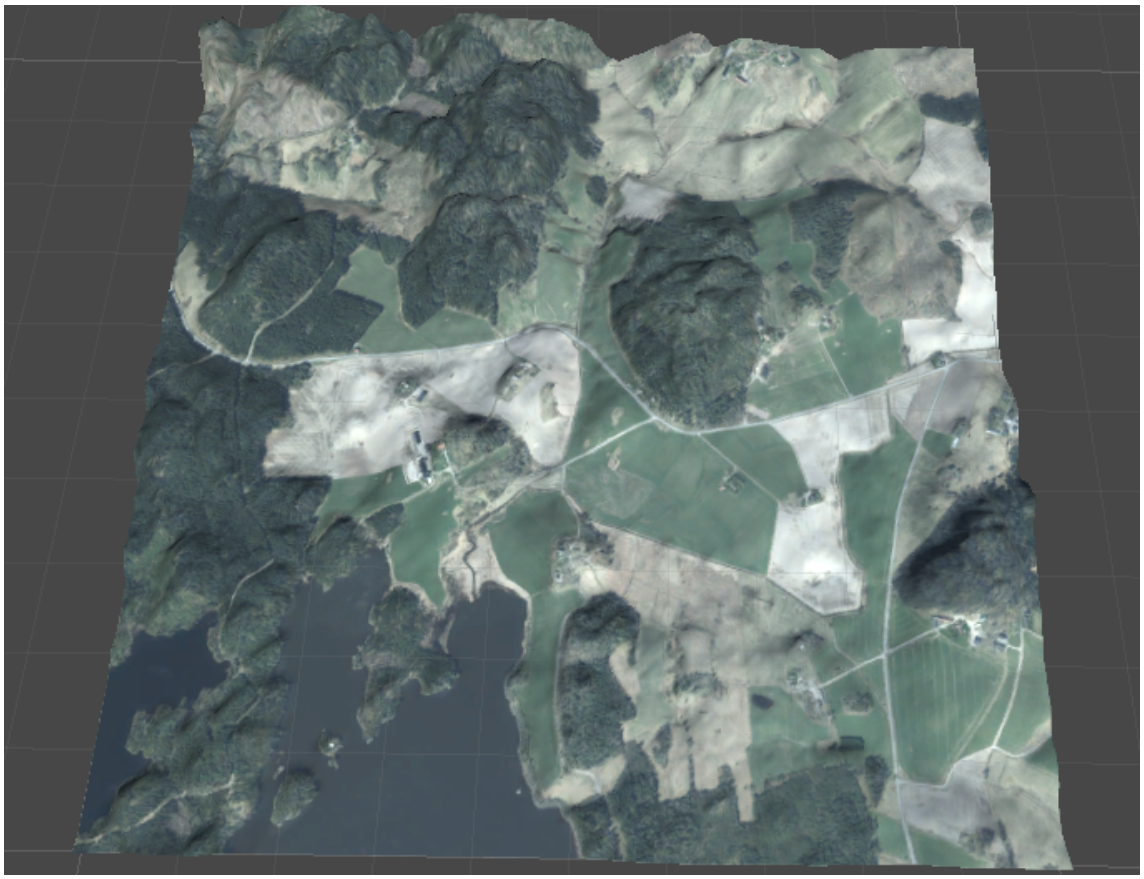

Figure 5.2: The second terrain of the project, generated by heightmap data, textured by an orthophoto, from a location in Sweden.

After experimenting with the orthophoto to be used as a texture in Unity, it was clear that the resolution of this photo was not sufficient for the project; the texture looked great from a distance away, especially from above, but when the camera was closer to the ground and at a higher angle of incidence, the textures became very blurry. Because of this blurry result, the idea of using orthophotos for texturing purposes was put aside, and focus was instead put on using land-cover data.

5.2 Resulting Environments

One of the resulting environments using land-cover data can be seen in Figure 5.3, which is a snapshot at the ground level of an island called *Lindön*, located on the west coast in the county of Bohuslän.

This location was recommended to us by our supervisor, as this area is rich in wildlife, which could then be interesting to simulate in future versions of Ecotwin.

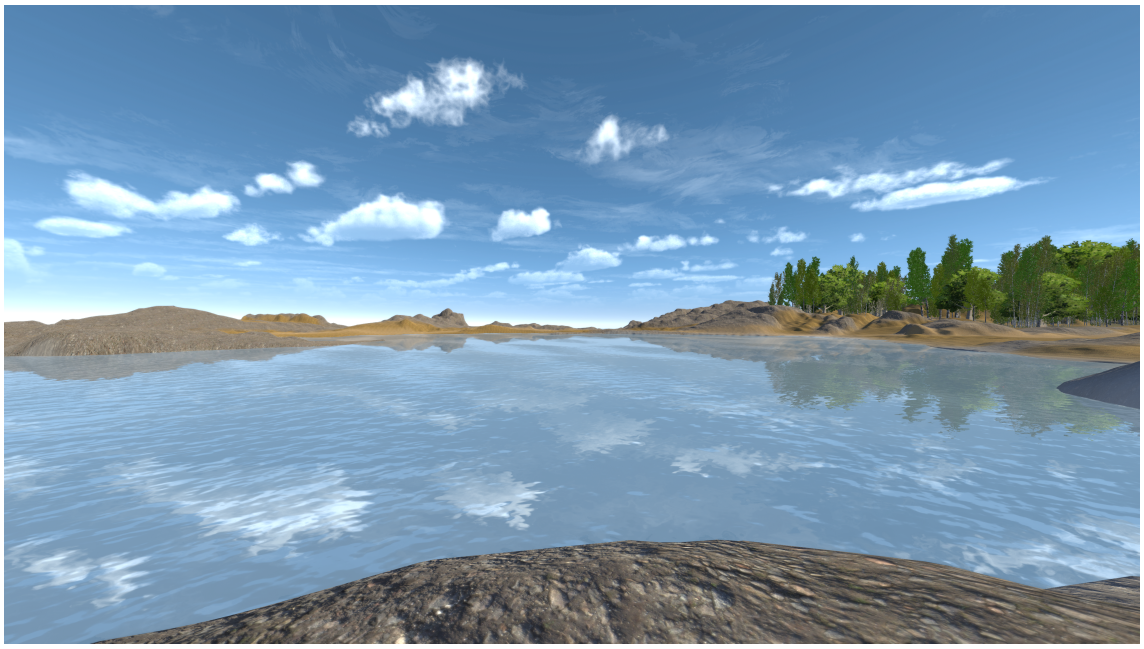

Figure 5.3: Lindön, the first example-terrain of the project that uses both land-cover data and programmatically generated textures, bodies of water, vegetation and trees.

The data used for this was as mentioned before; heightmap data to enable the generation of the topology and land-cover data to localise what texture should go where, where the forests are, where the bodies of water are, etc.

Similarly, in Figure 5.4, a bird's eye view of another modelled environment is shown, that of the previously mentioned island called Lilla Amundön.

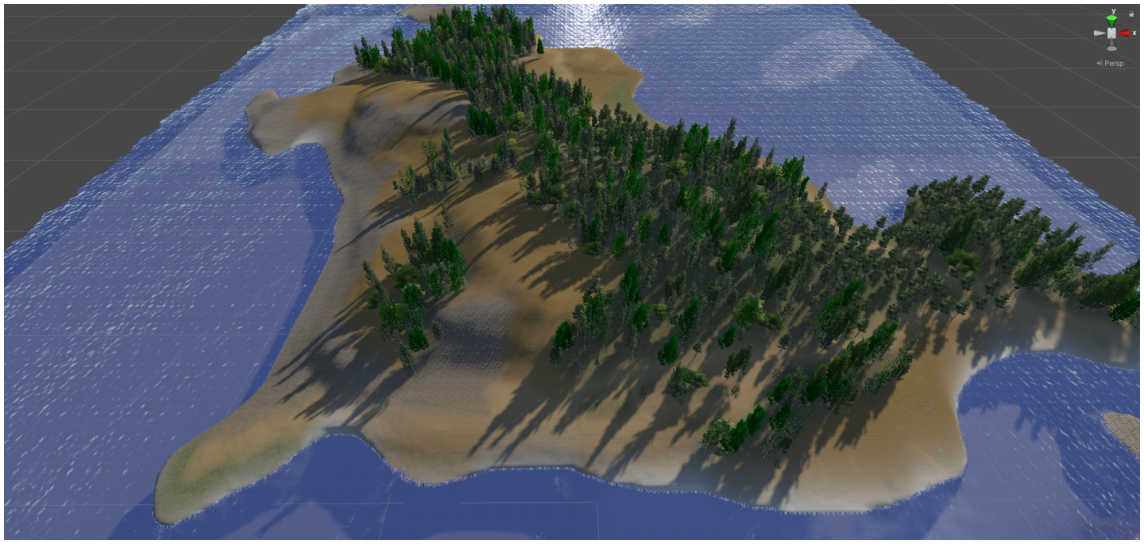

Figure 5.4: Lilla Amundön, the second example of terrain generated with heightmap and land-cover data.

The important point to make regarding these environments is that the chosen lo-

cations are arbitrary. This means that it is possible to create a digital version of any location in Sweden, as long as its corresponding land-cover data and heightmap data is available.

The land-cover data that was used had quite sharp edges, especially between land and water, as was discussed in Section 3.1. The result of the script that remedied this issue by blending the edges between land and water can be seen in Figure 5.5.

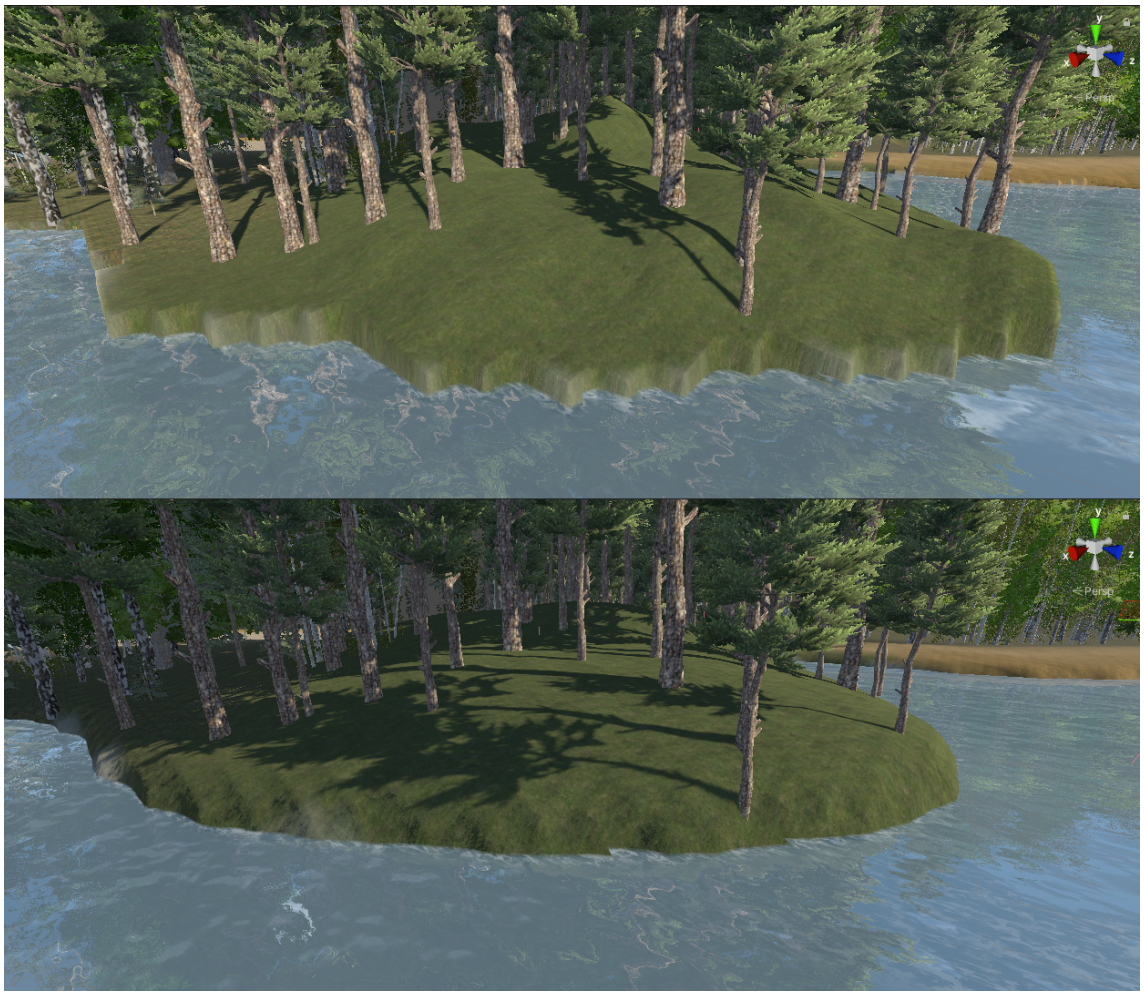

Figure 5.5: The effect of the script that creates a more smooth terrain.

Furthermore, to remove the feeling of emptiness in many of the resulting areas of the environment, vegetation was also added. This was implemented by utilising Unity's detail mesh feature [43], with the placement of specific objects depending on the land-cover data. More specifically, each type of flora was added as an individual layer in the detail mesh, and for each pixel in the land-cover data, there was a certain probability to place a flower there. The type of flower that could be placed in the specific location depended on the land-cover data, and was also randomised to an extent. Examples of these additions can be seen in Figure 5.6 and Figure 5.7.

Figure 5.6: The added flora in a forest biome.

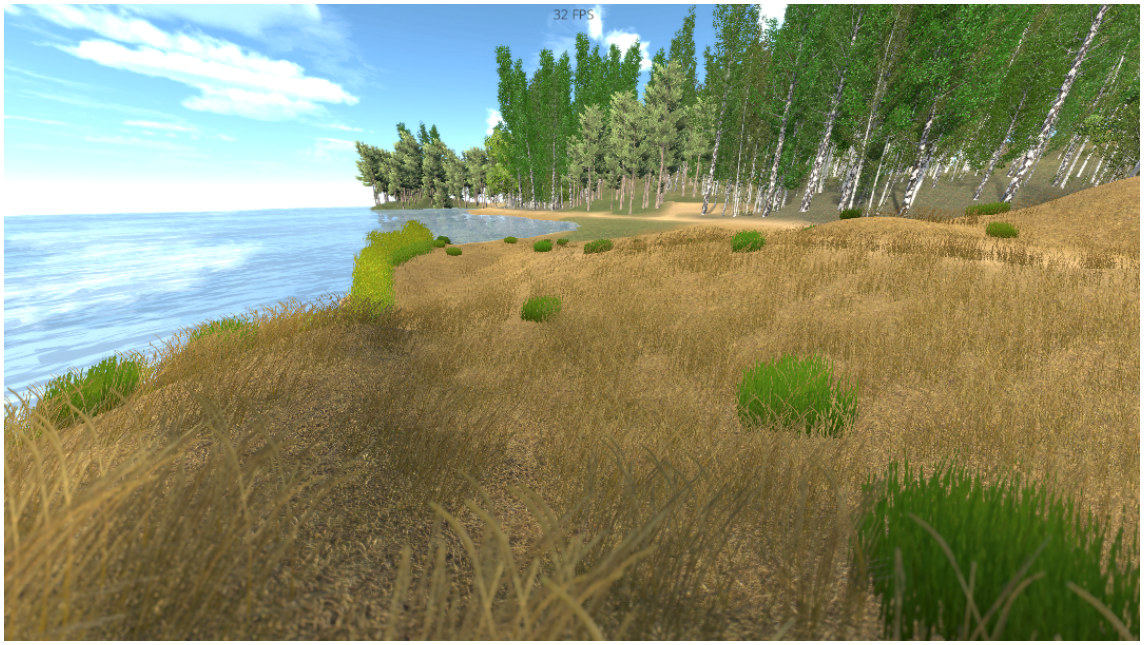

Figure 5.7: The added vegetation in an open field.

It is worth noting that these additions were done to increase the realism and visual appearance of the environments; this vegetation does not correspond to any actual data of the environments, and will thus not be accurate concerning the current vegetation of the locations in the real world.

Regarding the representation of bodies of water that were present in the different environments, an asset from Unity's standard assets was used, called Water4Advanced [44]. To automatically place this asset at the proper locations, the land-cover data was also used. In the land-cover data, there were two different classes of water bodies; oceans and lakes. For oceans, the solution was simply to lower all the ocean pixels to a set depth. For lakes, however, there could be many different shapes and depths which made it a much more complicated problem.

The current solution creates lakes by clustering nearby lake pixels together and then creating a bounding box around these pixels to generate the water object. However, due to some complicated lake shapes, this can lead to one lake being generated as two lakes or two lakes that are close to each other being generated as one lake. This can create problems when the lakes have different depths, which can ultimately make them clip through the environment. Nevertheless, this solution is currently used, future work could however improve this solution by possibly clipping the object upon collision with terrain or making each pixel a unique water object.

5.2.1 Post-processing

To add even more realism to the environment, post-processing effects were also used. Post-processing is a method used to improve the visual appearance of images or videos after the rendering process. In Figure 5.8, the differences with and without post-processing are visualised, once again in the environment of Lilla Amundön.
Note that not all effects are displayed in this figure, an example being the depth of field effect, since the entire scene is in focus. These effects could be added to any camera used in the project, whether it was a camera used for the first person controller or a camera that follows a specific agent in a simulation, etc. All the effects that were used are covered in detail below.

Ambient Occlusion is an effect which is used to determine to what extent a certain area is exposed to the ambient light, i.e the light that comes from the surroundings, as opposed to direct light sources (i.e. the sun). In this project, this effect makes the largest difference inside forests, since the rays of the sun do not reach all parts of the forest. Therefore, since a forest should ideally not be entirely dark, ambient occlusion helps this cause by taking into account the ambient light, which results in these areas being more illuminated.

Auto exposure is an effect which can be described as the phenomenon that occurs when entering a bright area before being in a dark one. This is achieved by simply altering the brightness that the camera is exposed to, when moving in and out of areas that are in shadow, most notably under trees in the different environments.

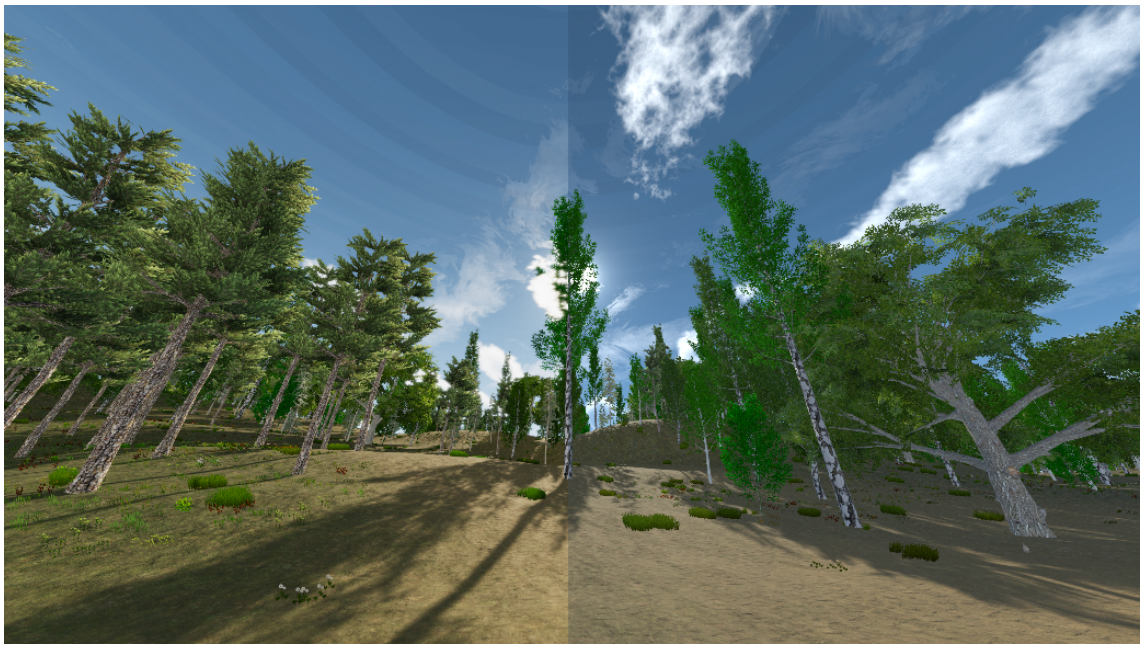

Figure 5.8: The difference between post-processing enabled (left) and disabled (right).

Bloom is the effect that represents light bleeding across objects that are exposed to it. See Figure 5.9 for an example of bloom being applied in an environment, in this case, the sun bleeding through the leaves of the trees.

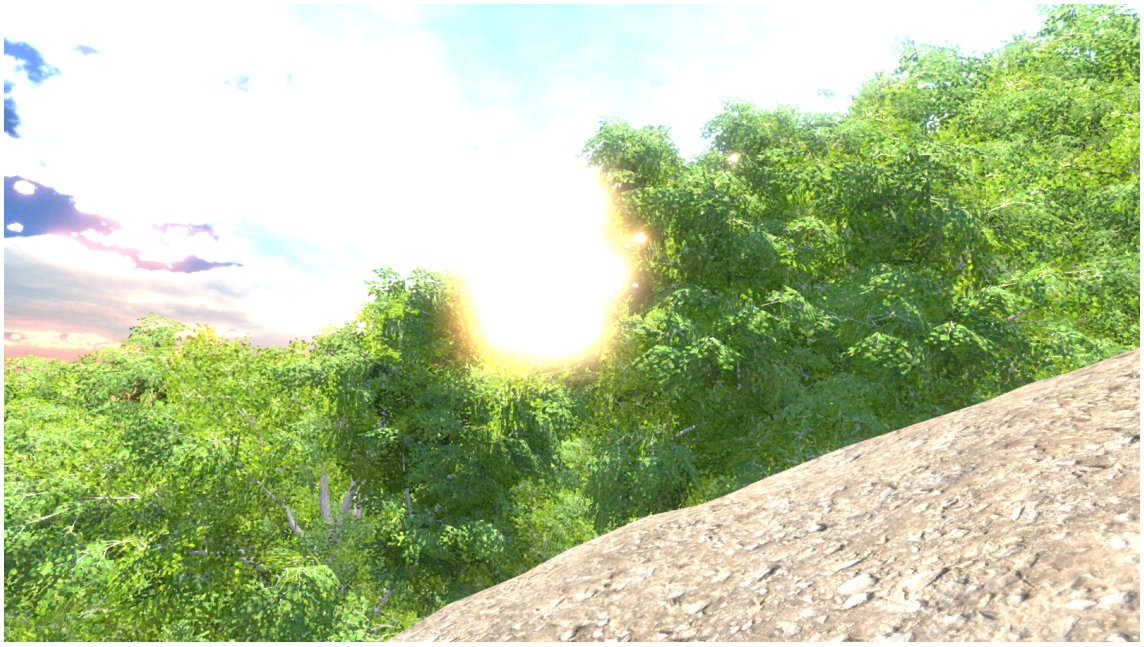

Figure 5.9: A part of Lindön with bloom enabled.

Colour Grading is an effect which is used to improve the visual quality of the

image that is displayed on the screen. Examples of the attributes of the image that can be altered are colour, contrast, gamma and saturation. Thus, it can be used to alter the pixel colours into certain colour schemes, for example shifting them towards more warm colours, which gives the feeling of summer in the environment.

It also enables the ability to use a tone mapper, which alters the pixel colours to be more suited for display on modern computer monitors.

Depth of field is an effect that is used to simulate the focus of the human eye. When this occurs, the object in focus is displayed clearly to the camera, while the surrounding objects and environments become blurry. Unity's raycast feature was used to implement this effect, by keeping track of what part of the environment should be in focus at all times. This was done by dynamically changing the focus distance of the depth of field effect, depending on the distance from the camera to the point where the raycast hit. In Figure 5.10, the visual blurring of the depth of field component in post-processing can be seen.

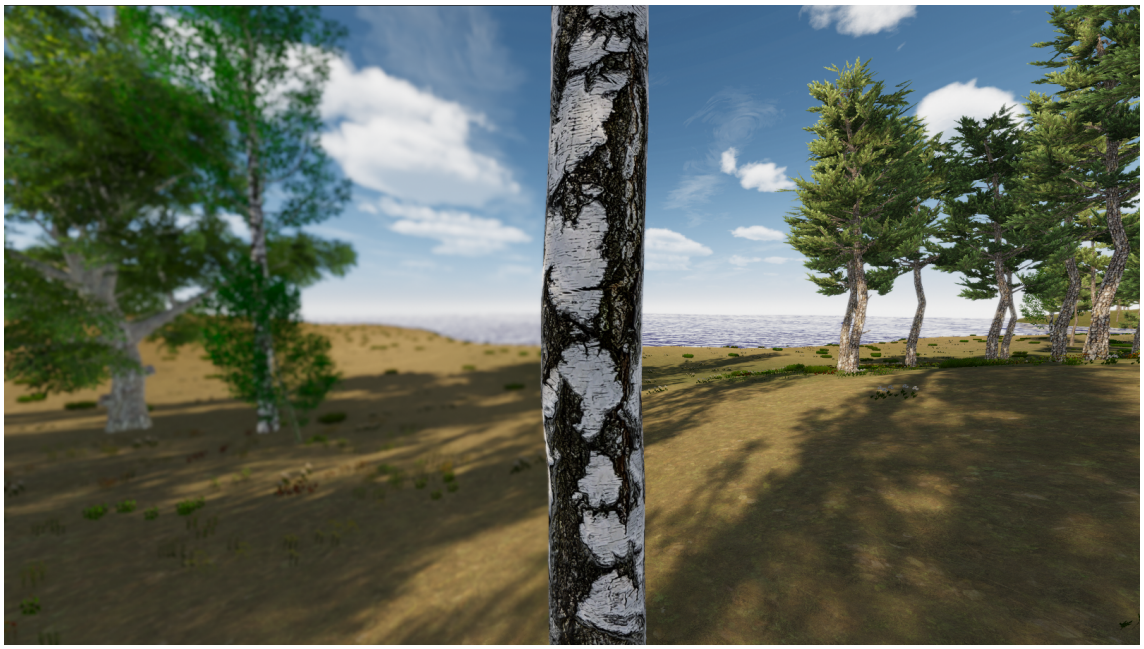

Figure 5.10: A comparison of the depth of field effect used in the environment, enabled to the left of the tree and disabled to the right.

Motion blur is the effect that is felt when moving one's head quickly from one side to another. This effect is simply applied to the camera, to make it more realistic when making rapid movements in the environment.

By default, Unity supplies the environment with a light-blue skybox. In Unity, a skybox is a six-sided cube of textures that is drawn behind the terrain, to form a background. To enhance the realism of the project in this aspect, an external skybox was added from an asset in the asset store. This skybox, which can be seen

in Figure 5.11, contains clouds in the sky, the sun, and also a thicker cloud cover that adds shadows appropriately.

Figure 5.11: The skybox that was used in the project.

Furthermore, to make this skybox dynamic, a script that put a rotation on the skybox was developed. This creates the visual effect of moving clouds in the sky.

5.2.2 Automatic Terrain

In Figure 5.12, one can see a comparison of the Unity editor before and after the "Build Ecosystem" button is pressed, which is the implementation of the aforementioned editor script. This addition makes it easy to generate environments automatically.

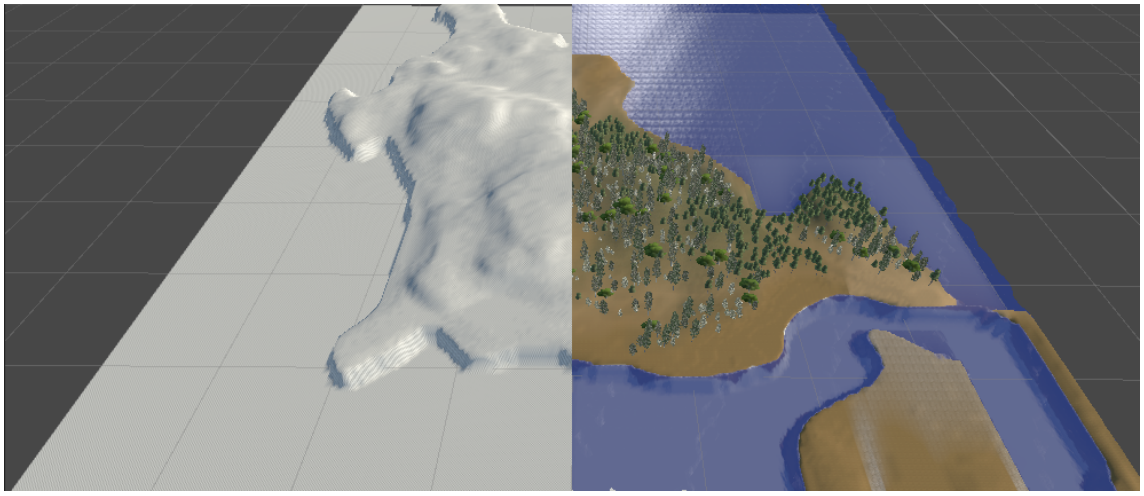

Figure 5.12: Before (left) and after (right) pressing the "Build Ecosystem" button.

In Figure 5.13, the interface for selecting the land-cover data for the desired terrain is shown, which is prompted after pressing the “Build Ecosystem” button.

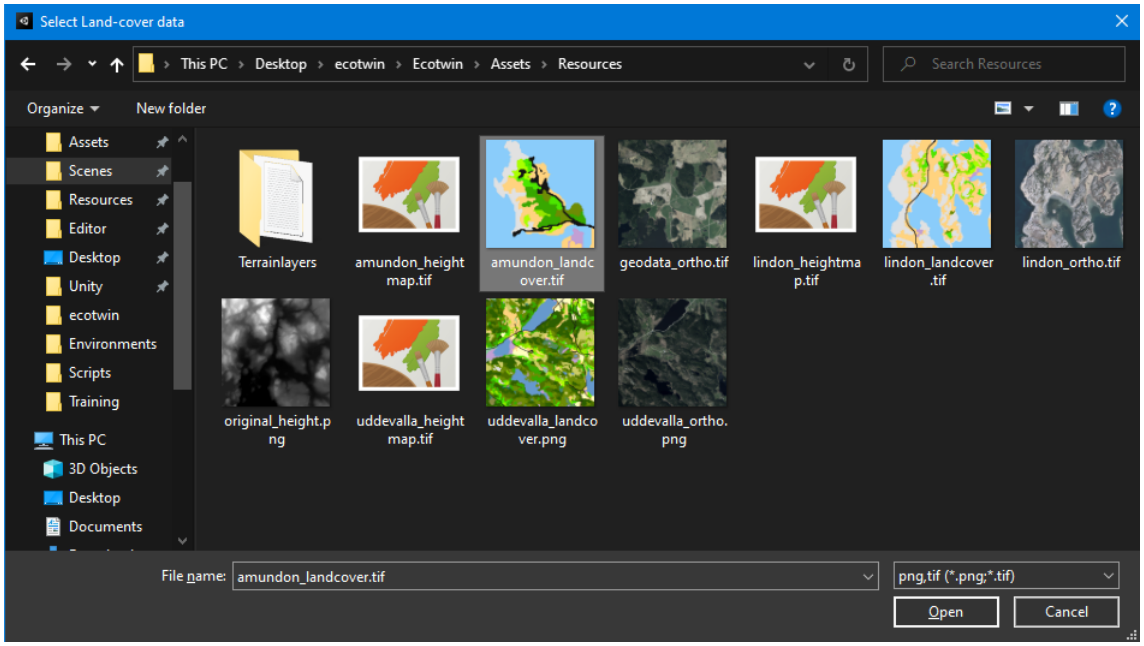

Figure 5.13: Interface to select land-cover data for the environment building.

5.3 Animal Navigation

When the resulting three-dimensional environments were able to be generated from arbitrary locations in Sweden, the work on the reinforcement learning agents was then started in these environments. In this section, the results for the goal of animal navigation using reinforcement learning will be covered.

5.3.1 Navigation

In Appendix A, a comparison between the final stages of the learning between iteration 4 and a model that was trained without curriculum learning can be seen. By observing the training without curriculum learning, it seems like the agents found a local maximum, where they could maximise their rewards by going into the water and thereby minimise their future negative rewards.

5.3.2 Learning Process

A collection of the resulting graphs of the training session with the four iterations can be seen in Figure 5.14 - 5.18.

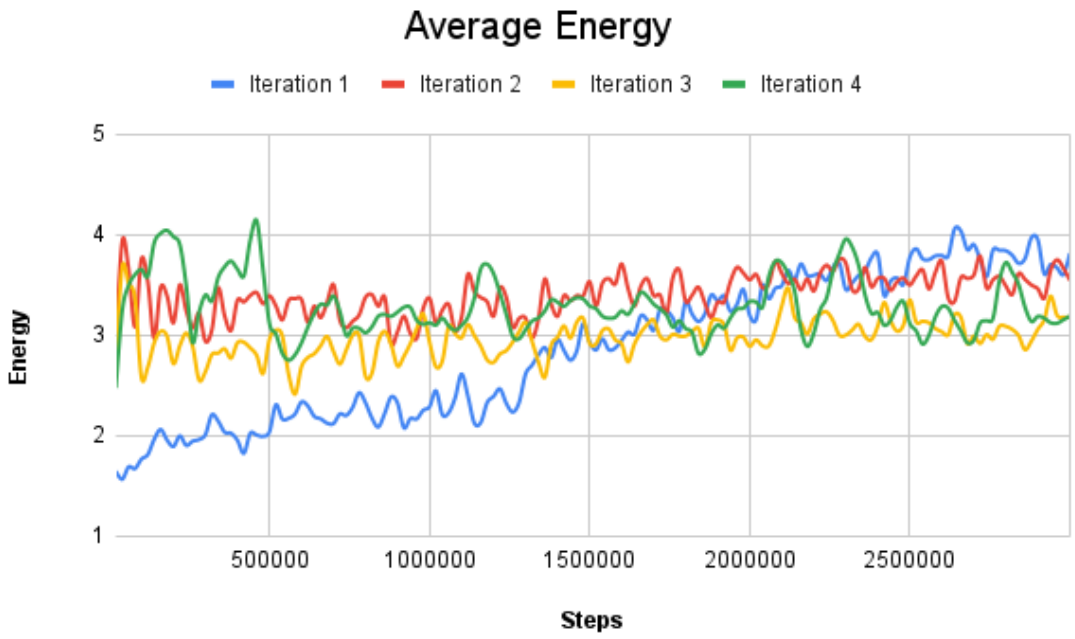

Figure 5.14: The average energy for all agents during training.

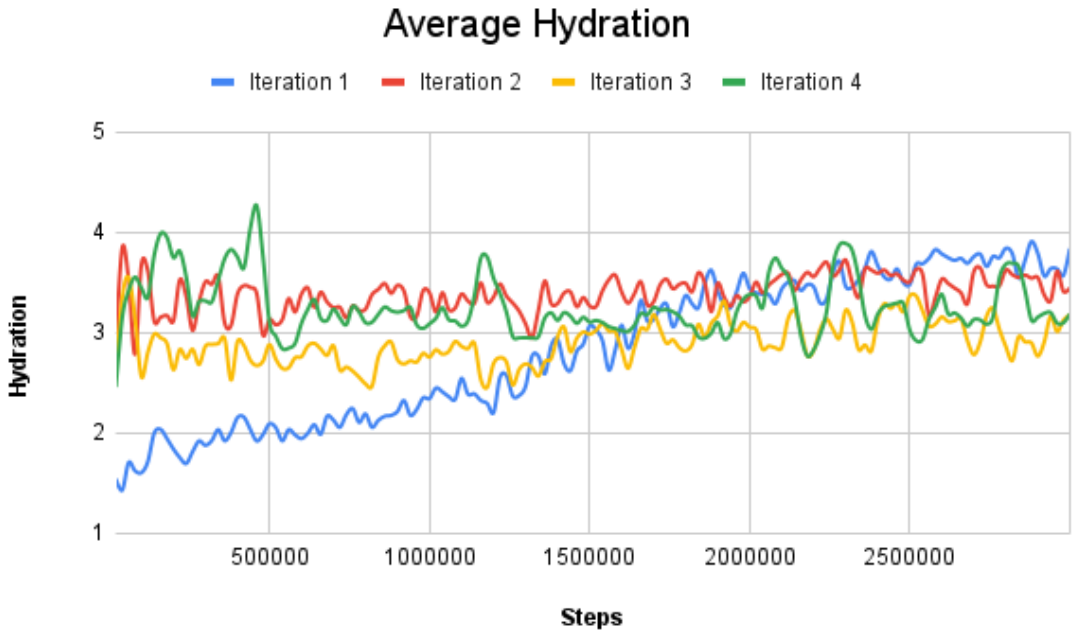

Figure 5.15: The average hydration for all agents during training.

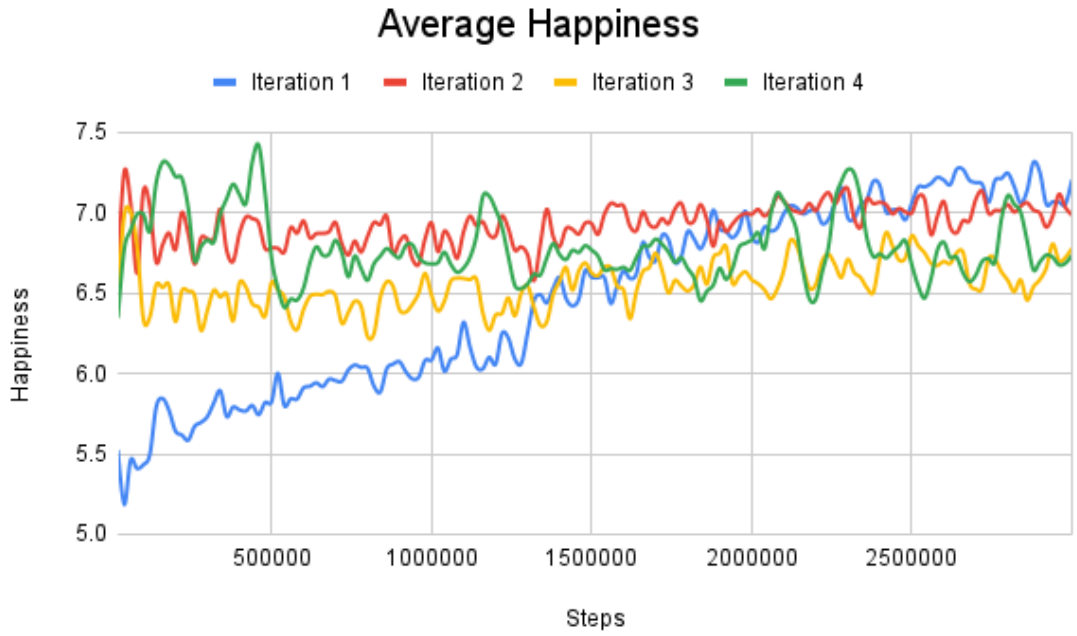

Figure 5.16: The average happiness for all agents during training.

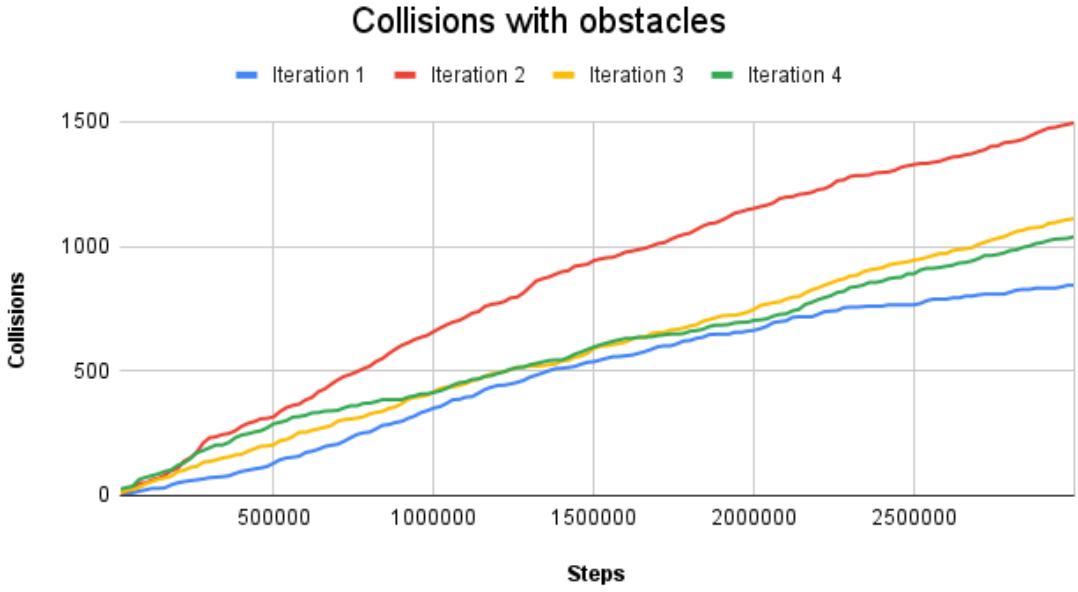

Figure 5.17: The amount of times the agents being trained collided with obstacles during each iteration.

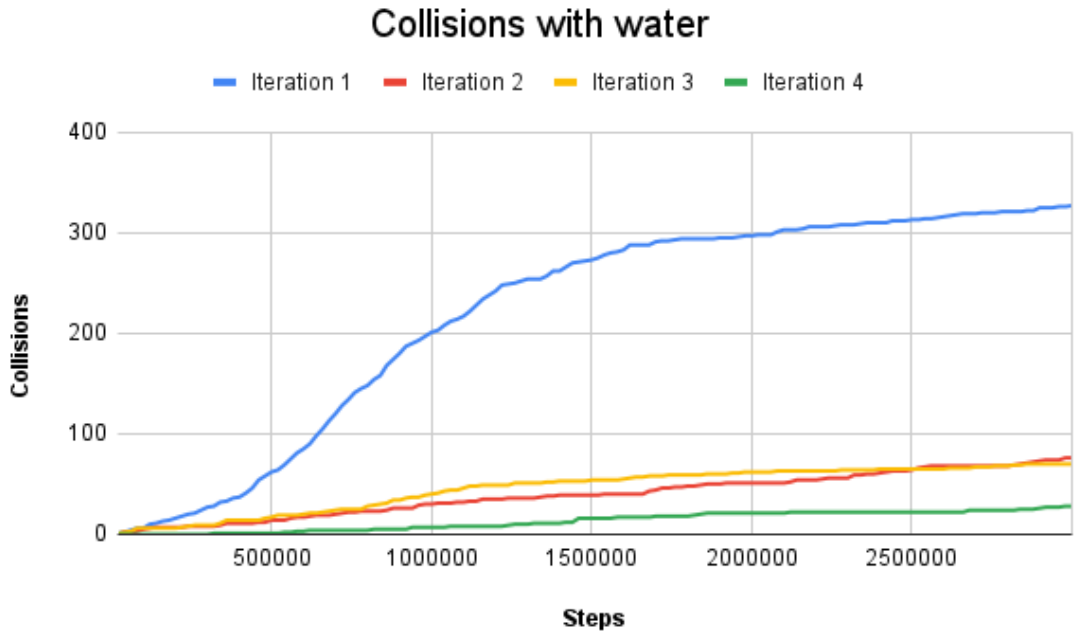

Figure 5.18: The amount of times the agents being trained collided with water during each iteration.

5.3.3 Population Dynamics

In Figure 5.19, the resulting graph of five inference runs is shown, where **60** agents were initially spawned in random positions on the previously mentioned terrain of Lilla Amundön.

Similarly, in Figure 5.20, the resulting graph of the five inference runs with **five** initial agents is shown.

It is worth mentioning that the chance to have offspring was not part of the training, but only done for these inference experiments. The reason for this was primarily to avoid behaviours that might have the agents avoid eating/drinking since doing so to a certain extent will likely result in the loss of resources (the transfer of resources to the child), which the agent would perceive as a negative reward. Since there is no other positive reward for reproduction, the agent would likely learn to not eat and drink sufficiently to reproduce, which would ruin the purpose of the experiments.

To show the importance of the fourth iteration, inference tests of the third iteration were conducted. In these tests, it was noticed that the agents were not able to keep a consistent population and did not seek out new areas of the island once they settled in the southern part of the environment. A population experiment with iteration 3 can be found in Appendix B.

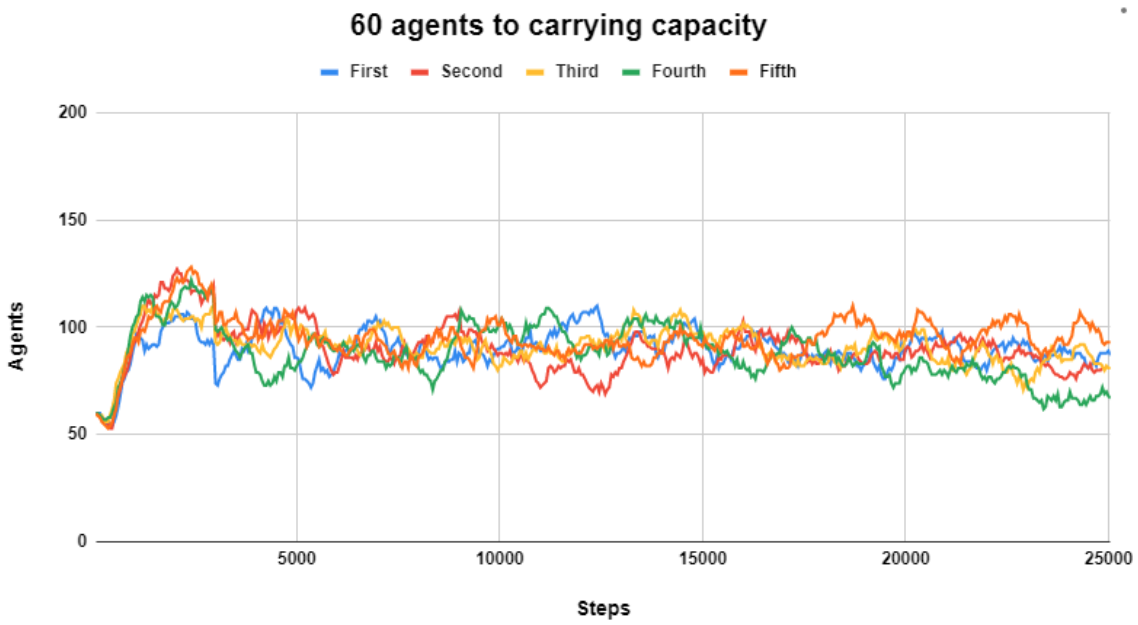

Figure 5.19: Five inference runs with the chance to spawn offspring depending on gathered resources, with 60 agents initially.

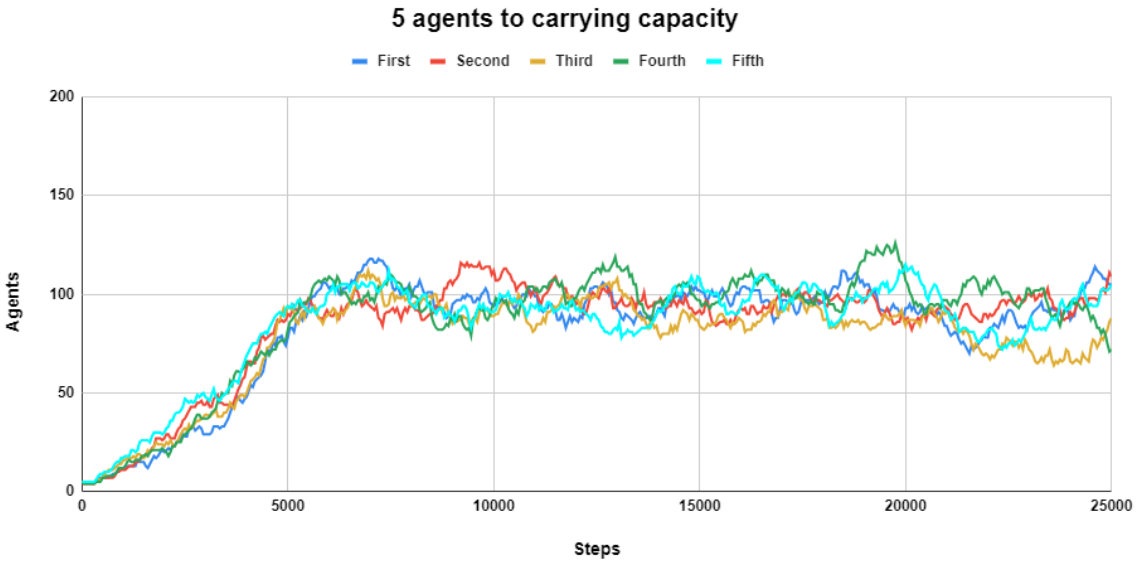

Figure 5.20: Five inference runs with the chance to spawn offspring depending on gathered resources, with five agents initially.

5.3.3.1 Human Intervention

For the hunting experiment, Figure 5.21 shows five iterations of this experiment. The black triangles represented the points where a shift in hunting pressure occurred, which is measured on the right vertical axis. The hunting percentage stayed at 0% for 7500 steps, and then for each subsequent 7500 steps, it increased by 5%. This increase was done until the population reached extinction.

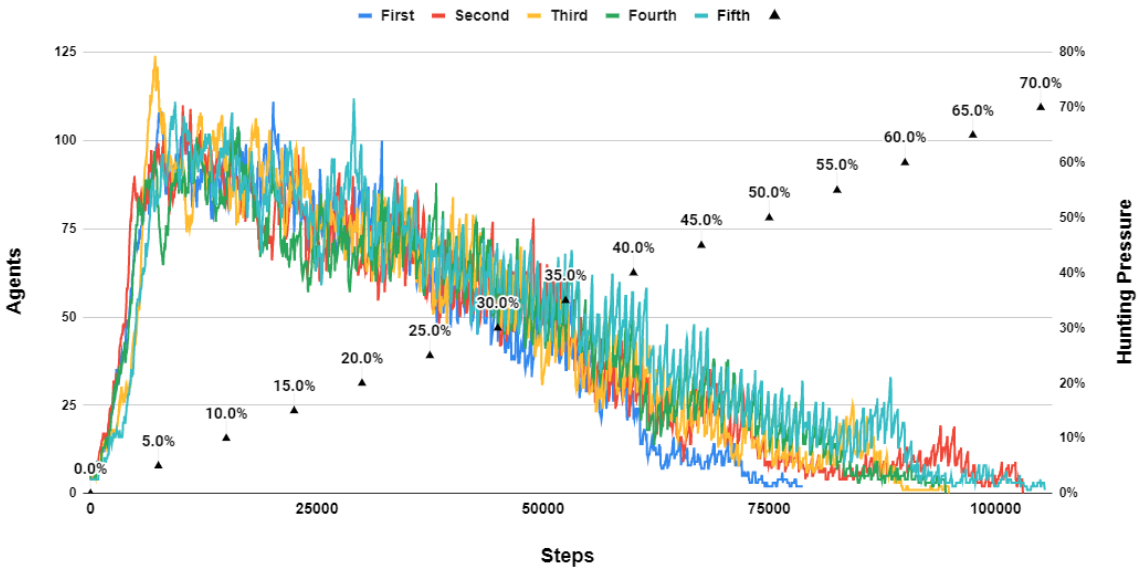

Figure 5.21: Five inference runs with the chance to spawn offspring depending on gathered resources, with the addition of hunting.

For the sea level experiment, which can be seen in Figure 5.22, five instances were run. The black triangles in this figure represented the points in time when the sea level was increased by one meter, which is measured on the right vertical axis.

The resulting graph of the road-building experiment can be seen in Figure 5.23. The black triangles here meant the increase in the lethality of the road, i.e. how likely it was for an agent to die if it traversed on the road.

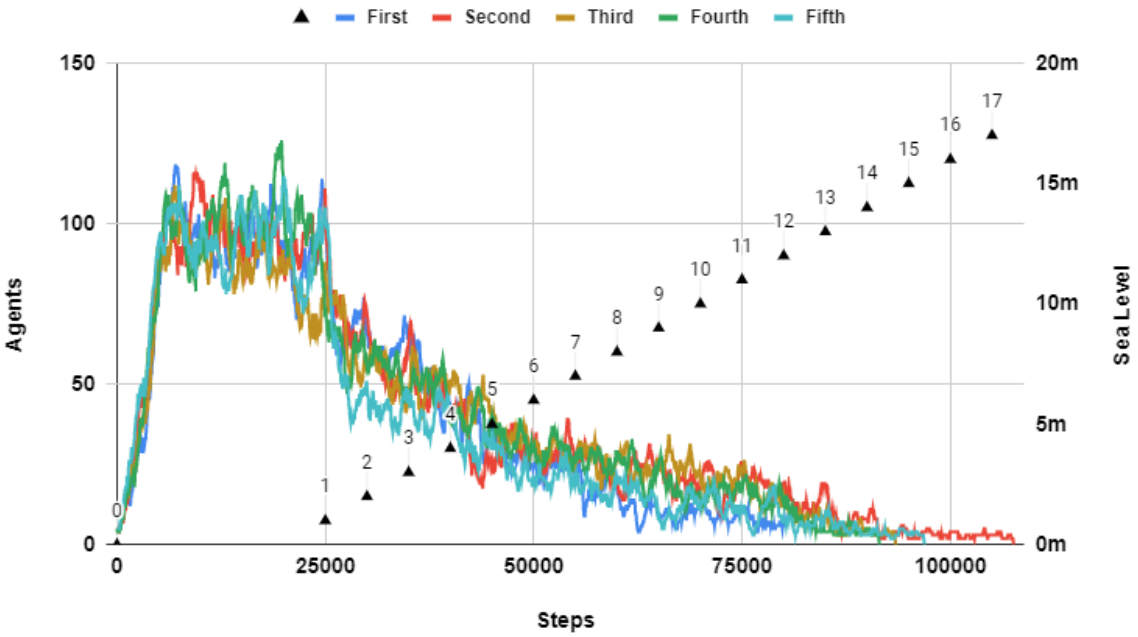

Figure 5.22: Five inference runs with the aspect of sea level increase included.

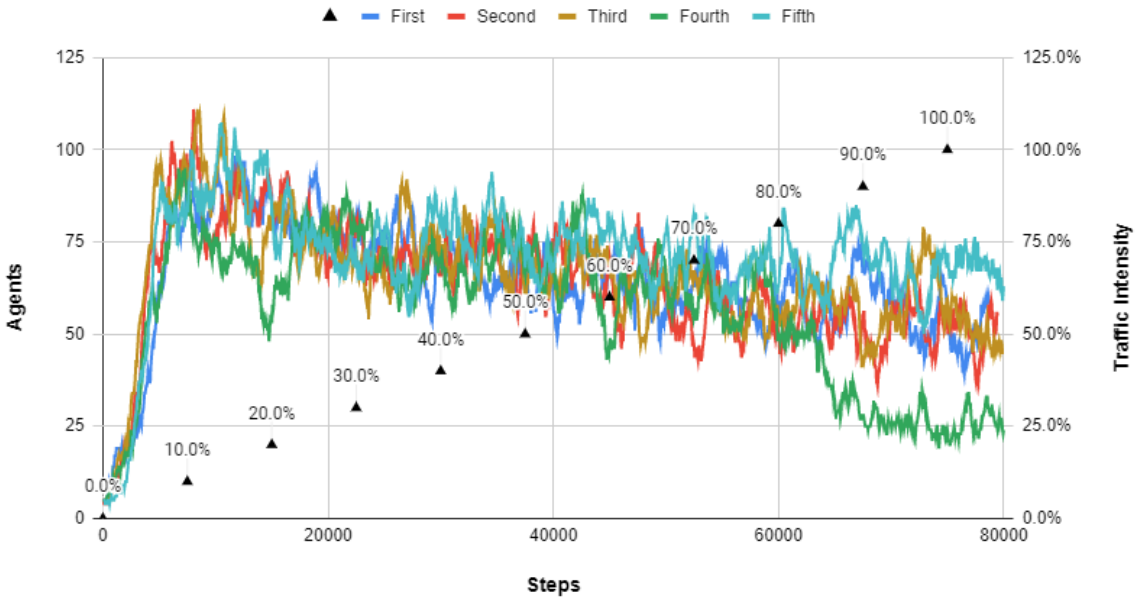

Figure 5.23: Five inference runs with the addition of a road.

6
Discussion

This section will cover discussions about the results regarding the three-dimensional terrain generation with satellite data, as well as the realistic animal behaviour in Unity, while also discussing possible future work.

6.1 Terrain

Regarding the terrain, the goal to model any location of Sweden as a three-dimensional environment was accomplished. With the proper land-cover data and heightmap data, such a three-dimensional environment can be produced in Unity. Examples of such generated environments were shown in Figures 5.3 and 5.4. There is no exact measure of how close to reality the produced environment is, but it strongly depends on the accuracy of the land-cover data.

6.2 Training

As mentioned earlier, curriculum learning was used to speed up training, by dividing the learning aspects into several steps. In Appendix A, it is easy to see that the addition of curriculum learning is significant in making the agents learn faster, thus also making the entire training process faster. However, because the positions the agents spawned at had a big impact on the success of the agent, the curves can appear unstable, meaning that it could be uncertain if they were learning properly. Ideally, the curriculum stages should have been utilised such that the happiness/hydration/energy was able to increase consistently or stabilise. Despite this, the inference tests did show that the agents improved between iterations, as can be seen when comparing Appendix B to Figure 5.20.

As can be seen in Figures 5.14, 5.15 and 5.16, the agents are able to adapt to a more complex environment. Because of the reward function that is used (i.e. delta happiness, mentioned in Section 4.6), the theoretical maximum happiness is $\ln 51 + \ln 51 \approx 7.86$. However, the agents are very unlikely to achieve this happiness due to the need to travel between resources, which will simultaneously drain both of them. As the resources become fewer and more spread out, both the energy and hydration fluctuate much more, as can be seen in iterations 3 and 4 of the training. Iteration 4 in particular seems to vary a lot compared to the other iterations. This can be attributed to the fact that agents can spawn in areas where there are no resources, and also the fact that twice the number of agents are now competing for

a limited number of resources.

Regarding Figure 5.17, a possible reason why the total amount of collisions increase after the first iteration is likely because the agents only spawn in areas with no obstacles (open fields with vegetation) during this first iteration. This essentially means that the agents do not get any training in this regard until further iterations, where the agents spawn more randomly across the entire terrain.

Another reason could be that colliding with an obstacle does not mean that the agent dies, but that the agent loses -1 for both resources. Should a resource be close to a tree, the agent might prioritise this reward and tolerate the negative reward it would get if it collides with the obstacle.

The accuracy of the state could be another factor. Since the state encodes everything as one meter, it might sometimes be difficult for the agent to gauge where exactly in the pixel it is located, since float values get rounded down to the closest integer. An image of this can be seen in Appendix C. This could cause a mismatch where the same state and actions could lead to different outcomes, something you ideally would not want in a reinforcement learning environment. A solution to this issue could be to model smaller details, for example, one pixel being ten centimetres. However, this would require a much larger image to represent the same viewing area than what the agent currently has.

In Figure 5.18, a clear decrease in water collisions can be seen. Already in the first iteration, the line seems to flatten out and the other 3 iterations have quite low collisions. Similar to the obstacles, one possible reason is that the agent tries to eat resources close to the water and is not able to turn away from the water. This can occur due to the agent only being able to rotate as a part of a movement, and therefore it is sometimes not able to change its trajectory away from the water in time.

Finally, the project only used the predefined network structures readily available in the configuration file of ML-Agents. The training and results of the agents could be further improved by defining a custom network structure, but for this project, this was not pursued due to time constraints.

6.3 Animations

Furthermore, one feature that could be discussed is the animator controller. The one that is currently being used is a minimal version, only containing the most basic actions. Since this minimal version saw success in the training results and inference experiments, a future project could work on implementing more actions, such as sleeping. On the one hand, this would increase the complexity of the agents, but on the other hand, it would add more realism to the agents.

A possible issue with the current approach is that one action taken by the agent

might not result in that animation being played, as mentioned in Section 4.2. For example, the eating animation is longer than the run animation, which means that when the agent is eating and subsequently decides to move, it might take more than one action to trigger the physical movement. This setup implies that the agent would have to learn that triggering an eat action means that it might have to learn to take two running actions after eating, to start running. Although this setup is unorthodox, due to one action not resulting in a change of state, it might also make it more realistic in the sense that even though the agent is thinking about its next resource, it cannot leave until it has finished eating.

Another improvement would be to create custom three-dimensional models and animations. This would allow the action space to be more detailed rather than relying on pre-made animations as actions. It could then be possible to have more accurate and detailed motions to have more control over the exact movements of the agents.

6.4 Population Experiments

As was shown in Figure 5.19, the five inference runs with reproduction resulted in the population stabilising around a carrying capacity of around 70-80 agents. This indicates that the ecosystem can reach stability and could thus be further used for other tests. It is also worth noticing that this carrying capacity was reached regardless of the agents' initial positions or population size, as can be seen when comparing Figure 5.19 and Figure 5.20.

The parameters that control the population in these tests can be tweaked so that they could support a larger population by for instance adding more resources or decreasing the time it takes for them to regrow. Nevertheless, the fact that the population manages to find a carrying capacity implies that the animals can navigate to new areas to find resources and survive for some time.

6.4.1 Population Experiments with Human Intervention

In the hunting experiment, see Figure 5.21, one can quite easily spot that the size of the population takes a hit when the hunting becomes excessive. Another observation would be that hunting around 5% to 20% of the population seems to not have drastic consequences, as the graph looks decently stable around 75 agents, even though there is a slight downwards trend. However, when the percentage rises to 20% and above, the population starts decreasing in size. Tests like these could thus be used, with the correct parameters, to help determine how many animals can safely be killed during a hunting season, while still maintaining a healthy population size for the specific species.

In the sea level experiment (Figure 5.22), one can observe a steady decline in population size when the sea levels increase. The fact that they become extinct at a

certain sea-level increase might seem like an obvious outcome, but the main takeaway from this experiment should rather be on the specific sea levels. For instance, already after three meters of sea level increase, the carrying capacity seems to have been halved. Important to note however is that the sea level increase likely would impact the environment in ways that are not modelled, such as resource growth being halted due to increased salt levels in the earth and temperature differences, among other factors. Nevertheless, results like these might help increase the knowledge surrounding how animals in a certain ecosystem are affected because of the possible consequences of climate change, which is a very pressing matter in the current situation around the world.

In the road-building experiment (Figure 5.23), the main conclusion that can be drawn is that the population seems to survive the addition of this road, even if the lethality when crossing it is 100%. In some of the simulations, the hare population went all but extinct west of the road, which lead to a severe decrease in carrying capacity, as can be seen most clearly in the fourth run. However, for the runs where the agents manage to survive on the west side of the road as well, the ecosystem as a whole managed to stabilise, even if the carrying capacity was lowered as a consequence. However, if the road was placed in a different location, or if there were to be multiple roads built, the result might have been different.

These kinds of tests could be used to conclude whether or not a specific addition could be built in a specific location, without disturbing any wildlife. An improvement to such a test is to have access to the exact information of where the population of animals is, which the tests performed in this thesis do not have. The more specific the information is, the more accurate the tests will be. Regardless, the tests that were performed without this exact information, still give indications as to whether or not it is a good idea to build the piece of infrastructure in a certain location, since the carrying capacity is still accurate.

These experiments are simplified and many of them are put to the extreme, to simply show the possible applications of the project. For example, hunting up to 70% of a population will likely never even be considered by the governing body that handles the hunting. Tests like these were rather done to see if the population behaved as expected, i.e. not surviving when such harsh events occurred.

6.5 Limitations

There are currently many limitations with the agents and the environments which hinder them from being used to fully model ecosystems.

6.5.1 Agent

The agent is very limited in what actions it can do and how it behaves compared to its real-life counterpart. The agent does not sleep, the reproduction is asexual, and how much one resource aids the agent is also unknown. There is also another

assumption made; the animations that the agents have are how the actual animals behave, which is of course not true in the real world. For instance, hares do not travel the same distance for each jump they do, their movement speeds are not constant, etc.

6.5.2 Environment

The environment is static and does not account for any temperature changes, there is no day-night cycle, which would both influence the resource regrowth time, among other things. Furthermore, the resource placement is not based on actual data but is instead randomised. The system does not currently account for any other living entity that might exist in the environment which could impact it, such as predators.

Regarding the realism of the terrain, it is highly reliant on the land-cover data. If there was a possibility to get access to land-cover data with higher resolution, i.e. more accurate divisions of the classes, the terrain would also become more true to the real location.

6.6 Future Work

In this project, the possibilities for future work are many, listed below are some examples of what could be done:

- Introduce predators, which would force the agents to be more reactive and make it more difficult for them to survive. This would also mean that it would be necessary to prove that the environment can support predators as well, which are different to the hares in that their resources would not be static, but rather dynamic (i.e. moving prey).
- Introduce permanent resource areas such as lakes and make the agents have to navigate between areas to survive.
- Model temperature, wind, seasons or other environmental aspects.
- Introduce sleep cycles for the agents.
- Incorporate other data sources to model the environment such as OpenStreetMap [45]. This data source has more specific data about infrastructure and other man-made objects, which could help provide more features for terrain generation for any future projects.
- To remove the geographical restriction, other land-cover data sources could be considered. One example is Copernicus' global land-cover data [46], which has a lower resolution (100 meters) than the data source used in this thesis (10 meters), but might still be helpful to some degree since it covers the entire world.
- Further tests with different starting populations could be done to ensure that the environment is stable, even though the two tests that were done (Figures 5.19 and 5.20) strongly indicate that this is the case.
- Further experiments on other environments could be done, rather than just the one that was done for this project (Lilla Amundön), to see if similar carrying capacities are reached.

- Gather some real data about animal populations in these environments, to be able to compare the carrying capacity of the population tests against concrete data. This would ensure that the system produces realistic results.

One aspect for the agents to improve on is the usage of planning for the animals. Planning could be used by the agents to look several steps ahead to decide what action to take. This would be especially useful for situations where the agent does not see any resource in its immediate perception. There was a discussion to implement this in the thesis, but it was decided against this since it was expected to be very time-consuming. If it was to be done, each agent should ideally receive a vector that indicates how much food and water is located in each direction, to guide its next action choice, especially when there are no resources in its perception.

Another possible aspect to research is if it could be possible for the agent to learn to traverse environments with multiple levels, representing for example caves or tunnels. This would make the agents have to learn a more advanced level of three-dimensional navigation. Another more complicated version of three-dimensional navigation would be to have animals that can fly or swim, in this scenario they would also have to learn to navigate true three-dimensional environments. Currently, the model only considers heights as another dimension, however this only impacts the agents in steep areas and it was uncertain how much they take the height into account when deciding their paths.

A third aspect of the agents which could be explored is to implement an egocentric perception. This would in essence mean that the input they receive from the grid-sensor would be oriented from their point of view, instead of always being north. This could prevent the agent from learning very specific environment information, for example in one environment it might always give the best result to go south if there are large fields there. By having an egocentric view, the agent would not know if they are going north or south.

Regarding the agent's state, it could also be possible to make it more accurate by obscuring objects behind heights or obstacles. Currently, the agent can see everything within 15 meters of its position through its input channels, but in a real scenario, it is unlikely that it would see what was behind a tree. Nevertheless, this could be motivated by the fact that most animals have a good sense of smell, which would allow them to sense what they cannot see, but this is not modelled explicitly.

To improve the realism of the experiments that were done in this project, one could also consider extending the reproduction model by adding sexes and sexual reproduction based on physical encounters. This would mean that there would need to be two hares present to reproduce, they would need to be of different sexes, need a certain level of resources, be close to each other, etc.

A redesign of the curriculum learning iterations could also be considered. One example would be to instead of having the agents spawn at random in the environment,

there could be an iteration that had agents only spawn close to obstacles, which would force them to learn to navigate past trees to get to a reward. Another scenario might be that the agent has to navigate from areas with no resources to areas with resources. This could lead to more stable learning since the spawn location of the agent would not matter as much in the success of its learning.

6.7 Ethical Aspects

In this thesis, there are both positive and potentially negative aspects to discuss regarding ethics and sustainability.

On one hand, this project could be used to analyse sustainability in the environment, by performing different experiments like mentioned above. As shown by these experiments, this project could for instance:

- Help improve decision making regarding which infrastructure expansions are reasonable and could be ethically motivated with concern to wildlife.
- Help determine what extent of hunting should be done for certain species.

This project could also act as another motivation to combat climate change when seeing how it affects certain ecosystems and animal populations. In this thesis, the consequence of sea level increases was observed, but other consequences could also be analysed, such as how the animals' food/water sources are affected, how their breeding is affected, etc.

On the other hand, there are also potential risks to consider. As with all projects that involve some sort of artificial intelligence, there are always malicious use cases. In the context of this thesis, one such potential use case could be that the agents that were trained in a specific environment, might be useful in some kind of military sense, which could be dangerous if they received more accurate input for their perceptions.

However, as the project stands now, this is considered extremely unlikely, and the use cases are more limited to the analysis of ecosystems than anything else.

7
Conclusion

Considering the available data, the goal of achieving three-dimensional environments using satellite data is considered as reached. The usage of land-cover data incorporates what was desired; satellite data. Many of the aspects that were needed for realism were done programmatically, like tree placement, texturing the terrain, adding vegetation, a moving sun in the sky, etc. But as mentioned above, it could still be improved upon.

The goal of animal navigation was also successful. The results of the training graphs, the initial inference runs that showed a carrying capacity, and the subsequent population experiments together show that the training has been successful. The agents learned to move around to consume the resources they needed to avoid dying of resource shortages, while also avoiding bodies of water and mostly avoiding trees. However, a big improvement for the realism regarding the population tests would be to have someone more knowledgeable in biology/with animals tweak the parameters, such that they could be more accurately modelled. These parameters would for instance be how to properly model the lifespan of the animals, how many offspring they can have, how often they have offspring, their metabolism, etc.

In conclusion, the quality of the next iterations of digital ecosystems could improve heavily with the use of satellite data and machine learning in combination, as this project shows.

Bibliography

[1] "Ecosystem," (Accessed: 06-June-2022). [Online]. Available: https://education.nationalgeographic.org/resource/ecosystem

[2] V. Volterra, "Fluctuations in the Abundance of a Species considered Mathematically," *Nature*, vol. 118, no. 2972, pp. 558–560, Oct 1926. [Online]. Available: https://doi.org/10.1038/118558a0

[3] Ecotwin. (Accessed: 13-December-2021). [Online]. Available: https://www.ecotwin.se/

[4] Ecotwin - GitLab. (Accessed: 09-June-2022). [Online]. Available: https://gitlab.com/ecotwin/ecotwin/

[5] T. Johnson, "Ecosystem," [Video game], 2021. [Online]. Available: https://store.steampowered.com/app/1133120/Ecosystem/

[6] Unity Real-Time Development Platform. (Accessed: 29-November-2021). [Online]. Available: https://unity.com/

[7] Government & Aerospace. (Accessed: 02-December-2021). [Online]. Available: https://unity.com/solutions/government-aerospace

[8] R. M. Smelik, K. J. De Kraker, T. Tutenel, R. Bidarra, and S. A. Groenewegen, "A Survey of Procedural Methods for Terrain Modelling," in *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, 2009, pp. 25–34.

[9] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," *CoRR*, vol. abs/1912.06680, 2019. [Online]. Available: http://arxiv.org/abs/1912.06680

[10] *Reinforcement Learning Agents acquire Flocking and Symbiotic Behaviour in Simulated Ecosystems*, vol. ALIFE 2019: The 2019 Conference on Artificial Life, 07 2019. [Online]. Available: https://doi.org/10.1162/isal_a_00148

[11] E. Alonso, M. Peter, D. Goumard, and J. Romoff, "Deep Reinforcement Learning for Navigation in AAA Video Games," *CoRR*, vol. abs/2011.04764, 2020. [Online]. Available: https://arxiv.org/abs/2011.04764

[12] J. Schulman, O. Klimov, F. Wolski, P. Dhariwal, and A. Radford, "Proximal Policy Optimization," (Accessed: 29-November-2021). [Online]. Available: https://openai.com/blog/openai-baselines-ppo/

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[14] "Proximal Policy Optimization," 2018. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html

[15] Unity ML-Agents Toolkit. Accessed: 28-January-2022. [Online]. Available: https://github.com/Unity-Technologies/ml-agents

[16] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A General Platform for Intelligent Agents," *CoRR*, vol. abs/1809.02627, 2018. [Online]. Available: http://arxiv.org/abs/1809.02627

[17] A. Juliani, A. Khalifa, V. Berges, J. Harper, H. Henry, A. Crespi, J. Togelius, and D. Lange, "Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning," *CoRR*, vol. abs/1902.01378, 2019. [Online]. Available: http://arxiv.org/abs/1902.01378

[18] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey," *CoRR*, vol. abs/2003.04960, 2020. [Online]. Available: https://arxiv.org/abs/2003.04960

[19] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, p. 9, May 2016. [Online]. Available: https://doi.org/10.1186/s40537-016-0043-6

[20] J. Yamada, J. Shawe-Taylor, and Z. Fountas, "Evolution of a Complex Predator-Prey Ecosystem on Large-scale Multi-Agent Deep Reinforcement Learning," *CoRR*, 2020. [Online]. Available: https://arxiv.org/abs/2002.03267

[21] C. Strannegård, W. Xu, N. Engsner, and J. A. Endler, "Combining Evolution and Learning in Computational Ecosystems," *Journal of Artificial General Intelligence*, vol. 11, no. 1, pp. 1–37, 2020. [Online]. Available: https://doi.org/10.2478/jagi-2020-0001

[22] Y. Yang, L. Yu, Y. Bai, J. Wang, W. Zhang, Y. Wen, and Y. Yu, "An Empirical Study of AI Population Dynamics with Million-agent Reinforcement Learning," *CoRR*, vol. abs/1709.04511, 2017. [Online]. Available: http://arxiv.org/abs/1709.04511

[23] *A Sustainable Ecosystem through Emergent Cooperation in Multi-Agent Reinforcement Learning*, ser. ALIFE 2021: The 2021 Conference on Artificial Life, 07 2021, 74. [Online]. Available: https://doi.org/10.1162/isal_a_00399

[24] H. Overweg, H. N. C. Berghuijs, and I. N. Athanasiadis, "CropGym: a Reinforcement Learning Environment for Crop Management," 2021. [Online]. Available: https://arxiv.org/abs/2104.04326

[25] M. Lapeyrolerie, M. S. Chapman, K. E. A. Norman, and C. Boettiger, "Deep Reinforcement Learning for Conservation Decisions," 2021. [Online]. Available: https://arxiv.org/abs/2106.08272

[26] Google Earth. (Accessed: 29-November-2021). [Online]. Available: https://earth.google.com/web/

[27] P. Dam, F. Duarte, and A. B. Raposo, "Terrain Generation Based on Real World Locations for Military Training and Simulation," *2019 18th Brazilian*

Symposium on Computer Games and Digital Entertainment (SBGames), pp. 173–181, 2019.

[28] Wikipedia, "Heightmap - Wikipedia, The Free Encyclopedia," http://en.wikipedia.org/w/index.php?title=Heightmap&oldid=1058499180, 2021, [Online; accessed 23-January-2022].

[29] ——, "Displacement mapping - Wikipedia, The Free Encyclopedia," https://en.wikipedia.org/wiki/Displacement_mapping#:~:text=Displacement%20mapping%20is%20an%20alternative,often%20along%20the%20local%20surface, 2021, [Online; accessed 04-April-2022].

[30] ——, "Bump mapping - Wikipedia, The Free Encyclopedia," https://en.wikipedia.org/wiki/Bump_mapping#:~:text=Bump%20mapping%20is%20a%20texture,perturbed%20normal%20during%20lighting%20calculations., 2022, [Online; accessed 04-April-2022].

[31] "Cities: Skylines online heightmap generator," (Accessed: 27-February-2022). [Online]. Available: https://github.com/sysoppl/Cities-Skylines-heightmap-generator

[32] "Cities Skylines," (Accessed: 27-February-2022). [Online]. Available: https://www.paradoxinteractive.com/games/cities-skylines/about

[33] "SLU. Science and Education for Sustainable Life," Jan 2022. [Online]. Available: https://www.slu.se/en/

[34] Naturvårdsverket, "Nationella Marktäckedata (NMD)," [Online; accessed 23-January-2022]. [Online]. Available: https://www.naturvardsverket.se/verktyg-och-tjanster/kartor-och-karttjanster/nationella-marktackedata

[35] "QGIS. A Free and Open Source Geographic Information System," (Accessed: 10-February-2022). [Online]. Available: https://www.qgis.org/en/site/

[36] "Post-processing and full-screen effects," (Accessed: 03-March-2022). [Online]. Available: https://docs.unity3d.com/Manual/PostProcessingOverview.html

[37] Unity Asset Store. (Accessed: 12-December-2021). [Online]. Available: https://assetstore.unity.com/

[38] Unity Assets - Animal Pack. (Accessed: 12-December-2021). [Online]. Available: https://assetstore.unity.com/packages/3d/characters/animals/animal-pack-deluxe-99702

[39] Unity Physics Engine. (Accessed: 29-November-2021). [Online]. Available: https://docs.unity3d.com/Manual/PhysicsSection.html

[40] "Ground Fitter," (Accessed: 08-May-2022). [Online]. Available: https://assetstore.unity.com/packages/tools/animation/ground-fitter-121798

[41] Grid Sensors for Unity ML-Agents - Version 2.0. Accessed: 5-March-2022. [Online]. Available: https://github.com/mbaske/grid-sensor

[42] Wikipedia, "Carrying capacity - Wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Carrying_capacity, 2022, [Online; accessed 25-April-2022].

[43] "Grass and other details," (Accessed: 01-April-2022). [Online]. Available: https://docs.unity3d.com/Manual/terrain-Grass.html

[44] Standard Assets (for Unity 2018.4). (Accessed: 26-April-2022). [Online]. Available: https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-for-unity-2018-4-32351#content

[45] "OpenStreetMap," (Accessed: 10-February-2022). [Online]. Available: https://www.openstreetmap.org/#map=16/57.6001/11.9125

[46] M. Buchhorn, B. Smets, L. Bertels, B. D. Roo, M. Lesiv, N.-E. Tsendbazar, M. Herold, and S. Fritz, "Copernicus Global Land Service: Land Cover 100m: collection 3: epoch 2019: Globe," 2020. [Online]. Available: https://zenodo.org/record/3939050

A

Appendix 1

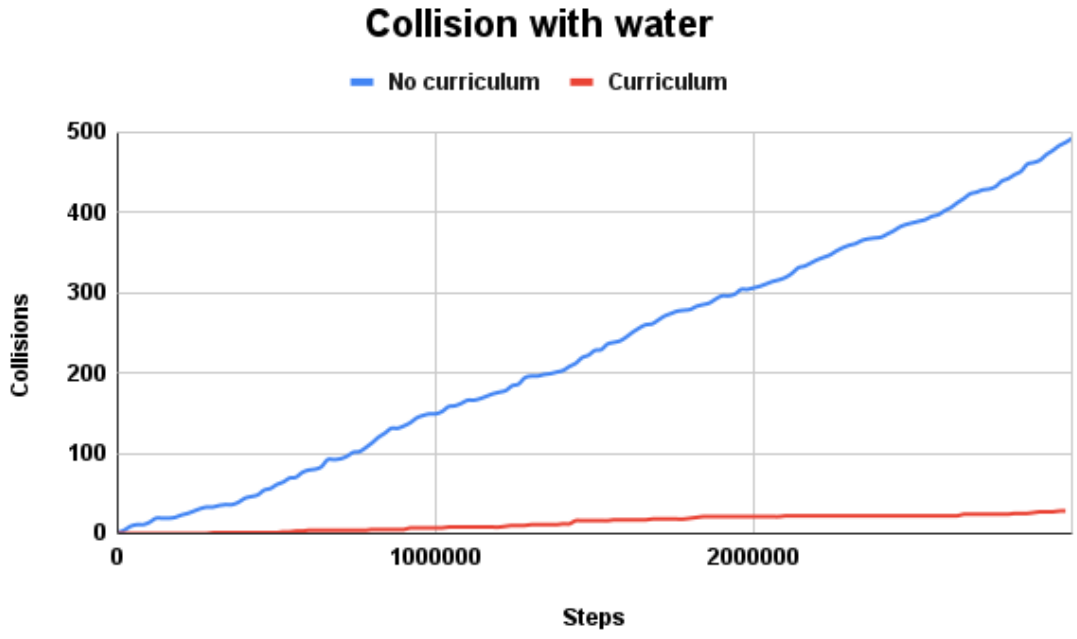

Figure A.1: Comparison between iteration 4 which trained with curriculum learning and the final three million steps of a training session without curriculum.

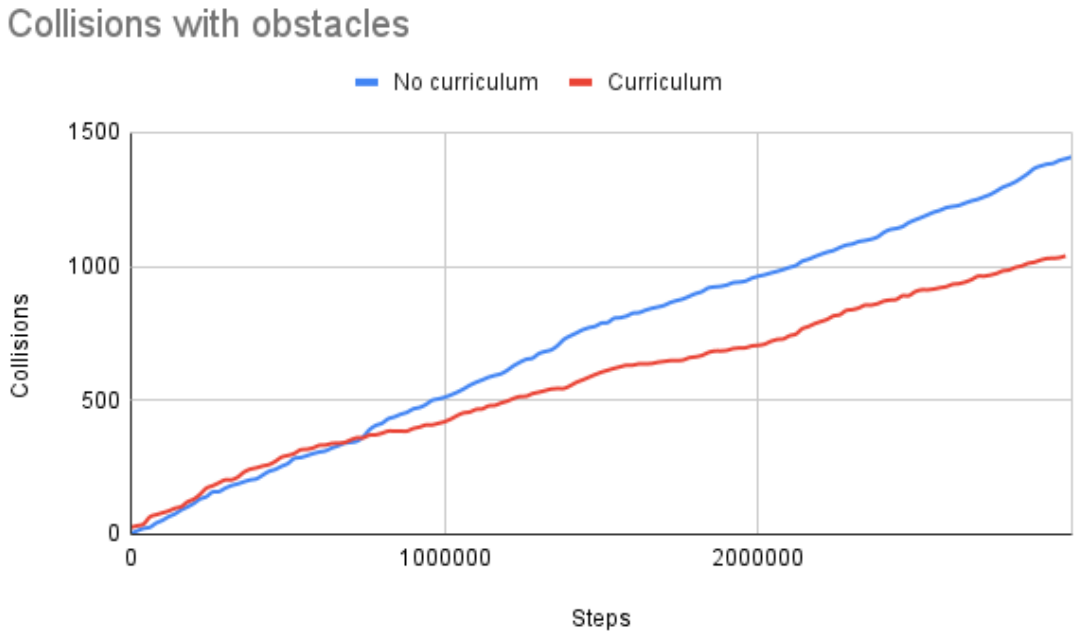

Figure A.2: Comparison between iteration 4 which trained with curriculum learning and the final three million steps of a training session without curriculum.

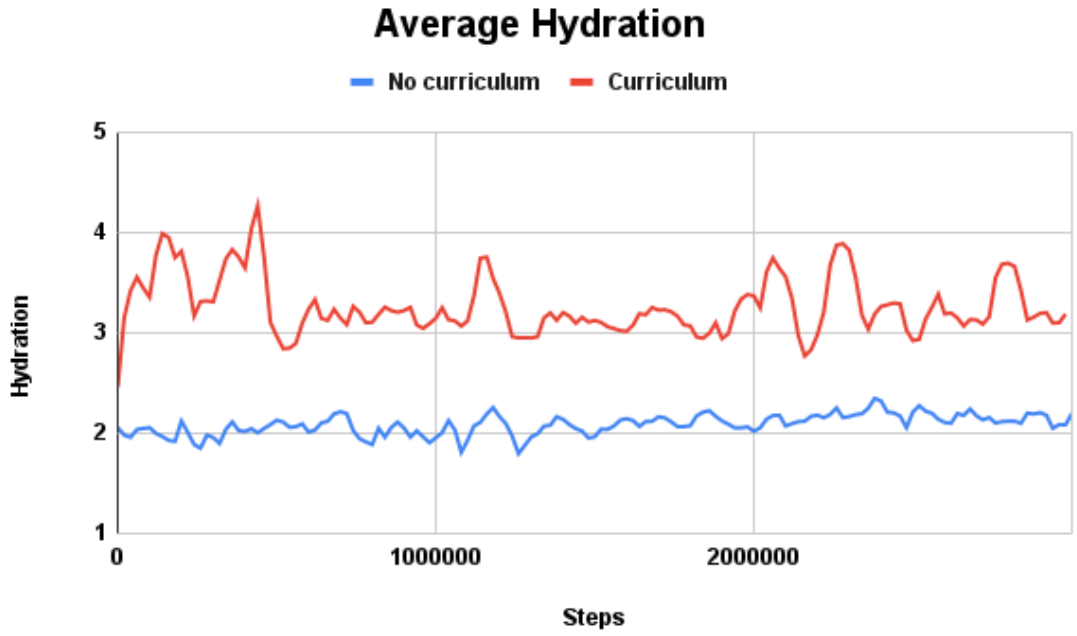

Figure A.3: Comparison between iteration 4 which trained with curriculum learning and the final three million steps of a training session without curriculum.

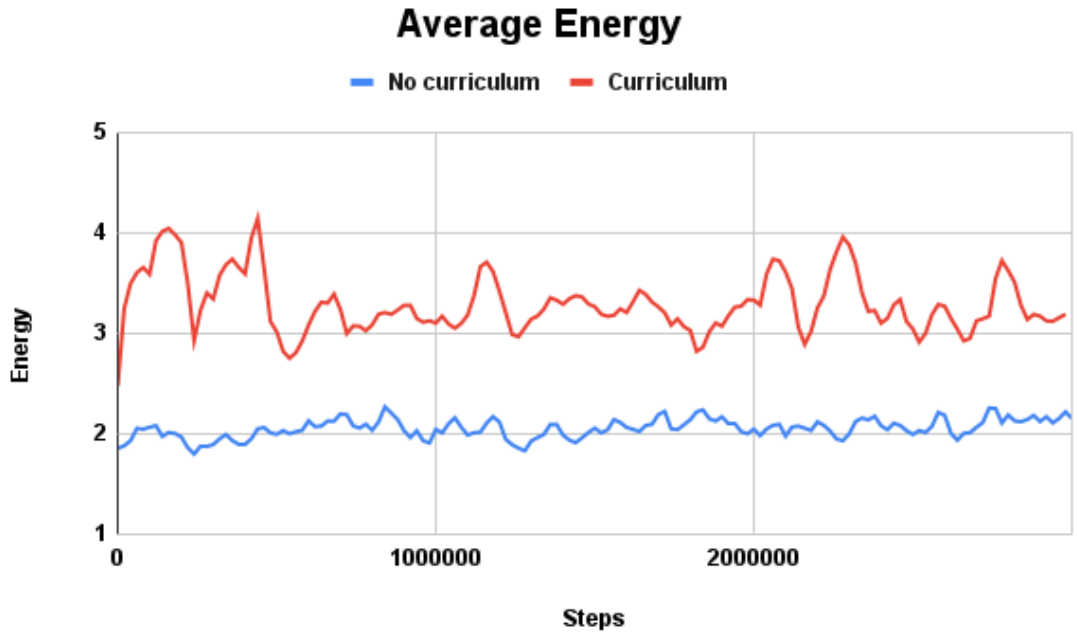

Figure A.4: Comparison between iteration 4 which trained with curriculum learning and the final three million steps of a training session without curriculum.

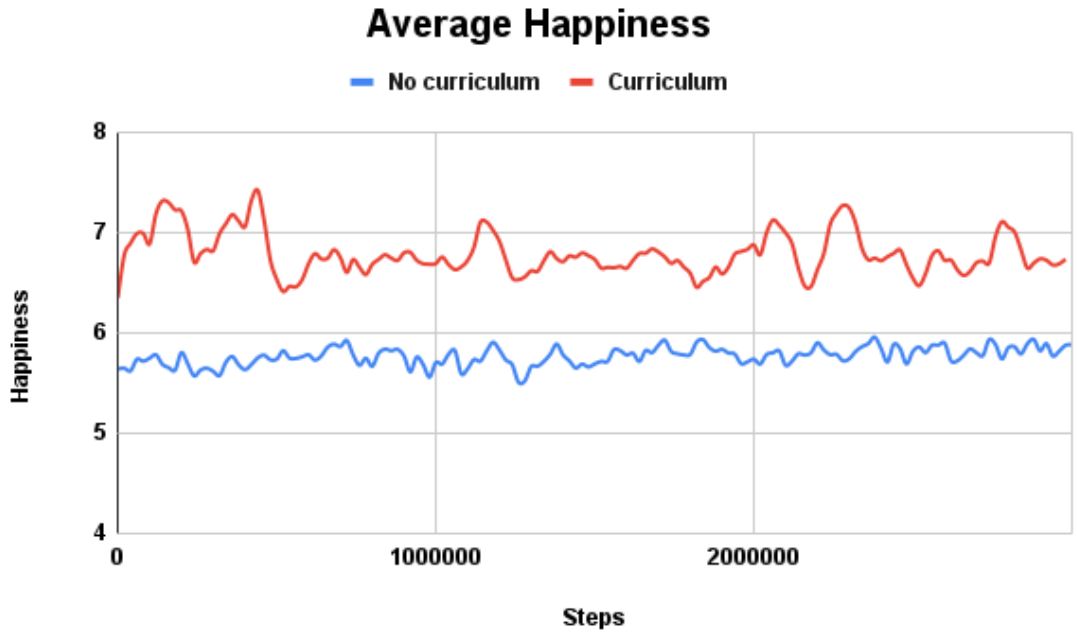

Figure A.5: Comparison between iteration 4 which trained with curriculum learning and the final three million steps of a training session without curriculum.

B

Appendix 2

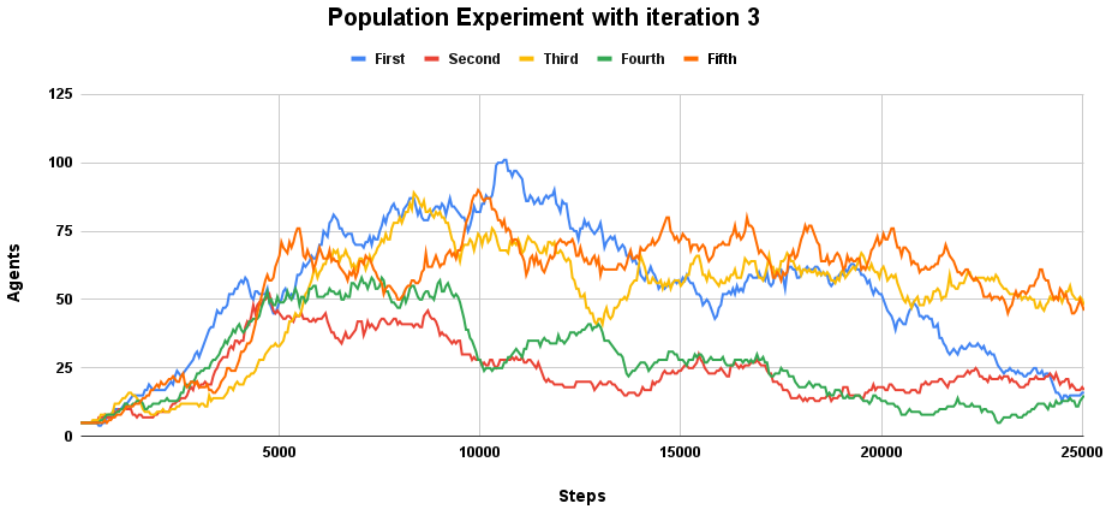

Figure B.1: Five runs of a population experiment with three iterations of training done. A downward trend can be observed for the populations, which implies that they are unable to fully utilise the resources of the environment.

C

Appendix 3

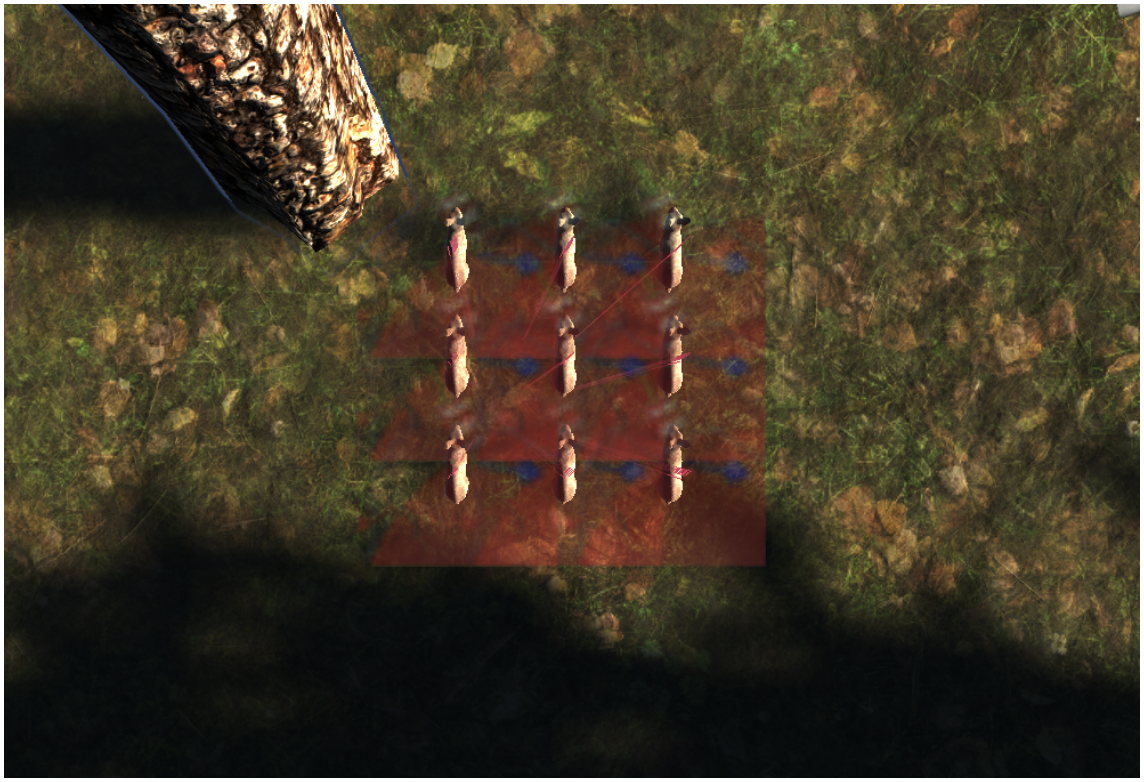

Figure C.1: Shows nine agents standing next to each other, however, in the state they are considered to be in the same position.