Leopold-Franzens-Universität Innsbruck

Department of Computer Science
Interactive Graphics and Simulation Group

Bachelor Thesis

Map Synthesis for Low-Poly 3D Scenes using Generative Adversarial Networks

Tobias Christoph
tobias.christoph@student.uibk.ac.at

advised by
Stephanie Autherith
Matthias Harders

Innsbruck, January 10, 2022

Abstract

In our ever growing digital world, the generation of novel data has become an increasingly vital role in all aspects of content generation. Procedural content generation (PCG) applied to the entertainment field of computer science is one of many examples where the synthesis of new content has been progressively enhanced by the application of machine learning. Generative Adversarial Networks (GAN), a popular deep learning framework, are applied to the creation of low-poly maps. In particular, the Pix2Pix approach will be utilised to generate elevation as well as biome-maps. Our approach also enables users to condition their desired maps both structurally and artistically. The input to the system will merely be a very minimal effort sketch that represents the area-of-play. For example, game designers could utilise this approach to easily create a number of diverse maps for video games. The generated maps are furthermore rendered for demonstrative purposes. Our results fit the desired structure defined in the input sketches and exhibit a lot of variety in the biomes. The scenes demonstrate great quality by illustrating clean and usable terrain which concludes that GANs are very applicable to this purpose.

Keywords Generative Adversarial Networks, procedural content generation, game design, terrain generation, deep learning.

Declaration

By my own signature I declare that I produced this work as the sole author, working independently, and that I did not use any sources and aids other than those referenced in the text. All passages borrowed from external sources, verbatim or by content, are explicitly identified as such.

Signature:  Date: 10.01.2022

Acknowledgements

First of all, I would like to thank my advisor Stephanie Autherith for encouragement and guidance at all stages of this thesis as well as for the chance to write about such a creative topic. The input and ideas really provided lots of clarity and the work, despite of course having its rough phases, was always done in an easy-going atmosphere.

Special thanks also goes out to all the people who decided not to believe in vaccinating themselves during this pandemic and them being the reason why we haven't gotten to a state where this virus is under control, which resulted in this thesis having been worked on mostly in lockdowns and quarantines.

Lastly and again less sarcastically, I would like to thank my closest friends not only for their emotional support but also for all their Spotify playlists I listened to for countless hours during the implementation and writing process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Synthesising content has always been a substantial part of the development of the steadily growing entertainment sector of computer games. The most common approach, procedural content generation (PCG), has gained popularity over the last years. With PCG, large amounts of content can be automatically created which reduces the workload of game designers. One vital component is the play area. Most modern games feature some sort of map which lets players move around it linearly or explore more openly. The task of actually creating them can be very tedious depending on how diverse and detailed they should be. Often, game designers have to spend countless of hours establishing their world. Despite how detailed one might want to create their map, one of the most important steps of it will in many cases be the generation of the terrain.

Since in most cases it is in the interest of the developers to ensure players keep playing their computer game a certain level of replayability has to be guaranteed. Players should not get bored of the game and one aspect that could discourage them could be repetitive terrain while exploring. Terrain after all consists of different features that established an interesting scene. Different biomes which might have different characteristics as well as the elevation of the terrain plays a major role in that.

This thesis will explore the automated generation of low-poly maps by utilising Generative Adversarial Networks (GAN) via a Pix2Pix model. The designers should still be given sensible tools to condition their maps. With our approach, they are able to style both the structure as well as the theme of biomes to a certain degree. First of all, they will be able to determine which areas of the maps will be traversable. All a designer has to do is to merely sketch out the accessible regions in a map as well as choose from available themes by selecting different colours in their sketch. The trained networks will then generate a map that matches these characteristics. This distinction of non-traversable and traversable regions is particularly meaningful since it defines the most fundamental characteristic of terrain. For example, in a finished game, a player can walk through grass plains but might not be able to traverse steep cliffs.

Regarding the structure of this thesis, the creation of a dataset with which the training of the networks has been done will be explained first. Then, the utilisation and implementation of the GAN which generates the desired maps will be thoroughly discussed and evaluated. To showcase the final generated results, renderings will be made. Ultimately, the final results will be discussed and an outlook about further improvements to this approach will be made.

Other papers have already described the generation of terrain and maps procedurally, some even with the use of GANs. Our approach nonetheless features novel aspects considering we have explored how GANs can synthesis both components of a modern 3-dimensional map, the elevation and biome data, at once. This demonstrates that combining multiple PCG tasks whiting a single GAN is feasible.

Chapter 2

Related Work

This chapter will review several books and papers which are related to procedural creation of low-poly maps and GAN. Firstly, general applications of procedural content generation will be discussed in Section 2.1. A focus will be drawn to the generation of maps in 2.1.2. Furthermore, in Section 2.2 GAN will be discussed. There, a introduction of the used Image-to-Image approach will be made in particular. The aim is to provide an overview of the applied research and display where this thesis draws its inspiration from to clarify the methods portrayed in Section 4. A large part of this thesis is based on GAN research published in the last few years as this method has gained popularity and PCG applicability recently.

2.1 Procedural Content Generation

Numerous different techniques of PCG exit and the choice of the most suitable one depends solely on what content needs to be generated. *A Survey of Procedural Techniques for City Generation* [17] provides a broad overview of the different kinds of variants such as using Perlin noise, L-systems, tiling systems, fractals and algorithms based on achieving a cellular structure. Their main focus lies on the procedural generation of cities.

Another use of L-systems can for instance be seen in *A Constructive Approach for the Generation of Underwater Environments* [1] where the authors generate different kinds of coral. The use of cellular automata in the creation of infinite cave levels in real time was researched by *Johnson, Yannakakis, and Togelius* [16]. *Parberry* on the other hand utilises value noise, a variant of Perlin-noise, for creating infinite terrain using real-world elevation data [21]. Similarly, *Zhou et al.* use digital elevation models to create terrain by allowing users to sketch the areas in which features like mountain ranges should be present [37].

2.1.1 Video Games

Regarding PCG in computer games, notable applications are the widely popular games *Minecraft*[1], *No Man's Sky*[2] and *The Binding of Isaac*[3]. They have the common goal of providing endless variation in their game content that make sure it never feels repetitive. *Shaker, Togelius, and Nelson* provide the most extensive overview of PCG in video games in their book *Procedural Content Generation in Games* [30]. They approach the topic from several different points of view and give thorough explanations as well as real world examples

[1] Mojang, Minecraft, 2011
[2] Hello Games, No Man's Sky, 2016
[3] Edmund McMillen, Florian Himsl; The Binding of Isaac, 2011

by mentioning which and how popular video games utilise procedural generation. The book is divided in different types of procedural generation: search based approaches; constructive methods; fractals, noise and agents; grammars and L-systems; rules and mechanics; answer set programming and mixed-initiative content creation. And despite PCG being present in published games for three decades, research is far from exhausted.

Similarly, in their paper *Procedural content generation: goals, challenges and actionable steps*, *Togelius et al.* mention that the goal of procedural generation is to advance so far that complete games can be generated with just minimal parameters. They envision a system that can create multi-level multi-content procedurally. According to the authors, this still faces numerous challenges and it is no surprise that almost all PCG approaches focus on single types of content. The development of a video game is simply too complex to be totally automated. But they do provide a list of steps one has to follow to procedural create a video game [34].

2.1.2 Maps

Many tools for creating maps for all kinds of different purposes already exist. These are often times online world generators, be it for computer games like *Cities: Skylines*[4], terrain for simulations, or just fantasy maps[5]. In most cases, these maps do only create a map that is of one particular style, either with biomes or just with heights.

Considering that many different options on how to generate maps exist, it is essential to evaluate the use-case of the desired map. Factors to consider are the dimensions, level of detail and of course the application. *Snodgrass and Ontañón* for example utilise Markov Models to create maps for side-scroller video games like *Super Mario Bros* [31]. A different approach was used by *Liapis, Yannakakis, and Togelius* in their paper *Sentient World: Human-Based Procedural Cartography* [18]. Their system allows users to sketch a rough map using certain colours as biomes and outputs a more detailed version of this map. To achieve this, they train Artificial Neural Networks and also provide the user with a few example maps after each refining step. With each iteration, the user can select which maps fits their intention the best and then generate an even more detailed map that is ultimately used to create a matching height-map. The resulting two maps both include the elevation data as well as the biome data. Biome-maps are composed of a colour image with each pixel's colour representing one biome. Height-maps are grayscale images that represent the maximum elevation as white and the minimum elevation as black. Their work perfectly shows the intention of mixed-initiative design - to provide quality content with minimal effort from the user-perspective.

2.2 Generative Adversarial Networks

With the introduction of Generative Adversarial Networks, *Goodfellow et al.* presented a new method of utilising machine learning and neural networks. It enables them to synthesise computer generated content with the aim of being indistinguishable from real data. They work by making use of two separate neural networks. One being a generator which produces new data and the other being the discriminator, whose objective it is to distinguish the generated data from real data.

The generator network starts by feeding the discriminator random noise. It will then assign a probability $D(x)$ which corresponds to how likely the data x is real or not. The

[4]Colossal Order, Cities: Skylines, Paradox Interactive, 2015

[5]Azgaar's Fantasy Map Generator, https://azgaar.github.io/Fantasy-Map-Generator/ (Accessed: 29-11-2021)

discriminator is trained to maximise this probability while the generator is trained to minimise $log(1 - D(G(z)))$. In the beginning, the discriminator will have no trouble in determining that the random noise came from the generator and not from the actual training data. The generator will then try to deliver different data to the discriminator, hoping to trick it at some stage. They therefore play a minmax game with the value function displayed in 2.1 with D and G being the discriminator and the generator, p_{data} the distribution of the data and $p_{\mathbf{z}}$ the distribution of the noise.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\big[\log D(\mathbf{x})\big] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\big[\log\big(1 - D(G(\mathbf{z}))\big)\big] \tag{2.1}$$

This way, over time and with each weight update, the generator will learn which variables are the ones that lead to better results in trying to convince the discriminator that its synthesised content is genuine. When the networks have trained long enough, such that the discriminator is no longer able to accurately determine whether data is generated or not, the quality of the generator is sufficient and can be used to generate novel data [11].

It is undisputed that the use of artificial intelligence has greatly expanded the capabilities of data generation. *Gatys, Ecker, and Bethge* for example utilise Neural Networks to transfer styles of paintings [8]. More methods have been extensively surveyed in *Procedural Content Generation via Machine Learning (PCGML)*. The authors contrast existing methods of creating content, namely constructive methods and search based approaches, with using machine learning. For their creation of platformer levels, game maps and interactive fiction stories, they consider five categories: backpropagation, evolution, frequency counting, expectation maximization (EM) and matrix factorization [32].

The use of GANs for the creation of video game content, more precisely for the video game DOOM, was researched in *DOOM Level Generation using Generative Adversarial Networks*, where the authors used Wasserstein GAN with Gradient Penalty as introduced in *Improved Training of Wasserstein GANs* [12, 10].

2.2.1 cGAN

Considering the traditional GAN lacks in means of controlling the generated output, Mirza and Osindero introduced conditional GANs. By extending the generator and discriminator with an extra label $\boldsymbol{y}$ during the training step, the system can be conditioned on it. This way, unseen or user generated labels can be given as an input to the generator. A trained generator will then be able to create data that corresponds to this input. This can be seen graphically in 2.1 as well as in the Equation 2.2 which defines the minmax game of cGANs [20].

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\big[\log D(x|\mathbf{y})\big] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\big[\log\big(1 - D(G(z|\mathbf{y}))\big)\big] \tag{2.2}$$

The authors of the cGAN paper, like *Goodfellow et al.* in theirs [11], present their methods on the *MNIST* dataset[6] which consists of handwritten digits. *Mirza and Osindero* were able to condition their generator to then enable the generation of new handwritten digits correlating to a number as input label. They also tested their method on annotated images from *Flickr*[7] and were able to generate new tags and annotations to before unseen images [20].

[6] Yann LeCun and Corinna Cortes and Christopher J.C. Burges, THE MNIST DATABASE of handwritten digits, http://yann.lecun.com/exdb/mnist/ (Accessed: 23-11-2021)

[7] https://www.flickr.com/ (Accessed: 01-01-2022)

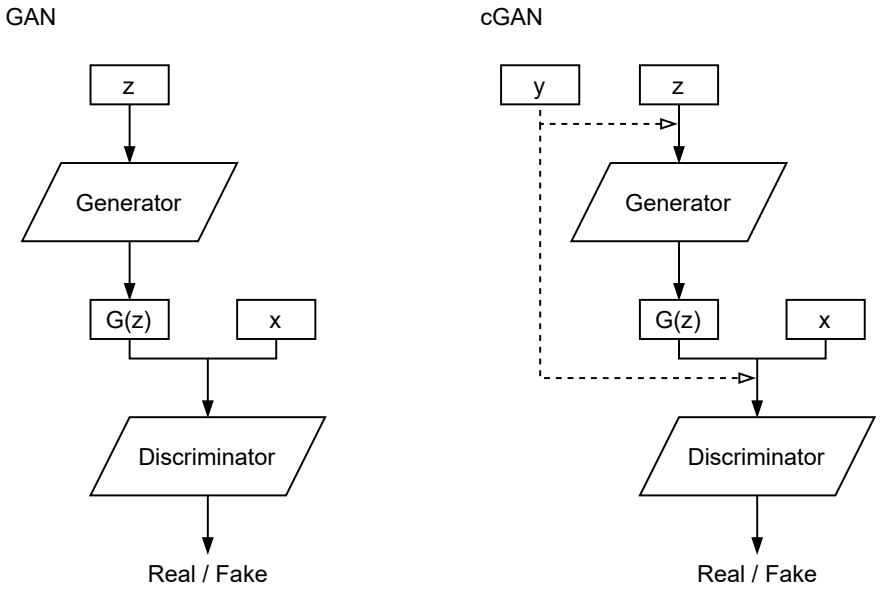

Figure 2.1: Structure of standard GAN with just the noise as input for the generator on the left. The conditional GAN on the other hand receives both the label y as input for the generator as well as for the discriminator as seen on the right.

This control over GANs gained a lot of attention when *NVIDIA*[8] announced its GauGAN that enables users to create completely synthetic images by drawing a rough sketch of the elements in the desired image [22]. They even further improved their results when they released GauGAN2[9] in November 2021. The new model even allows users to simply describe a scene with a sentence or phrase, creating images from just a few words [28].

2.2.2 Pix2Pix

A special kind of conditioning was introduced by *Isola et al.* and their paper *Image-to-Image Translation with Conditional Adversarial Networks*. The previous approach of cGANs applied conditioning to discrete labels and tags. Hence, they developed a method that uses pictures as input by changing the network architecture to utilise a U-Net [15].

First applications of this system on maps have been by *Beckham and Pal* in *A step towards procedural terrain generation with GANs* [2] where they generated texured maps that used height-maps as input. As training data the NASA Visible Earth project was used which maps provides a satellite as well as height-map of the Earth. Their approach has been further improved by *Mattull* in *Toward Improving Procedural Terrain Generation With GANs* [19].

A similar technique was presented in *Procedural 3D Terrain Generation using Generative Adversarial Networks*. The authors *Panagiotou and Charou* first procedurally generated satellite images and then used Pix2Pix to generated corresponding height-maps.

The method *Ping and Dingli* used in their paper *Conditional Convolutional Generative Adversarial Networks Based Interactive Procedural Game Map Generation* focuses more on the video game utilisation of generated maps [26]. They generate more complex maps with

[8] https://www.nvidia.com/ (Accessed: 14-12-2021)
[9] gauGAN2, http://gaugan.org/gaugan2/ (Accessed: 24-11-2021)

a similar approach to the previously mentioned method by [18] which uses sketches as an input. But in contrast to their biome-based application, in [26], the user merely sketches the traversable game-play area with the output map corresponding to it. Their resulted label map is only generating the two-dimensional map, but can through a theme renderer be presented and used in three dimensions as well. Further details on Pix2Pix and our concrete implementation of it will be described in Section 4.2.

Chapter 3

Dataset

As for all machine learning tasks that rely on training data, a dataset that represents the content and the to be generated output is essential. These datasets have to be of a substantial size to handle outliers and enable the networks to find patterns to train on. The task of finding a large dataset that include both elevation data as well as biomes seemed more simple than it ultimately was. Attempts of utilising online-tools that generate maps transpired to be not very fitting. In most cases only either biome-maps or height-maps were generated. Therefore, the decision to create a new dataset, was made. The use of an existing video games level data, similarly to how it has already been done with the game DOOM which has its level information stored in WAD files, was also cosidered [10]. Since we intended to use GANs that work with images, this idea was abandoned. The focus was also laid on terrain generation and a simple height-map together with a biome-map seemed more fitting. To give the system a first try, a small dataset with only height-maps was initially created. For this, real world elevation data was used to generate these height-maps. Originally intended for the video game *Cities: Skylines*, the generated elevation maps from *terrain.party*[1] and a dedicated *Cities: Skylines map generator*[2] fitted well for this purpose.

When it was established that the system is working successfully with just height-maps, the search for an easy way to generate biome-maps that fit to height-maps was started. The biomes of terrain have to correspond to the terrain. It is not sensible for example to have a mountain biome at sea level. It therefore did not work to simply overlay Perlin noise as the elevations on randomly generated biomes. To achieve a meaningful distribution not only the elevation of the terrain but also climate has to be taken into account. The authors of *AutoBiomes: procedural generation of multi-biome landscapes* for example create a small climate simulation that takes temperature, wind and moisture and precipitation into consideration to aid the system in assigning their biomes [7]. A similar approach has been taken by *Thomas Würstle* who additionally also uses a global circulation with linear humidity as well as a pressure based circulation to generate biomes [35]. Our results should of course represent those characteristics of logical biomes and heights as well. Therefore, a dataset that features those attributes was crucial to us.

Ultimately, a project by the *GitHub* user *pecarprimoz* was used for the generation of the biome and height-maps [24]. Using their *Unity*[3] project, it was possible to generate a large number of maps with lots of variety. The created maps are all 128 by 128 pixels large and can be exported as PNG files. The size of the map is a good compromise between allowing a fair

[1] https://terrain.party (Accessed: 21-11-2021)
[2] Cities: Skylines map generator, https://heightmap.skydark.pl/ (Accessed: 21-11-2021)
[3] Unity Real-Time Development Platform, https://unity.com/ (Accessed: 30-11-2021)

amount of detail but still being low-poly and easy and quick to work with for the networks of the GAN. It utilises the height, moisture and temperature data to create biomes that fit these characteristics, similar to the papers mentioned before [7, 35]. The following pages will describe how these maps were used to create the dataset. Additionally, Section 3.2 will explain what modifications were performed to them to achieve better results.

	R	G	B	Colour
Water	35	52	255	
Sand	156	255	153	
Steppe	200	176	138	
Grass	120	80	0	
Forest	61	94	0	
Mountain	182	233	14	
Peak	98	104	118	

Table 3.1: Biome-colours and their RGB values

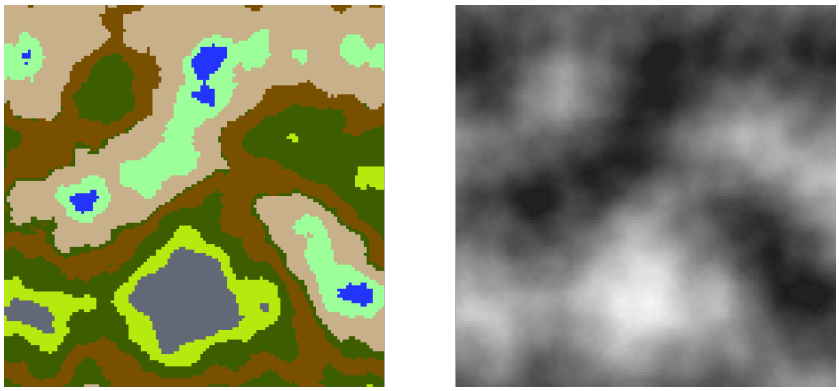

Figure 3.1: 128x128 pixel biome and height-map created in *Unity* [24]

As seen in Figure 3.1, the biome-map is composed of seven different colours. Each colour corresponds to one biome which are listed in Table 3.1. The height-maps are corresponding to each tile's elevation, ranging from lowest as value 0 to highest at 255.

Since the aim of this thesis was to provide certain freedom in stylisation from a user, this already has to be accounted for when creating data for the system to run with. Similar to the original Pix2Pix paper [15], where an image serves as the condition, a sketch drawn by a user will be the input in our case.

3.1 Traversability-Map

As already stated in Chapter 1, the input of the system should be a user created sketch of an area-of-play which will then correspond to the traversable areas of the generated maps. This sketch will serve as the *traversibility-map* of the to be generated maps.

In the example shown in Figure 3.2, the traversable regions are coloured in white. These white areas are the area of play where a possible future player of a game that utilises the final generated maps should be able to move their character around in. The areas in black should be inaccessible to a player. This will be discussed again in more detail in Section 4.6.

Figure 3.2: 128x128 pixel example input traversability-map

This stylisation subsequently needs the training data to also feature labels that describe a possible area of play, according to the traversable regions of a map. As briefly mentioned in Section 1, a players movement is restricted by the biomes *Mountains*, *Peak* and *Water*, as well as by the steepness of the terrain. It is not possible for a player to climb cliffs that are too steep. To create these traversability-maps for the maps of Figure 3.1, a short algorithm was developed that will mask the characteristics and create a similar looking map to the ones desired as input. To calculate the gradient of pixels adjacent to each other, the sobel filter, Equation 3.1, was used.

$$S(i,j) = \sqrt{dx(i,j)^2 + dy(i,j)^2} \tag{3.1}$$

The order of difference in the directions x and y is calculated and then coloured in if the current's surrounding pixels are too steep. Further adjustments and filters are applied to ensure that the white areas are smoothed and filled in. The edges of the steep areas can be imperfect since the input sketch the user draws should also just represent the area of mountains, water and steep cliffs roughly. The two maps that ultimately create the traversability-map can be seen in Figure 3.3.

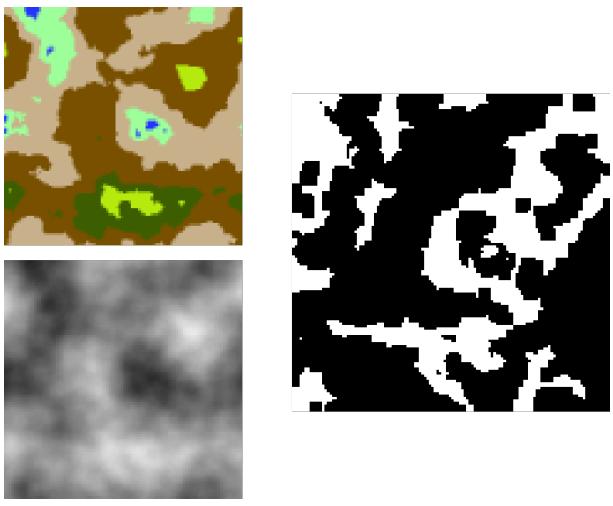

Figure 3.3: 128x128 pixel traversability-map generated from the left biome and height-map

This already functions as stylisation in terms of controlling the functionality of the map. A different way of conditioning would be to manipulate the actual look of it. The ways this can be done will be explained in more detail in Section 4.5 where the final modifications to the dataset, in order to enable more styling, was done.

3.2 Artifacts and Modifications

A few further adjustments were made to the biome-maps in order to maximise the quality of the system. Firstly, the *Unity*-generated height-maps add water just through a plane that intersects the terrain. This leads to the elevation map to actually have depth in its *Water* biome. This does make lakes in theory more realistic, but leads to complications for the generated model where in some cases the surface of water looks uneven. To circumvent this, the *Water* biome in the height-maps were flattened to the water's surface height. So the modified data will have uniform heights in lakes and no more depth which does not really matter in our case because we consider water to be non-traversable either way. Further post-processing will be done to the generated water to make for an actually correctly looking terrain. These changes will be discussed further in Chapter 5.

Additionally, because the used Pix2Pix model just creates actual images, the model seemed to have troubles distinguishing between the *Water* and the *Peak* biome. The similar looking colours might make it harder to generate meaningful terrain. Therefore, the colour of the *Peak* biome was changed to white with the characteristic of being just the icy summits of mountains.

3.3 Concluded Dataset

The data with which the networks will be trained ultimately consists of our three 128 by 128 pixel maps next to each other. The label will be the traversability-map on the left. Next to it will be the biome-map and the height-map like seen in Figure 3.4. For the training, a dataset handler will map the images to an input image and the target image. The two images in the center and right are stacked on top of each other. The label is consequently 3 channels, one for each of the RGB colours. The actual data will consist of 4 channels, 3 for the RGB colours of the biomes and one grayscale. The colours of the maps in the final dataset were expanded by modifications to the traversability as well as the biome-maps which is discussed in Section 4.5.

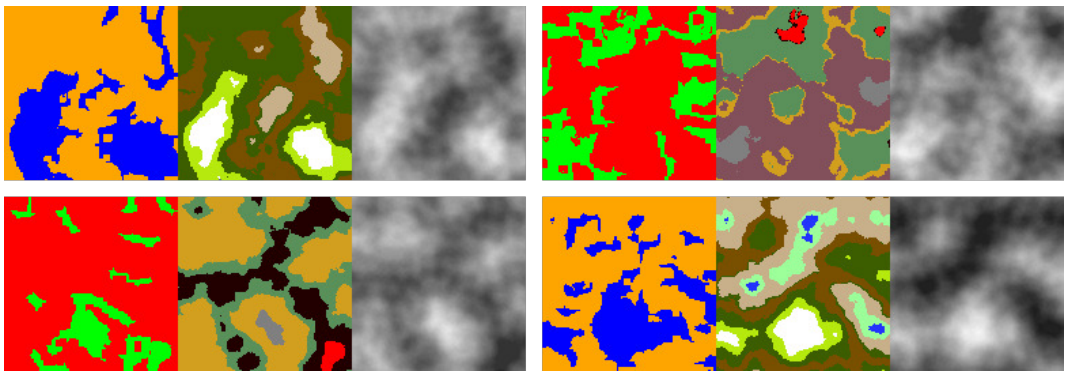

Figure 3.4: Four 384x128 pixel examples of a traversability-map, biome-map and height-map (from left to right)

The entire dataset consists of 4832 samples, each with a traversability-map, a biome-map and a height-map. Of all those samples, 4096 have been dedicated for training the network. A further 722 images have been used for testing it and 16 images for the evaluation. The entire size of the dataset is 143.1 megabytes.

Chapter 4

Methods

The following chapter will describe the methodology of this thesis and the procedure with which we structured the development. A detailed overview of the structure will be given in Section 4.1. Section 4.2 will describe the Generative Adversarial Network that we based our approach on, already mentioned briefly in Chapter 2. There, the specifics of the network architecture as well as the loss function it uses, will be discussed. Following this, the Section 4.3 will describe the initial use of a single input image to a single output image translation, while Section 4.4 and 4.5 will focus on the conditioning and ultimately translating one input image to the two maps we want to generate. Some more details on the input sketch and how different variations of it will be used for the evaluation of the system will be described in Section 4.6. Lastly, the rendering step which concluded our project and neatly displays the generated results will be described in 4.7.

4.1 Overview

The three different steps of our project can be seen in Figure 4.1. The process of creating a dataset was already described in Chapter 3. The first step in the actual pipeline is for users to create an input sketch, serving as a rough traversability-map.

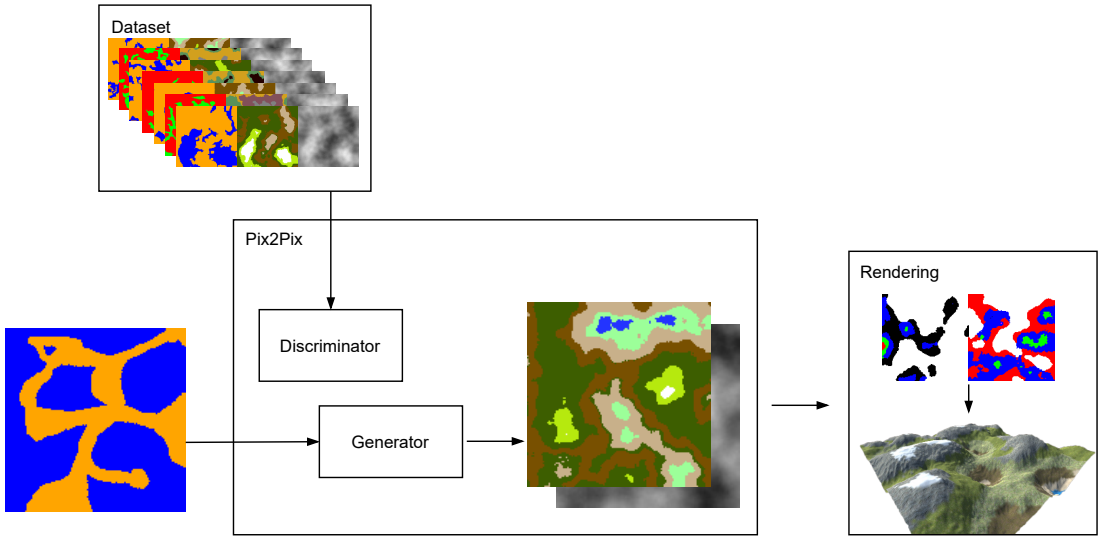

Figure 4.1: Structure of the designed system

This sketch will then be given to our GAN as the conditional label and serves as the styling of our desired output. The GAN has been trained with the maps from the dataset and will then generate new biome and height-maps. After some short post-processing these generated maps can subsequently be rendered in *Unity*. This will result in our final output, a 3-dimensional scene.

4.2 Generative Adversarial Network

Considering our main goal of this thesis, to generate biome and height-maps efficiently, the use of machine learning seemed applicable. As made clear in [32], the approach one might want to take depends greatly on the use-case of the system and the desired outcome. Our task of creating new maps based on unseen input sketches fits perfectly with the main purpose of Generative Adversarial Networks.

For our use case, we decided to not rely on numerical data distribution like the traditional GAN or on the binary or simple numerical labels the cGAN implemented. We wanted to create low-poly scenes with terrain that is of satisfactory quality by it's stylistic components and not necessarily by how large and detailed it is. Therefore, we had actual maps as images in mind. That is why we came to the conclusion to use the Pix2Pix Generative Adversarial Network, in particular *Aladdin Perrson*'s *PyTorch*[1] implementation [25] as the basis of our model.

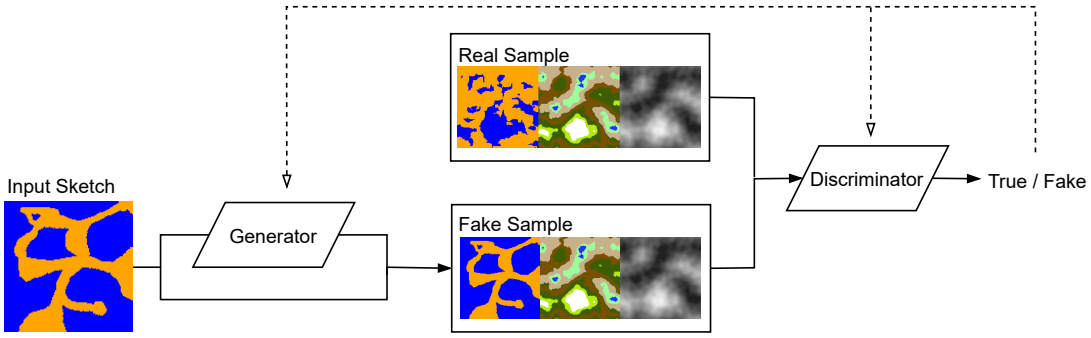

Figure 4.2: Overview of the conditional training

Like shown in Figure 4.2, the training of the GAN works by showing the discriminator both real samples from the dataset on the left as well as the generated fake samples that the generator outputs on the right. The discriminator this way either takes label Y and the real sample X in the shape of (Y,X) or once again the label Y but this time with the generated sample G(Y) in shape (Y,G(Y)). Y is in this case corresponding to the traversability-maps that have been created for the dataset or for the input sketch serving as the traversability-map. The handling of the data from out dataset, which contains 384 by 128 pixel images, was done by mapping the three images to their corresponding 128 by 128 image. The left image will just serve as the input image. For the target image, the right image was first transformed to grayscale since it was read as an RGB image from the dataset handler. It is then stacked on the center image. The resulting 128 by 128 pixel 4-channel image will be the target image. The generator trains with the 128 by 128 pixel input image and the discriminator with both 128 by 128 images.

[1]PyTorch, Facebook's AI Research lab (FAIR), 2016, https://pytorch.org/ (Accessed: 18-12-2021)

In order to synthesised low-poly scenes, the maps for our purpose do not have to be the of the largest scale. We chose a map size of 128 by 128 pixels. The maps still have enough detail and variation in terms of biomes while not being unnecessarily large, resulting in faster training. The image size in turn has to conform to the structure of the GAN's generator network, which has to be altered in contrast to the original Pix2Pix paper which uses differently sized images. This will be explained in detail in Section 4.2.2 where the generator model's U-Net network structure is discussed.

For the discriminator model, the implementation of such a network was not necessary because its task is mainly to distinguish the real samples from the ones a generator creates. Similarly, because our network should produce both a biome-map as a 3-channel image (RGB) as well as a grayscale height-map, we had to define the channels of the generated images to be 4. Further hyperparameters that were changed through evaluating the losses as well as the results were the learning rate and the batch size. The decision on both of these will will be discussed thoroughly in Section 5. The final model was trained with a batch size of 4 and the discriminator's learning rate was lowered to 2×10^{-5} while the generator's remained at 2×10^{-4}.

4.2.1 Loss Function

In the Pix2Pix approach, *Isola et al.* define the traditional conditional adversarial loss function as seen in Equation 4.1. $D(x, y)$ being the estimate of the discriminator D, that real data is actually real; $G(x, z)$ the output of the generator; and $D(x, G(x, z)$ the estimate of D, that fake data is actually real. $\mathbb{E}_{x,y}$ is the expected value over the real data and $\mathbb{E}_{x,z}$ is the expected value over the fake data. The formula is derived from the cross-entropy which quantifies the difference between two probability distributions, in our case the real and generated distributions (G, D).

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))] \tag{4.1}$$

They also determined that incorporating an additional loss function into their system, the model yields results that greatly improve their performance. They explain that the *Least Absolute Deviations (LAD)*, also known as L1 Loss, boosts their results by ultimately decreasing the blurriness over the L2 Loss (Least Square Errors function) which was for example used by *Pathak et al.* in their attempt to inpaint gaps of images according to their surroundings [23]. In the L1 loss function, minimising the error is achieved by minimising the sum between absolute difference of the real and the generated values as shown in Equation 4.2 [15].

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}\left[\|y - G(x, z)\|_1\right] \tag{4.2}$$

Therefore when it comes to the generator model, in contrast to only trying to outplay the discriminator like in the original cGAN, the network can be focused on generating images that come closer to ground truth by not only applying the traditional GAN loss to the generator but actually combining it with this L1 loss [15]. The final loss function is shown in Equation 4.3.

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{4.3}$$

For the calculation of the discriminator model's loss, a Binary Cross Entropy (BCE), specifically PyTorch's `torch.nn.BCEWITHLOGITSLOSS`[2] was used. It combines a Sigmoid layer and the BCE loss and is more stable than using Sigmoid followed by BCE. For the discriminator, this loss was calculated for both the real samples as well as generated fake ones.

The generator on the other hand obviously only calculates the BCE loss of generated samples. But as shown in Equation 4.3, it has to calculate the L1 loss too. The Lambda parameter for the L1 loss of 100 deemed as being the most sensible. It remains therefore unchanged from what the authors of the original Pix2Pix paper used. The two losses were then added together to account for the loss of the generator model.

4.2.2 Network Architecture

Regarding the network of the generator, the U-Net architecture was used. In contrast to original encoder-decoder networks, at the upsampling step, the U-Net concatenates the previous upsampled image and the corresponding encoded input sketch. This helps with containing detail and positional information that might be lost by simply downsampling and upsampling again [27]. Regarding the features of each layer, after the initial downsampling step which increases the features to 32, they are doubled each time until a maximum of 256 after 4 downsamples. Likewise, on the decoding path they are reduced until the final step where we will receive a total of 4 channels again as our output.

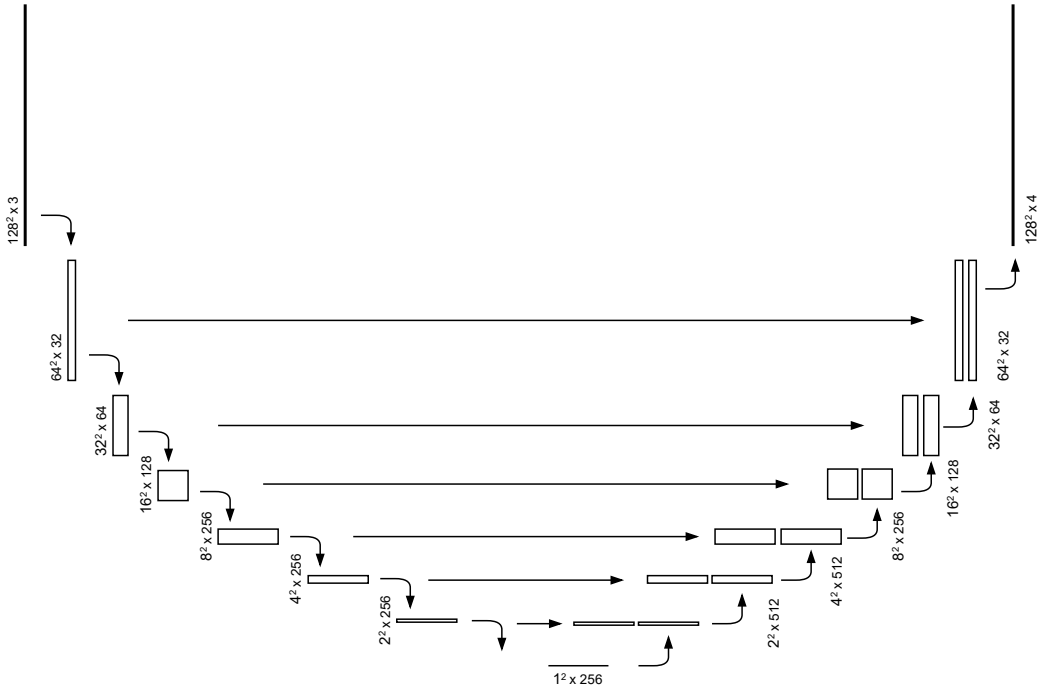

Figure 4.3: U-Net structure of the network

Similarly to the discriminator, which uses a network block consisting of convolution, then BatchNorm, then Relu - the generator does so for learning to create the output as well. At each step, firstly a convolution is applied. This will be either a down sampling or a transpose

[2]BCEWithLogitsLoss, https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html (Accessed: 25-11-2021)

convolution for the upsampling steps. The kernel-size for each sampling is 4 with a stride of 2 and a padding of 1.

Lastly, a ReLu will be used as the activation function. For the encoder convolutions, the down steps on the left side of Figure 4.3, leaky-ReLu is used. The images are decreased in their dimensions. A 128 by 128 pixel image will ultimately only be a vector before the upsampling starts again. The bottleneck then only consists of the *useful* information. In each of the upsampling steps the feature size doubles. This is because the image from a lower layer gets concatenated with the previously downsampled image of the same layer. It will then actually be quartered until the next up-layer by applying the convolution and upsampling. In total we only have six down steps plus the initial and final samplings. This corresponds to the map sized of 128 by 128 pixels we used. If one were to use a different map size the amount of layers might change.

Apart from the bottleneck step as well as the initial down and up step, Batch Normalisation is applied. Using it, the input for each of the layers of deep neural networks can be normalised which stabilises learning as well as increases the speed of the training [14].

4.3 Biomes2Height

Before the conditioning was implemented, the system was tested with synthesising height-maps from biome-maps. A similar thing was already done by the authors of [18]. In their approach, they trained multiple neural networks to then ultimately generate height-maps. They also utilised iterative refining where users can choose their result from a number of presented samples.

By using our Pix2Pix network we have shown that GANs are also very suitable for creating height-maps just with the input of biome-maps. In this case we had to select different parameters. For example, the input channel size was set to 3 to enable an RGB image while the output has to be set to a single channel grayscale image. Maps that could have been generated by other means could therefore be expanded. If a user for example just has a biome-map but desires elevation that fits their map, Pix2Pix seems like a applicable method.

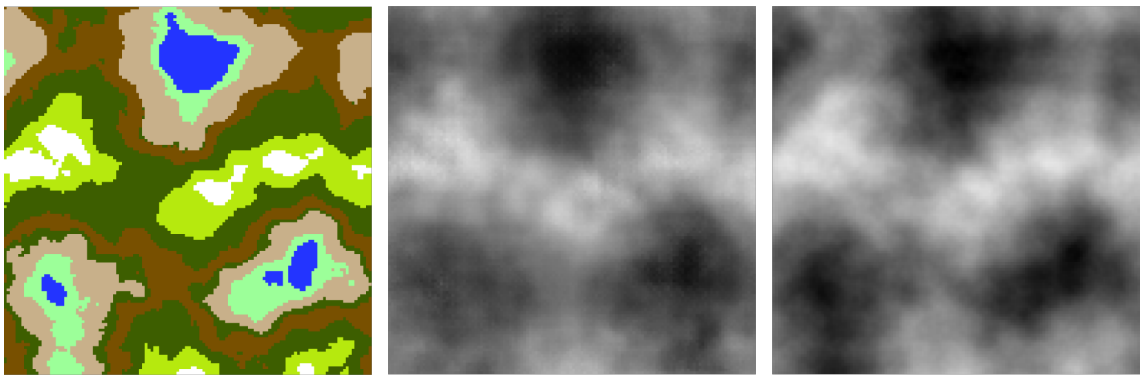

Figure 4.4: Example of Biomes2Height: original biome-map, generated height-map (250 epochs), ground truth height-map (left to right)

A simple translation of just a biome-map to a height-map can be seen in Figure 4.4. When compared to the ground-truth, the generated height-map shows very clearly the details we expected to see: a mountain range of higher elevated terrain that stretches from left to right through the middle of the image as well as lower elevated regions where the lakes are.

4.4 Stylistic Conditioning

Creating height-maps to corresponding biome-maps already raised the possibility of stylistic conditioning for our project. It enables users to sketch their own biome-maps that could then have their height-maps generated for them. But sketching an entire map with meaningful biomes already puts a lot of workload to the designers and would defeat the purpose of easy and good looking map generation that can be used by users with little know-how of game and level design. Consequently, we selected a different method, where users only have the very basic task of drawing the area-of-play. How this is done was already presented in Section 4.6 and the following chapter will explain how these input-sketches were used to condition maps.

4.4.1 Sketch2Biomes and Sketch2Heights

When the network was originally trained with having just a standard 1 map-to-1 map translation, the training data was either just traversability maps with only the biomes or only the heights. This was explored to see whether the idea of a traversability-sketch would even work properly. The processed inputs and the corresponding outputs can be seen in Figure 4.5 and Figure 4.6. The training data for these images were the same maps that have been included in the definitive version of the dataset, only in a different form. For these maps, the network did not train with a 4 channel image of biomes *and* heights but only with either as either 3 channels or 1 channel in the height-map scenario. The training data images therefore only featured the traversability-map and the height or biome-map (256 by 128 pixels).

As seen on the left side of Figure 4.5 and 4.6, only a rough sketch without a lot of detail was given as input to the GAN. Once again, the results were very promising and showed the

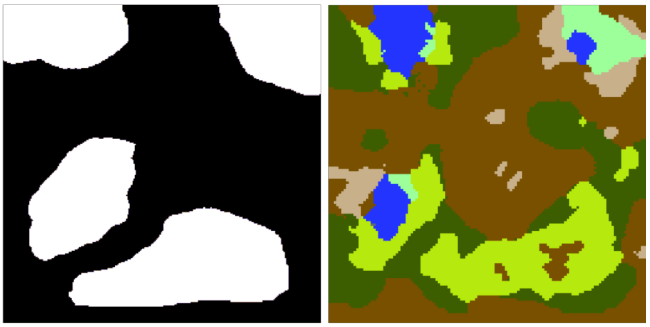

Figure 4.5: Example of Sketch2Biomes: input sketch (left) and output biome-map (right)

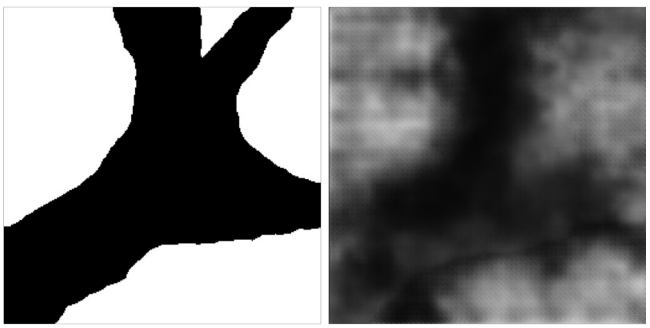

Figure 4.6: Example of Sketch2Heights: input sketch (left) and output height-map (right)

potential of using the sketch of a traversability-map as input. The generated biomes of Figure 4.5 clearly show non-traversable biomes like lakes, mountains and beaches in the white parts of the sketch. Furthermore, the maps show a lot of detailing in the traversable biomes as well. This results in a map that has variability with grass, steppe and forest biomes and is not only monotonous.

Similar results can be seen in the generation of the height-map in Figure 4.6. Once again, only a simple sketch was created to show the potential of how low detailed the user's inputs have to be. If only a traversable region that spans from the bottom left to the bottom right is wished, the sketch could look similar to this one. Here the network, as expected, matched traversable regions to being a similar height. The regions which were coloured as non-traversable are higher elevated and also show a lot of variation in height which corresponds to terrain with lots of steeper cliffs.

A more detailed discussion of these results will be done in Chapter 5 where the network was trained on creating both a height and a biome-map at the same time. Nevertheless, these results as well as the one of Section 4.3 already showed the potential of working with sketches as inputs and provided a very promising outlook for the end result.

4.5 Further Styling

The styling of controlling where certain features should be present in the map would be achieved by these traversability sketches. Despite this, we wanted to widen the possibilities of this project and were curious, whether we could train it to also feature more artistic differences as well. A larger variety in themes is after all very important to ensure an interesting user experience.

One possibility would be to just apply different kind of textures in the rendering step of the terrain generation. These textures could be stored as *Texture Packs*, similarly to the way the video game *Minecraft* does so with its *Resource Packs*. This ensures high flexibility in terms of style, but might also require the possibly tedious task of importing and handling of resource packages. It also does not change the actual biomes of the maps and would not allow the possibility for the biomes to have actual properties. It could merely texturise for example a lake that had originally water textures, with lava. If it was then wished for this lava lake to have properties such as hurting or burning a player when they got close to it, that would furthermore have to be altered too. That is the reason why we explored the ability of choosing different types of worlds in the generation of the biomes already.

To not only apply different textures, but actually change the functionality of the biomes and biome-maps, the training model has to be altered too. This enables it to define the style and entire setting of the generated maps at the input step. In our case, the user can not only determine the area of play but also choose between whether the biomes of a realistic overworld or the ones from an hell-like underworld should be generated. This underworld will have seven completely different biomes from the original ones, listed in Table 4.1.

The Pix2Pix GAN would allow for the labelling of whether the maps are of type underworld or overworld to be made by a single label value (e.g. 0 for underworld and 1 for overworld). This way was intentionally not selected, because it would restrict the generation of new maps to be either of type underworld **or** of type overworld. Since it is very interesting to evaluate whether the network could mix two different styles of maps together, a different approach was chosen. Through labelling the actual input sketches by choosing different colours schemes in the drawing, the two versions could theoretically be mixed.

	R	G	B	Colour
Lava	255	0	0	
Obisidian	35	0	0	
Swamp	90	145	90	
Desert	210	160	30	
Wasteland	130	80	90	
Rocks	128	128	128	
Ice	135	170	210	

Table 4.1: Underworld biome-colours and their RGB values

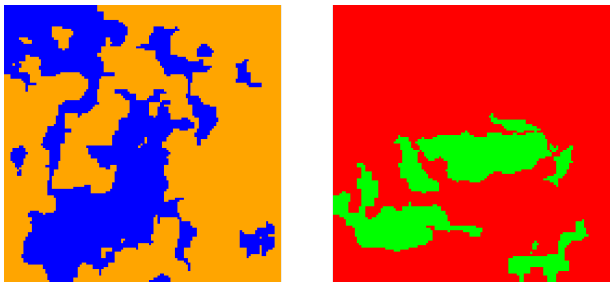

Figure 4.7: 128x128 pixel input traversability-map overworld (left) and underworld (right)

The model will be trained with traversibility maps in the colour schemes presented in the examples of 4.7. Consequently, a user just has to colourise their sketch in the underworld-colours instead of choosing the blue and orange overworld-colours and an entirely different world will be created.

4.6 Input Sketch

The purpose of the traversability-map was already mentioned in Chapter 1 as well as described how it is created from our real biome and height-maps in Chapter 3. The map which ultimately serves as the input for the GAN corresponds to these traversability-maps. It serves as the basic idea of where in the map a user will be able to traverse. These boundaries will either be determined by untraversable biomes (water, lava, mountains, peaks) or by the steepness of terrain. A user of our system will only have to draw a rough sketch of where they want these areas to be. That way, the shape and purpose of a map can easily be determined. Similarly to *Ping et al.*, the sketch serves as a mask of accessible terrain [26].

These sketches can be either very low detailed and just coarsely show where traversable regions are like shown in Figure 4.8. Here the blue (overworld) and green (underworld) regions correspond to non-traversable regions of the maps. The size of the input sketch has to match the maps that want to be synthesised, in our case 128 by 128 pixels. To guarantee that the generator model works with the input sketch we had to ensure that the sketches are 24 bit images. The actual colours for theses sketches are displayed in Table 4.2.

This way, only a few seconds are needed to outline the characteristics of a map. For example, maps with multiple different small areas that are interconnected by pathways can be generated. Another example would be to describe a large open area in the centre of the map that is accessible from multiple directions. Creators can be as creative as they like with these maps and determine certain aspects of a possible game that can be played on this map

	R	G	B	Colour
Overworld - Traversable	255	165	0	
Overworld - Non-Traversable	0	0	255	
Underworld - Traversable	255	0	0	
Underworld - Non-Traversable	0	255	0	

Table 4.2: Input sketch colours and their RGB values

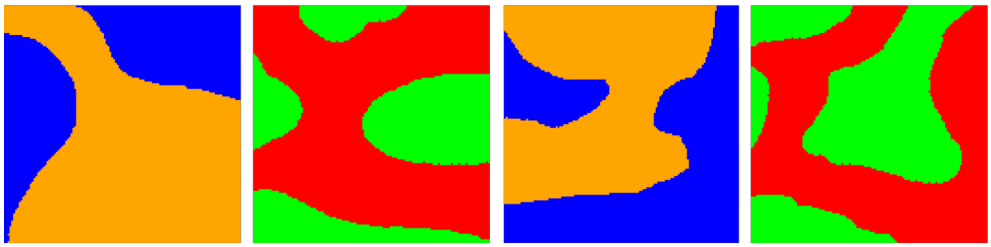

Figure 4.8: Examples of low-detail sketches

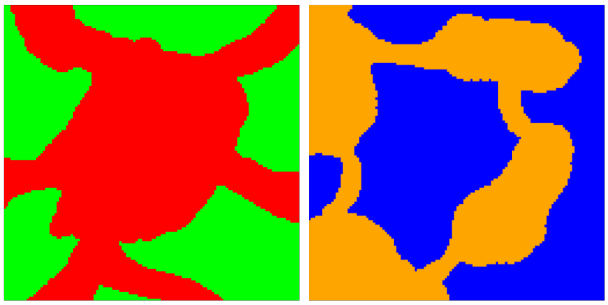

Figure 4.9: Examples of more detailed sketches

just by thinking about the feature of where users should be able to traverse in. Examples of maps with more thought-out characteristics are shown in Figure 4.9.

A user also has the ability to be even more precise with the definition of the traversable regions. An extensive evaluation of how quickly and with little detail these sketches can be drawn as well as how detailed they can be will be done in Chapter 5. There it will be also discussed, whether the two kinds of worlds can be used together in one sketch and intersected. This would enable even more possibilities in styling and investigates the selected methods further by combining the features the network was actually trained on separately.

4.7 Rendering

For the ultimate step of this project, the actual rendering of the generated scenes, the real-time development engine *Unity* was used. Directly generated height-maps did exhibit lots of noise which resulted in terrain being very bumpy. After a little experimentation, a simple Gaussian filter with a kernel size of 5 by 5 pixels smoothed the maps out enough to ensure they are usable. To load height-maps easily into the scene, *Game Surgeon*'s package[3] was used. With it, the 128 by 128 pixel height-maps can be loaded onto terrain. This elevation data will then be loaded onto terrain with a height of 32. Ultimately, the textures of the terrain

[3]HeightMapper unitypackage, https://www.youtube.com/watch?v=KiRq3SXhOc4 (Accessed: 4-12-2021)

had to be applied. These correspond to the generated biome-maps. Considering the output of the generator was an actual 3-channel-RGB image, a simple script had to be written that assigns the colours of the created image to the seven overworld or seven underworld biomes. Colour ranges of every biome's colour were defined and every pixel of the image got assigned an actual biome. Examples of this can be seen later on in Figure 5.2.

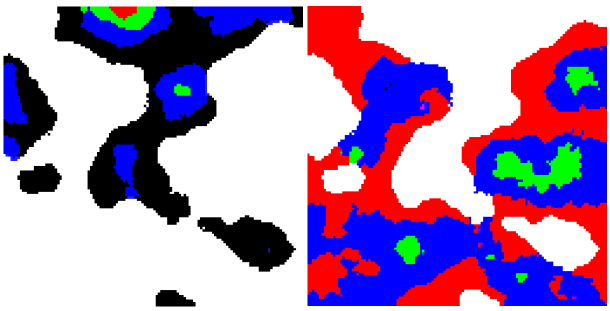

Figure 4.10: Examples of two splatmaps that are used to map textures onto terrain during the rendering step

Alpha-maps are commonly used when a combination of multiple textures is desired. With them, the weight of each of the textures can be independently defined. One example of them are *splatmaps*. With a splatmap, a single image can represent the intensity of multiple textures which should be *splatted* onto an object or terrain. Since *Unity* has the ability to texturise terrain by applying a splatmap, we also had to mask the created biomes onto two different RGBA splatmaps. This can be done manually in *Adobe Photoshop*. We decided to automate it with a script which iterates over every pixel of the generated biome-map and maps the biome to one of the 4 channels in the RGBA images. For our demonstrative purposes, the use of two splatmaps was sufficient. *Unity*'s *Terrain Toolbox*[4] allows to natively load two of these splatmaps onto a single terrain. The dimensions of the splatmaps will just be fitted onto the terrain. With them, all of our biomes can be assigned their unique texture, which makes the scene way more presentable. When both the elevation and the biomes are loaded onto the terrain, the resulting 3-dimensional scene can be explored. The results of these renderings will be displayed in the next chapter.

[4]Unity Import Splatmaps, https://docs.unity3d.com/Packages/com.unity.terrain-tools@2.0/manual/toolbox-import-splatmaps.html (Accessed: 2021-11-30)

Chapter 5

Discussion

In this chapter, the results of this project will be discussed and evaluated. Firstly, an overview over the generated maps will be given. A brief qualitative analysis on the generated and rendered scenes will be made as well as an analysis on how the scenes perform in terms of playability and variance. Section 5.2 will focus on evaluating the performance of the training as well as the final maps numerically and also discuss the usability of creating the maps and the process of receiving the final product. Additionally, a metric that compares the input-maps to created traversability-maps from the output, will also be introduced and measured.

5.1 Results

A few samples of generated maps have already been presented in section 4.4.1 and 4.3. In both of those cases, the end result was always either a biome or a height-map. Our final project will be able to create both of these at once. All of the steps of creating these maps were performed on a desktop computer with an Intel Core i7-10700F CPU at 2.9 GHz with an NVidia GeForce RTX 3060Ti graphics card used for the training of the GAN.

5.1.1 Overview

Users will be able to draw sketches of the maps they want to design and receive outputs that very well represent these characteristics. As shown in Figure 5.1, two different sketches have been drawn that vary a lot in design but ultimately are very easy to draw for everyone. Just by quickly thinking about which areas of the terrain should roughly be accessible, the final model will be able to generate quality results that will be structured according to the sketch.

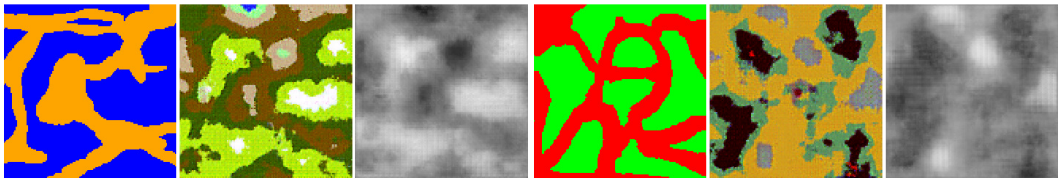

Figure 5.1: Raw overworld and underworld results from user drawn sketches

As can bee seen when looked at the output maps, the areas which should be traversable, orange and red respectively in the input sketches, have been realised by the system. One aspect that is interesting is that the overworld example seems to focus the inaccessible areas on highly elevated regions of the map. The *mountain* and *peak* biomes (light green and white)

as well as the brighter spots of the height-map clearly match the blue areas of the input sketch. In contrast, the underworld map seems to be more balanced in that it features both very low regions of the *obsidian* biome as well as higher *rocks* biome. Another difference is that the underworld result does not generate the most elevated *ice* biome as seen on the right examples of Figure 5.2. Although after 200 training epochs the maps already look very convincing, a post processing step of actually matching the colours to the biomes will still be necessary to then create splatmaps for the final renderings.

The results of the post processing of the biomes and heights can be seen in Figure 5.2. The new height-map has received some smoothing while the image of the biome-map had the actual values of the biomes assigned to them. These will then be used for a rendering of the entire scene which can be observed in Figure 5.3

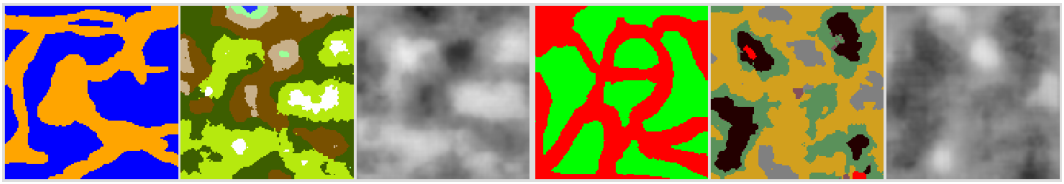

Figure 5.2: Post-processing applied to two examples

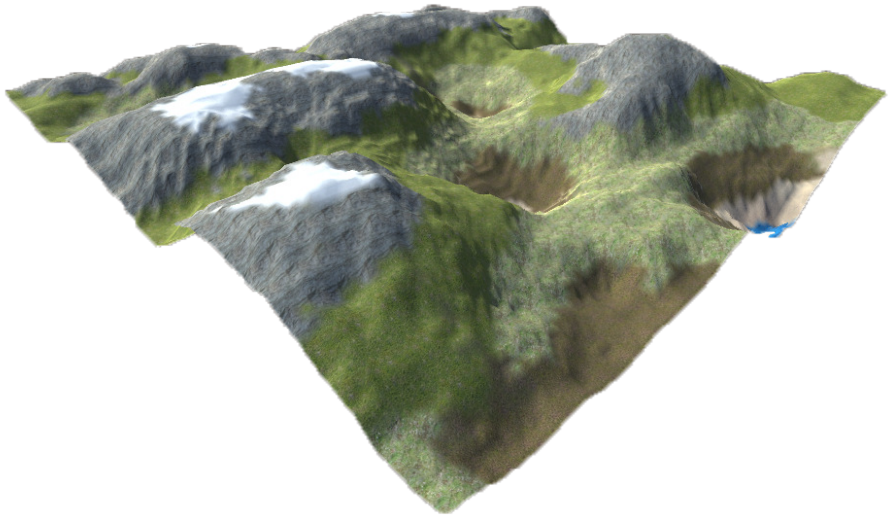

Figure 5.3: Rendered example of generated maps with a realistic looking overworld theme

These renderings, generated in *Unity*, mainly serve as demonstration purposes and will be explored some more in the next section where the actual features are discussed and how they could be used for actual graphic simulations or video games. As textures of this example, some free assets from the *Unity Asset Store*[1] have been used. They make it possible to render the originally rather low-poly dimensions of 128 by 128 pixels into a fairly realistic looking landscape.

When the texturing of the maps is done in this last rendering step, other textures can be used to achieve an entirely different setting. One example of a different set of textures being used can be seen in Figure 5.4.

[1]https://assetstore.unity.com (Accessed: 13-12-2021)

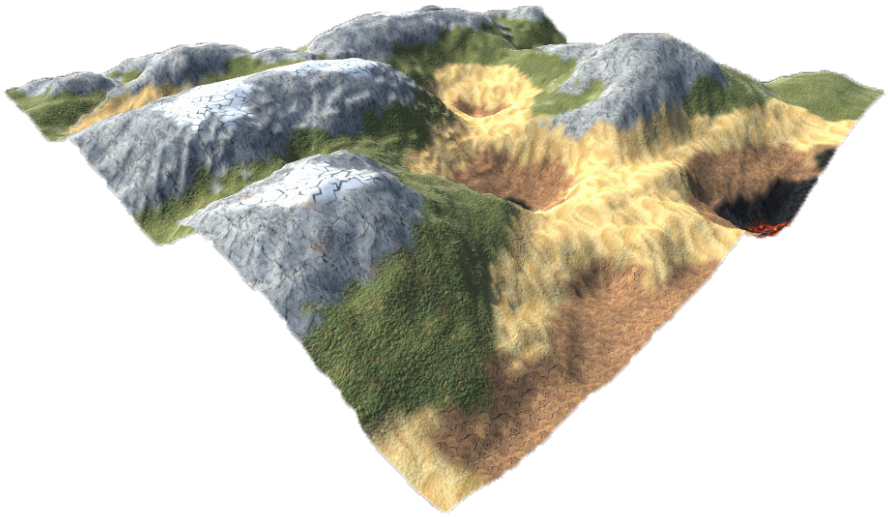

Figure 5.4: Rendered example of generated maps with a underworld theme

Here, the used textures represent the underworld theme more. It features different biomes like the *desert*, *wasteland* or *obsidian* biome. Purely the presence of darker biomes could effect players in comparison to a more realistic textured world. When looked at from a game-design perspective, properties like heat that could hurt players if they remain in the *desert* biome or less resources in a *wasteland* biome compared to the more lively biomes of the overworld could be implemented here too.

In general, colours play a vital role in how we perceive visual information [6]. This of course applies to video games too. Different colours can greatly affect which emotions are evoked. For example, a rather bright scenery and light colours will elicit more joy by players. This has been extensively researched in *How Color Properties Can Be Used to Elicit Emotions in Video Games* [9]. We also wanted to apply some of these characteristics to our generated maps.

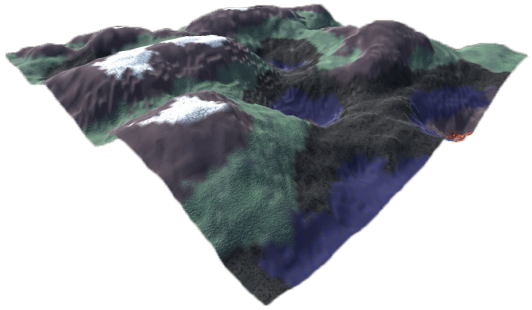

Figure 5.5: Moody scenery

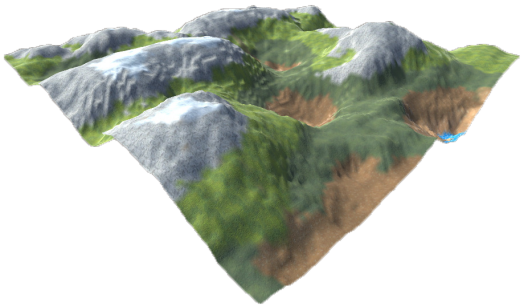

Figure 5.6: Uplifting scenery

Since we already had a underworld theme we figured that differentiating between cheerful and more moody scenery would be fitting. Figure 5.5 shows how the darker texures completely change the entire feeling. The terrain looks way more dangerous and otherworldly. Figure 5.6 on the other hand has a way more upbeat and lively view. This scenery does look less threatening. For both renderings, cartoony textures have been used which once again change

the entire setting of the maps and could be used for entirely different purposes. For example, if an entire video game is designed around a more low-poly look, it makes sense to not use the most realistic sketches too.

5.1.2 Quality

A comparison of how the final model generates maps after different epochs has been made in Figure 5.7. Here it is very well shown that the model is struggling quite a lot at the early stages, despite a rather low batch size of 8 having been used. It can also be seen, that the improvements of higher epochs do not change the quality of the results much.

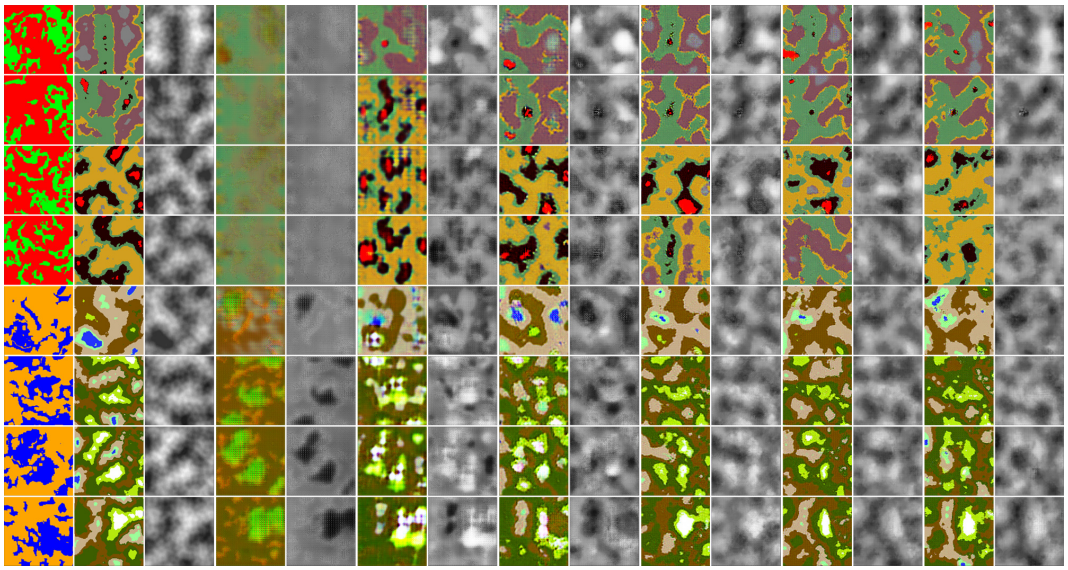

Figure 5.7: Raw example results from left to right: traversability-map & ground truth, 1 epoch, 5 epochs, 15 epochs, 50 epochs, 100 epochs, 200 epochs

While the output already shows a fairly decent structure after 15 epochs, the first useful results were received at around 50 epochs. What is also very noticeable, the results do not perfectly match the ground truth. While the input sketches do determine which areas of the map should be accessible or not, they do not retain information about how this area should be defined. The generated map might decide to place different terrain in certain spots than the ground truth which will both have the same characteristics of being traversable or not. What can be seen very well in this comparison, is that the model learns to match the elevations to the biomes very quickly. Even at the earlier epochs, it is immediately clear that the red lava lakes should be at a lower height than grey mountain ranges. Similarly, the white peak biome of the overworlds matches the position of the highest elevations in the height-maps.

After 200 epochs, the results are very presentable and almost indistinguishable from actual ground truth examples. This can also be observed when looked at the loss-values of the discriminator which at one point also has troubles differentiating between generated and real samples. This will be discussed thoroughly in Section 5.2.1.

When looking at the variety of the generated biomes, even after only a few epochs it is very clear that the model tries to create maps that, similarly to the training data, feature lots of different biomes. Ultimately, a very diverse landscape is generated.

This can also be observed when looking at the renderings of the maps. In Figure 5.8, the large variety in biomes as well as heights is clearly seen. This example features very high

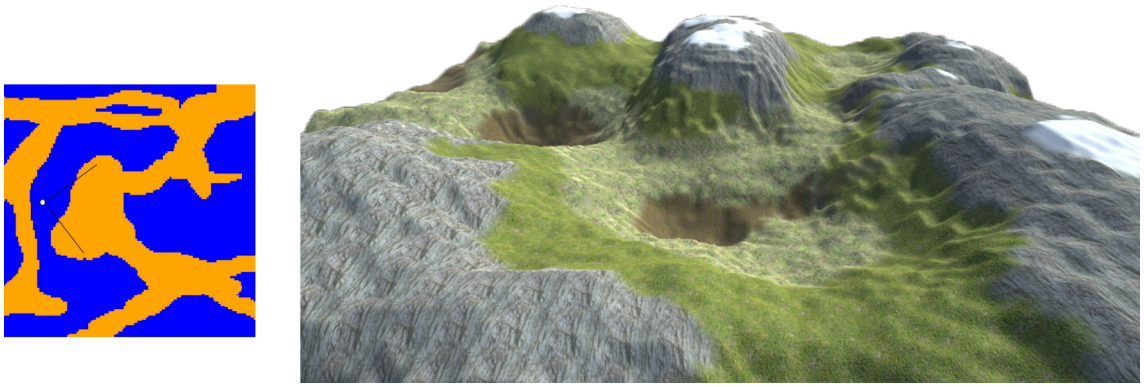

Figure 5.8: Rendering of one generated map (right) with the point of view indicated on the input sketch (left)

mountains with snow capped tops as well as deep valleys that still are not completely flat. Such a terrain enables lots of possibilities and 3-dimensional renderings like these showcase really well how our approach delivers usable results.

Additionally, the desired structure is also well visible in the renderings. Figure 5.9 shows one part of a map that, in it's input sketch, was marked as traversable pathway. The model generated a valley with steep cliffs on either side of it.

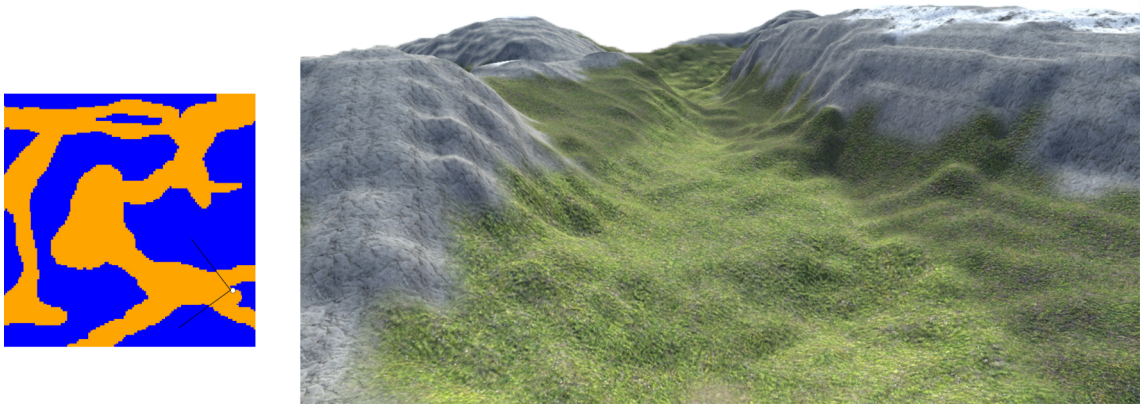

Figure 5.9: Rendered view down the generated valley (right image) that is located in the bottom right of the input sketch (left image). Non-traversable regions (blue) generated as steep cliffs and mountains to either side of the valley.

Thus, the playability of the generated maps is as desired. The cliffs are way too steep for players to be climbed and therefore enclose the traversable region in the middle. The mountain on the right side of Figure 5.9 also shows the snowy terrain of the *peak* biome. This biome was also assigned to be inpassible, possibly with the intention of being too cold for a player to traverse around.

5.1.3 Batch Size

One aspect the original Pix2Pix paper does not fully explain is, how the chosen batch sizes with which their training took place, impacted their results. In the end, the authors decided to use a batch size of 1, iterating in every epoch through the entire dataset [15]. Because it

was not obvious to us which batch size is the most suitable for this project we decided to train with different sizes and see how the training as well as the generated results will be impacted.

Different batch sizes determine how many images the network will train with at each training iteration. When a batch size of 1 is used, the network has to go through the entire training data in every epoch. When a larger batch size is used, for example 10, the network will receive *batches* of 10 images at once, resulting in having only one tenth the iterations at every epoch. The training hardware will of course have to deal with these increased batches and the memory the system uses will increase drastically.

We decided to evaluate batch sizes ranging from 64 to 4. A smaller batch size did not seem feasible in our case due to our large training data size. Going through the entire data of more than 4000 images at every iteration would result in more than 400000 training iterations. All the provided results in this section were trained with a discriminator's learning rate of 2×10^{-4} since these did show faster improvements for each of the batch sizes. The subject of learning rate and its impacts on the actual losses will be discussed in more detail in 5.2.1. As can be seen in Table 5.1, the time needed to train each of the networks decreases with a larger batch size. At the smallest batch size we trained, the training of 200 epochs took almost two and a half hours. This is substantially slower than only 43 minutes when a batch size of 64 was used.

Batch size	Time elapsed (200 epochs)	Iterations per epoch	Total iterations
64	43 min 26 sec	64	12800
32	55 min 12 sec	128	25600
16	1h 04 min 39 sec	256	51200
8	1h 32 min 02 sec	512	102400
4	2h 27 min 04 sec	1024	204800

Table 5.1: Stats on differing batch sizes

One of the most notable differences between large and small batch sizes, is that when smaller ones are used, the results already look decent after fewer epochs than compared to larger ones. When looked at Figure 5.10, the generated images still show lots of artefacts and little variation, whereas with a batch size of 8 in Figure 5.11, the maps look already way closer to an desired output.

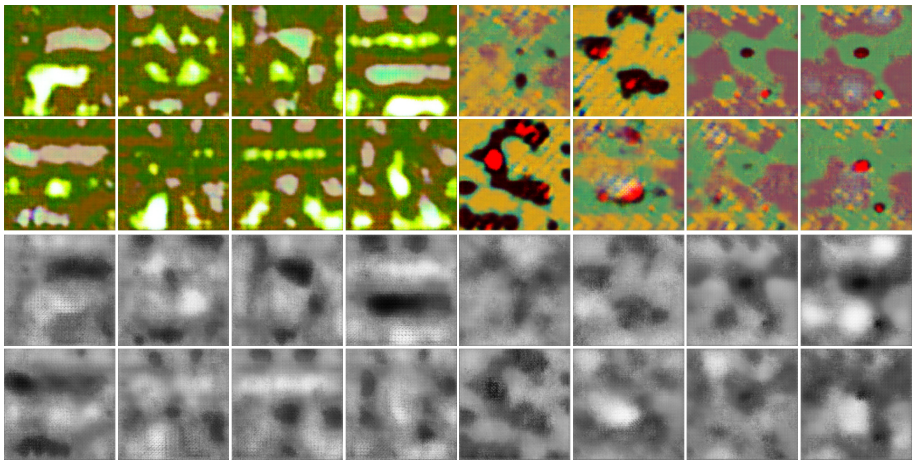

Figure 5.10: Raw generated maps with a batch size of 64 (after 25 epochs)

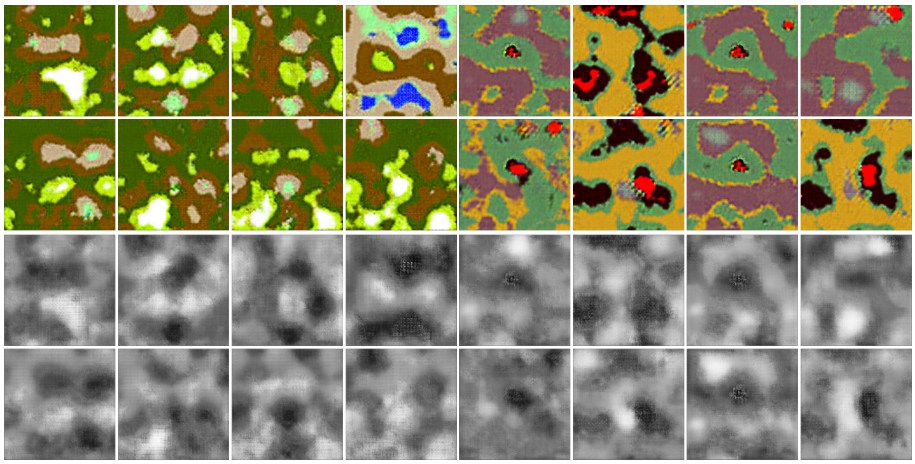

Figure 5.11: Raw generated maps with a batch size of 8 (after 25 epochs)

This can be observed too by looking at the losses of different batch sizes at different epoch times. The loss value of the L1 loss actually reaches a lower value on a larger batch size. Although, when looked at the point at which the L1 reaches a fairly consistent point around which it starts to oscillate, for a batch size of 8 this happens already at shortly after 100 epochs. The loss of the larger batch size training gradually decreases even more while interestingly, the batch size of 16 and 8 has increased L1 loss values for the latter few epochs. For a batch size of, the training actually stabilises earlier than all the others and noticeably sets itself apart already after 20 epochs. The batch size of 4 also seems to not increase towards the later epochs, unlike the other smaller batch sizes of 16 and 8.

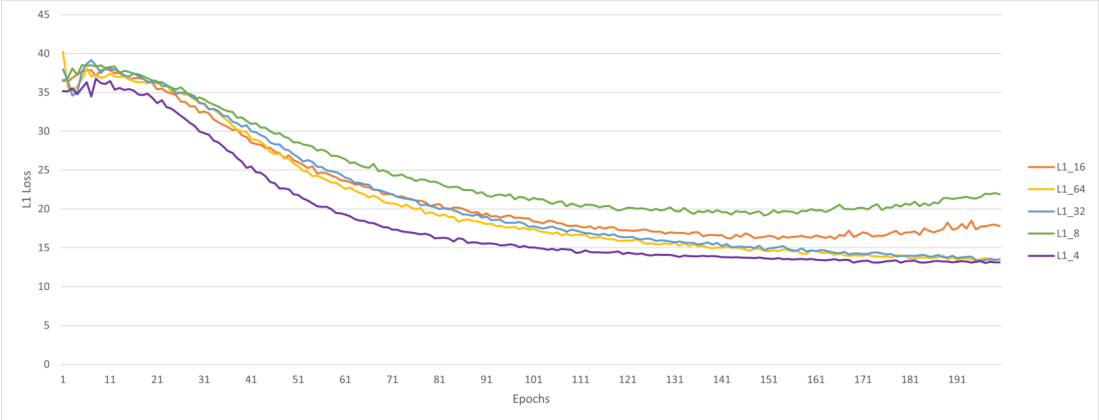

Figure 5.12: Generators L1 Loss

While the loss of the larger batch sized runs does also start to settle at fairly promising values after a certain amount of epochs, actual results do still expose artefacts even at epochs as high as 200. This can be seen in Figure 5.13 where especially the biome-maps display a small amount of noise and unclear biomes.

Ultimately we settled on using batch size of 4. This is a good medium between having accurate enough training and not putting too much strain on the used graphics card as well as not having a training time that takes too long. A batch size of 1 like the original Pix2Pix paper used might have still been interesting in terms of how fast it could achieve decent

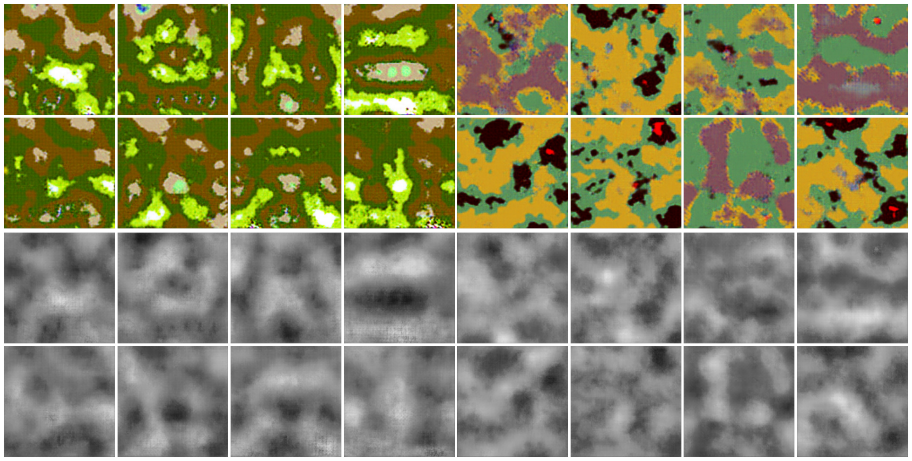

Figure 5.13: Raw generated maps with a batch size of 64 (after 200 epochs)

results. But since batch size 4 already delivered results that are satisfiable, concluded that it is a very suitable parameter for our use case.

This a little bit more detailed evaluation of how Pix2Pix GANs are affected by different batch sizes helped us in finding one that fits the best for our project and clearly showed the impact and benefits of smaller batch sizes when training GANs. Too few batches with a very large batch size also lead to unstable training. When possible, it does make sense to use a batch size of 1 to be certain that the models have the largest amount of training iterations. But one does have to make sure to then again tune the hyperparameters accordingly since a different learning rate might be needed.

5.1.4 Theme Combination

One part of this project was to incorporate both the control over the structure of the map as well as the artistic styling of choosing different types of biomes. As was shown in Section 5.1.1, the task of creating both maps in the overworld theme as well as ones in second underworld theme worked very well.

During development, the question whether these two themes could be combined during the sketching, arose. This would enable it for users to actually utilise more of the available biomes. Additionally, it would showcase how the Pix2Pix network works on generating mixed images with styles that have been trained with separately. Figure 5.14 shows one example of our two themes combined.

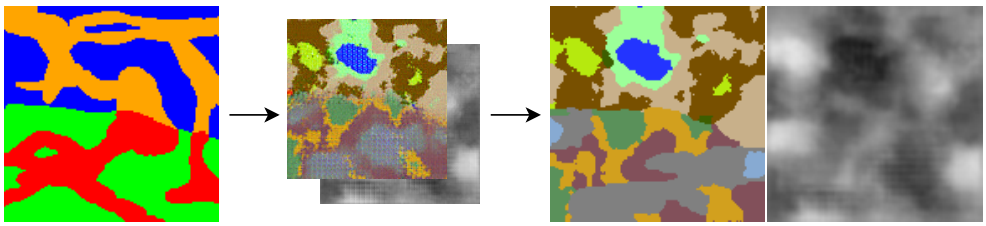

Figure 5.14: Combination of both themes in one input sketch. Centre images show the raw output, right images show the biome and height-maps after the post processing step.

As can be seen by the output on the right, the maps that have been generated still work exactly like the single theme maps shown in previous examples. This gives users even more degree of freedom and possibilities in designing their maps. During no step of this thesis has the GAN actually been trained with both themes combined. This gives more insight on the generalisation of the networks considering the exact same weights from the training of strictly only overworld and underworld inputs have been used.

Of course it would also enable it to use the traversable colour of one theme and the non-traversable of the other. This did in our case not work very well. After all, the model trained with the themes separately and hence has an easier time generating from red with green and blue with orange sketch. For example, the raw output in the middle of the figure shows some more pronounced artefacts appearing in the region where the two themes collide. How far this combination of themes can be taken will be explored more in Section 5.4.

5.2 Numerical Evaluation

A lack of metrics with which Generative Adversarial Networks can be analysed numerically still exists [29]. Nonetheless, one can still evaluate many different characteristics of the GAN. For example, since the generation of data is the main task of GANs, logically, using humans to evaluate the quality of the generated data makes sense. This is a very laborious task and of course can not be standardised [3].

A quantitative evaluation that focuses on the networks characteristics, namely the loss and the batch sizes that have been measured, will be discussed in the following section. Metrics, for example the *Inception Score*, *Frechet Inception Distance* and *Frechet Joint Distance*, which have been adapted to measure the performance of conditional GANs, focus on evaluating the generated samples by their variety of classes as well as how well they can be fitted into those classes [33, 13, 5].

These common metrics mainly serve the purpose of comparing different kinds of adversarial networks on their common data of generating images for the *CIFAR-10* dataset. It is composed of real world photographs of 10 different classes, for example *dogs*, *planes* and *horses*. The variety of the generated content as well as the quality can be improved by using the Inception Score or Frechet Inception Distance. Our model aims to generate maps of only one class (theoretically two seeing as an overworld or an underworld can be determined in the sketching step) and therefore does not really fit into the model of how these evaluation metrics work. Some attempts at using these scores on different kinds of data have already been made, but the scores obviously are not really comparable and mainly serve the purpose of comparing different GANs to each other. This has also been done in *An Enhanced GAN Model for Automatic Satellite-to-Map Image Conversion*, where the authors do state that it does not really serve a purpose of sensible evaluating the GAN with other use cases. They then just use the score on the real ground-truth data in comparison to their generated fake data and evaluate which GAN performed the best on their dataset [36]. This is also the reason why other similar papers which have been mentioned in Chapter 2 struggle to find a lot of quantitative analysis for their approaches [18, 34, 2, 19].

Hence, we have developed our own metric to quantitatively measure the performance and satisfaction of our generated samples. Our traversability-map algorithm which was used to create our dataset from Chapter 3 will be applied to the generated samples. This will enable us, to compare the actual input sketches users have made to these generated traversability-maps. For more focused measurements, actual test samples from the dataset used for validation can

be used in a similar way to actually receive a target score on how the networks results perform. More details on these measurements will be discussed in Section 5.2.2 while Section 5.2.1 will focus on the different losses.

5.2.1 Loss Analysis

As already mentioned in Section 4.2, we had to fine tune further parameters of the Pix2Pix architecture to fit our data. This was on the one hand done by looking at the actual results the models produce, but also by evaluating the losses they train with. The actual loss functions, including the L1 loss, have already been mentioned in Section 4.2.1. Some troubles have still arisen when first training the models where we initially settled on using the same learning rate for both the discriminator and the generator, namely 2×10^{-4}.

The graph of Figure 5.15 and Table 5.3 show how the losses of our model trained with batch size 8 developed over time. The increasing loss of the generator is basically just the BCE on the discriminator trying to distinguish whether generated samples match the labels. As desired in a perfect scenario, the total loss of the discriminator stabilises around 0.5. The loss is actually perfectly flat right from the beginning. This theoretically means that the discriminator is struggling with determining whether it has been shown fake or real samples, which seems suspicious.

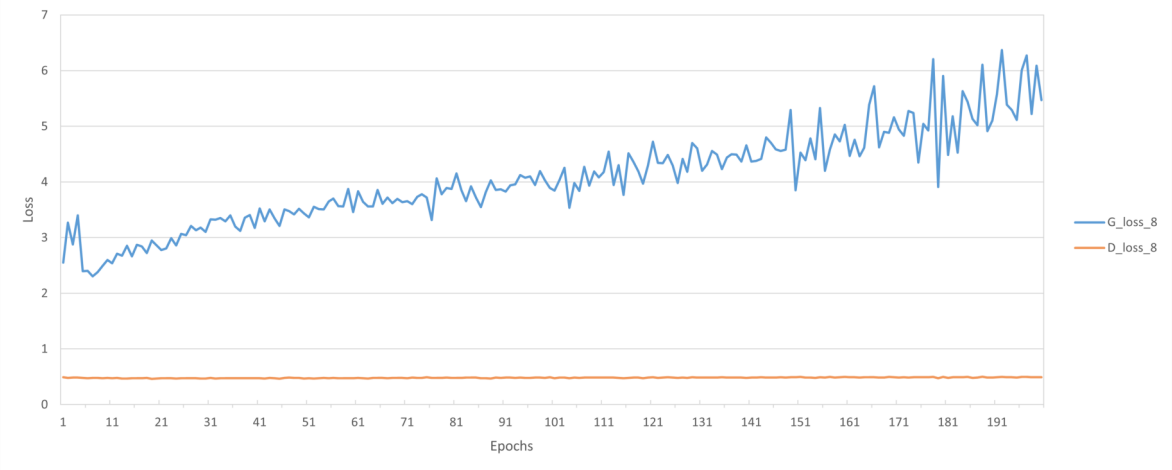

Figure 5.15: Evolution of the generator and discriminator's loss over 200 epochs

When taken a look at the actual values of the discriminators loss, as seen in Table 5.3, the total loss still stays around that 0.5 mark. One point that needs to be addressed, are the actual values of the discriminator on fake and on real samples. They show that the discriminator, when given real samples, actually maximises around 1. On the other hand, when shown fake samples, it actually minimises close to zero. This can be seen graphically in Figure 5.16. This does mean that the training of the GAN actually collapsed and it implies that the generators results are so easily distinguishable for the discriminator, that it detects them immediately and with almost 100% accuracy.

One might conclude from this, that the actual results of the generated maps are so bad, that even the discriminator does not have any trouble in determining fake and real samples. But surprisingly, as seen in Figure 5.17 the output maps do look very clean and satisfying in quality.

One possible reason why the output images still might be easily identified, could be because

	#1	#10	#25	#50	#100	#200
D_loss	0.4913	0.4802	0.4752	0.4688	0.4906	0.4927
D_fake_loss	0.1545	0.1354	0.1064	0.0987	0.0743	0.0296
D_real_loss	0.8258	0.8280	0.8450	0.8388	0.9068	0.9558
G_loss	2.5487	2.5992	3.0701	3.4374	4.0277	5.4698
L1	36.464	38.146	35.396	28.583	21.132	21.904

Table 5.2: Loss after 1, 10, 25, 50, 100 and 200 epochs

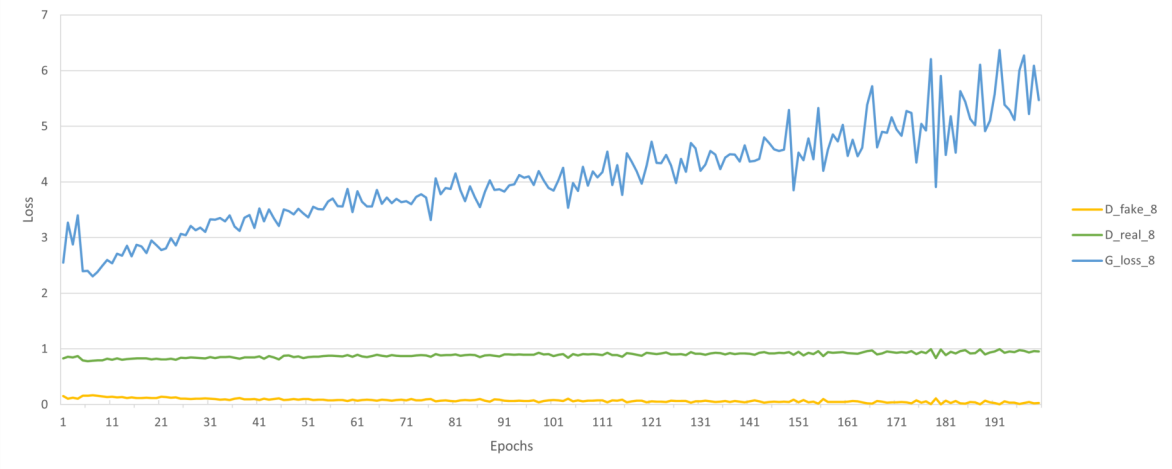

Figure 5.16: Evolution of the generator's loss and the collapsing discriminator's fake and real loss over 200 epochs

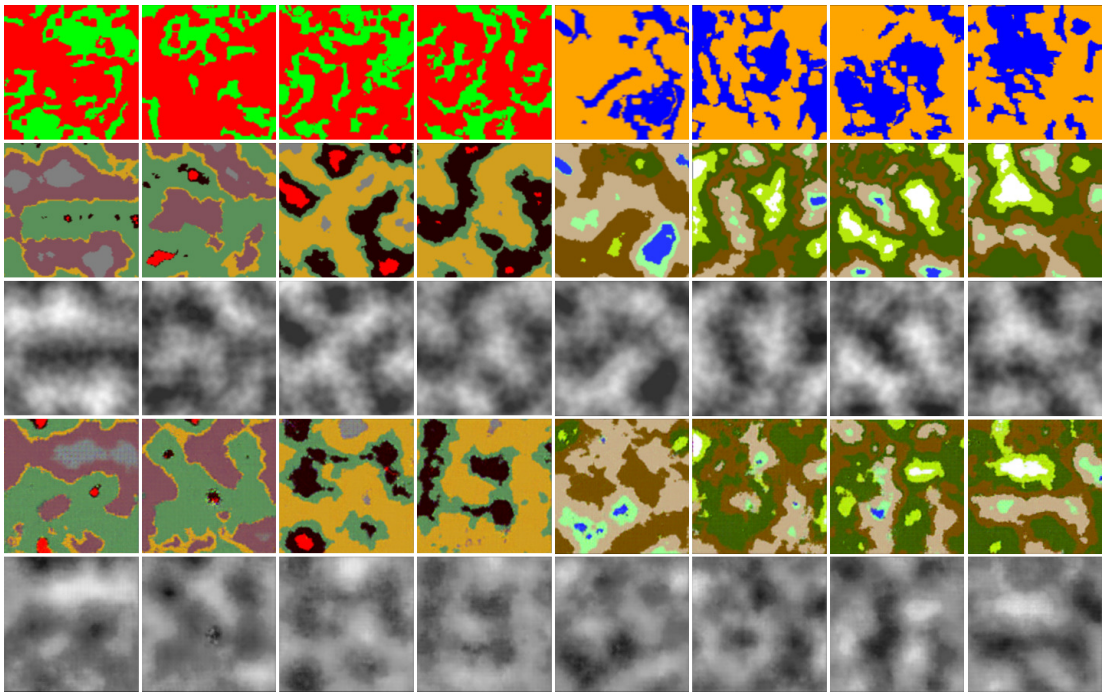

Figure 5.17: Raw example output after 200 epochs despite collapsing discriminator loss

the biomes that are generated actually are an image with RGB colours ranging from 0 to 255. They are thus in that regard unlike the training samples which have set colours for each of the biomes and not the full range of RGB. It might be interesting to apply some sort of filter that makes the samples the networks train with not as clean, which, if that is actually the reason for these values, could set the losses right again. Since these loss values would suggest that the results are actually of low quality, they could on the other hand also push the generator more strongly in the direction of generating better images. Nonetheless, the losses still do not really signify stable training of a GAN.

To circumvent this training collapse, we have decided to adjust the learning rate with which the networks train. Considering the layers of the generators U-Net architecture have also been altered changing the discriminator might help. One aspect could be to change the discriminator network structure too, but we first focused on the learning rate it trains with.

The first attempt of increasing the learning rate of the discriminator worsened the values of the losses drastically. The training collapsed even faster and never recovered from it. When trying the opposite, lowering the learning rate, we found that it helped in our case. Training with a batch size of 16 and a learning rate of 2×10^{-5} showed that the discriminators losses on fake and on real samples actually stabilised after around 60 epochs.

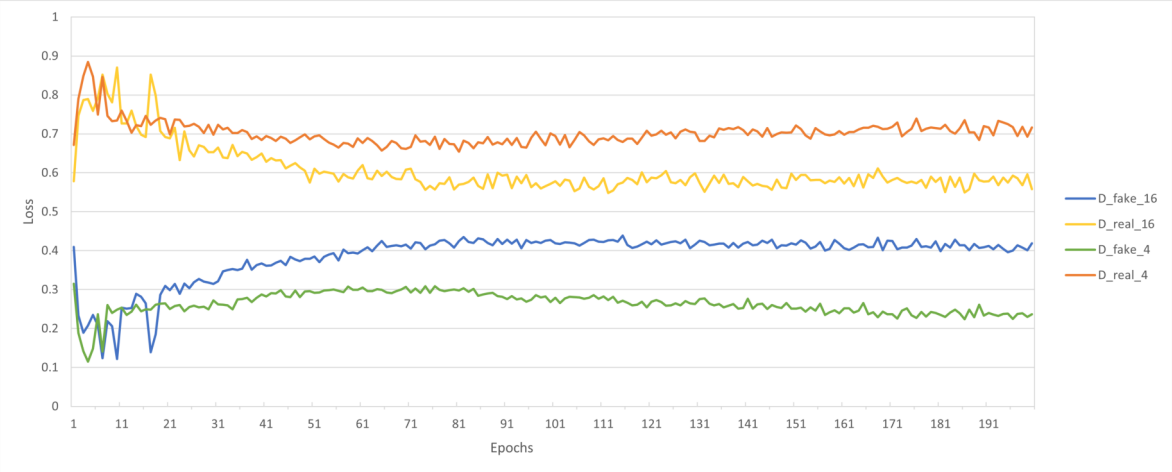

Figure 5.18: *Corrected* discriminator losses of batch size 16 and batch size 4

Figures 5.18 and 5.19 show how the loss is affected by this new learning rate. These values, especially the discriminator on fake and real samples, show that the training no longer collapses. The losses stabilise at a decent value of just below 0.6 for the real samples and around 0.4 for the fake ones in case of a batch size of 16. Interestingly, the losses of a smaller batch size are actually worse. The results do still look better with smaller batch sizes although this worse loss would imply that the discriminator has an easier time with distinguishing generated samples from real ones. This is probably because the learning rate has to be micro-adjusted to account for smaller batches. Moreover, these losses actually start to worsen again and do not seem to be as stable as the ones with a batch size of 16.

Important to address, the decreased learning rate also resulted in images that were not as clear as before. Example images can be seen in Figure 5.20 and 5.21 where it is very visible that a learning rate of 2×10^{-4} compared with 2×10^{-5} look way better. This might be due to the fact that the network just learns slower and takes longer to converge. It does show, that despite the training of the discriminator collapsing, the output of the generator can still be of use.

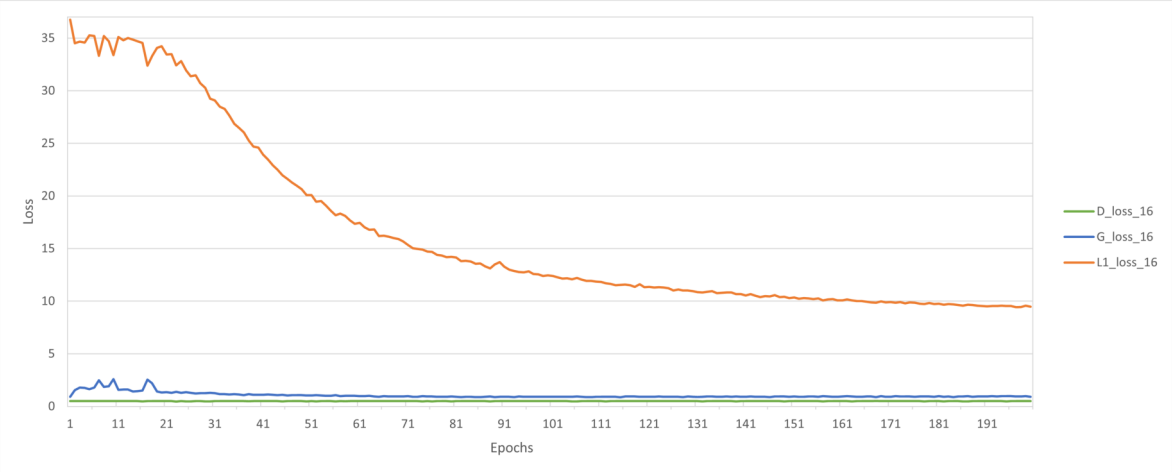

Figure 5.19: *Corrected* losses of batch size 16

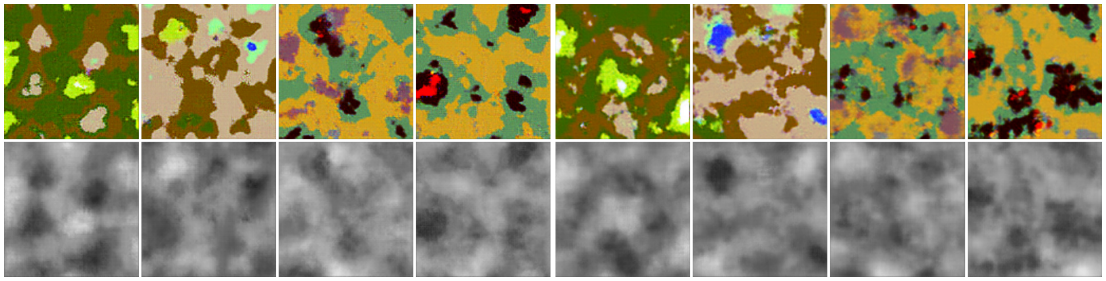

Figure 5.20: Raw comparison between different learning rates on batch size 16: 2×10^{-4} (4 samples on the left) and 2×10^{-5} (4 samples on the right)

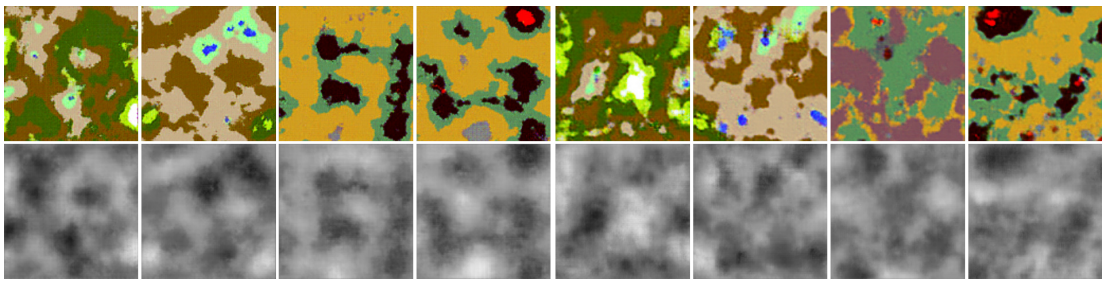

Figure 5.21: Raw comparison between different learning rates on batch size 8: 2×10^{-4} (4 samples on the left) and 2×10^{-5} (4 samples on the right)

Different loss functions could also be evaluated and analysed. Ultimately, although the results do indeed look clearer with the larger learning rate, it is also important to make sure the training of the GAN is stable. We decided on the final learning rate for the discriminator to be 2×10^{-5}. With it, the results do match the expected outcome and produce useful samples while still being a stable system. Further evaluation with other metrics will be done on the stable network.

5.2.2 Traversability-Score

A general analysis on how parameters like the batch size and learning rate as well as the epochs affect the quality of the generated maps has been done in the previous sections. To measure meaningfully characterises how the generated maps fit the actual desires of a user, we introduce our own metric, the Traversability-Score. Since the dataset for the training already required an algorithm to create the traversability-maps as the labels of the data, the same algorithm can again be applied to the generated height and biome-maps.

This way, a new traversability-map will be created for the generated maps. When then compared to the actual ground truth label of the generated samples, a value can be determined which represents how many tiles of the maps actually match. Each pixel of the label traversability-map will be compared to the newly generated traversability-map. In the case of a perfect match, meaning the output height and biome-maps perfectly determine non-traversable areas at the same spots, a score of 1 will be achieved. The process of first using the algorithm on the original maps and then on the generated samples after can be seen in Figure 5.22.

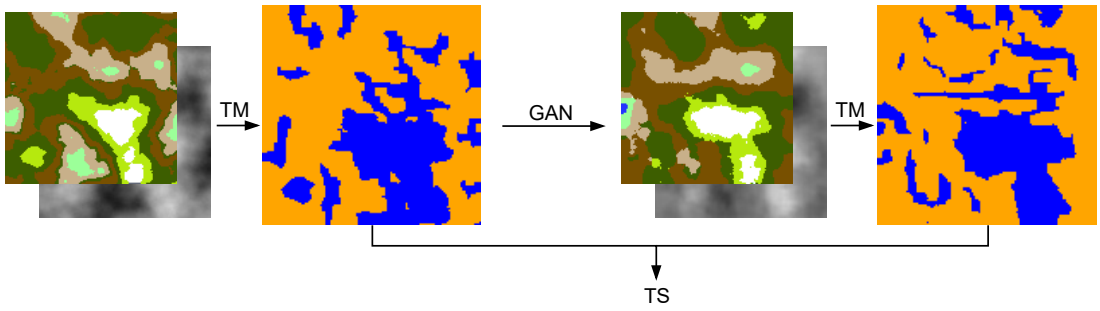

Figure 5.22: Procedure of obtaining a traversability-map, training the GAN, again obtaining a new traversability map of generated maps which can then be used to calculate the Traversability-Score (TS). In this particular exmaple, a TS of 0.7711 was achieved.

It is very visible, that the areas of non traversability match the original ground truth samples fairly well. Most of these maps that match very well achieve a Traversability-Score of around 0.7 to 0.8. This means that most of the area that can be accessed, as well as area that cannot be accessed, matches the desired area. A target score of 0.8 was therefore defined.

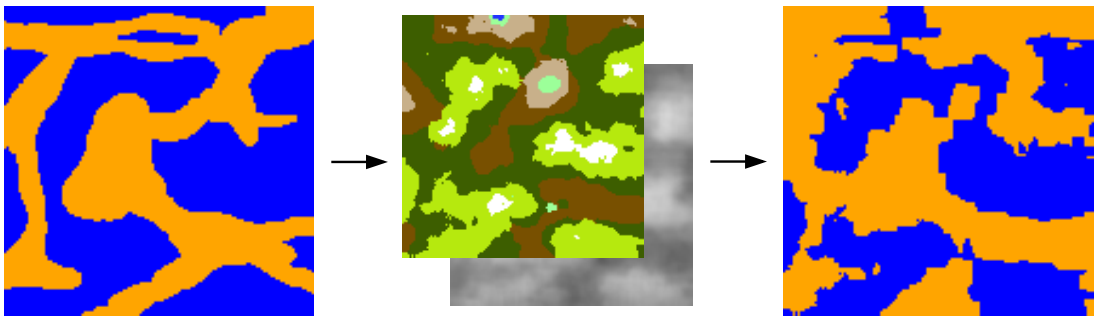

Figure 5.23: Generating maps from a sketch and traversability-map from generated maps as comparison

Similarly, the traversability-map algorithm can also be applied to the maps generated from a user's input sketch. This comparison then very well shows that the features that have been defined in the sketch, a desired area-of-play, was translated to the generated maps too. Displayed in Figure 5.23, the input sketch clearly intends to have some sort of path from the bottom left towards a more open area in the middle of the map. The left and top area of the map should also be traversable. The Traversability-Score for this sample is around 0.7753.

	#1	#5	#10	#25	#50	#200
Traversability-Score	0.4848	0.5856	0.6971	0.7380	0.7772	0.7824

Table 5.3: Traversability-Score after 1, 5, 10, 25, 50 and 200 epochs (sample size of 50 generated maps)

The score also improves well over increasing epochs. When looking at the actual results from Figure 5.7 earlier, the maps do show decent quality early on. This is also representative in the Traversability-Score presented in Table 5.3 where the score reaches the desired target values of almost 0.8 as early as 25 epochs into the training.

This value could actually also be used as some sort of additionally loss value too. The generators could not only be trained with discriminating generated samples and the L1-loss, but also with this Traversability-Score. It would ensure that the training actually focuses on working towards achieving a map similar in characteristics to the input sketch. This might be one aspect that would be interesting to investigate further in possible future works, as will be mentioned again in Chapter 6.

5.3 Usability Analysis

The main purpose of this project was to provide an easy way of creating very interesting and detailed maps. The following sections will establish that this can indeed be fulfilled by our approach. Firstly, the level of detail needed by the input sketches will be analysed. Furthermore, the actual detail of the generated maps and their features is discussed. Some further use cases and improvements that could be implemented will be addressed. Ultimately, Section 5.4 will explore, how far users can take the input sketches until undesired results arise.

5.3.1 Level of Detail

Drawing a sketch of accessible regions of a map is probably the easiest way of conditioning terrain. This still offers great possibilities for users when it comes to styling the structure of a map to their needs. The colours needed for the input sketch have already been established in Table 4.2. It is entirely up to the user how detailed they want to condition their maps. Figure 5.24 shows that both very detailed maps as well as very rough sketches can be used as input.

In the top row, the input sketch features very elaborate descriptions of the traversable regions. Narrow passages and lots of intersections have been drawn. This might be the intention of a terrain designer who desires many pathways in their landscape. In contrast, the middle sketch already contains more open areas which are still connected by some paths. The bottom row shows a sketch of a very open area-of-play. With sketches like this one, the designer could want to give more of an open-world feeling to the terrain.

All of the levels of detail have been translated well to the generated biome and height-maps. They in fact actually still show fairly even levels of detail. This entails, that the

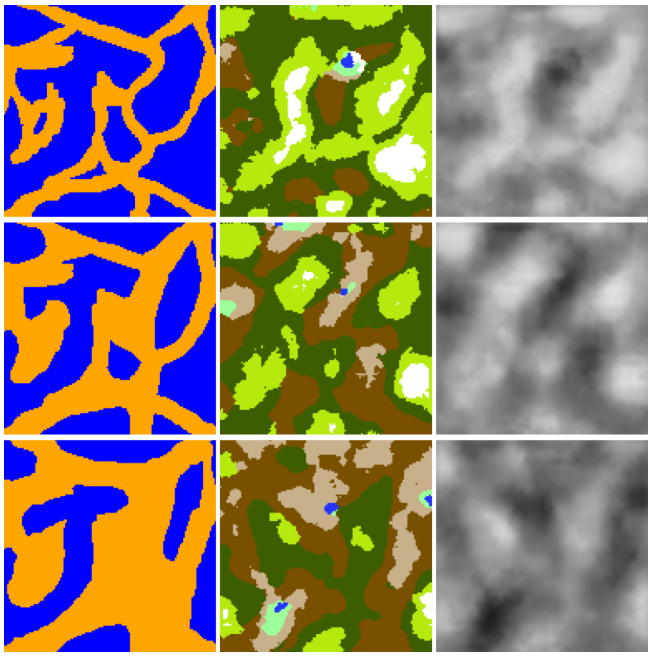

Figure 5.24: Each row consists of a input sketch with generated biome and height-maps. The first column's sketches: first with high detail, second with medium detail, third with low detail. (TS from left to right: 0.73. 0.75, 0.74)

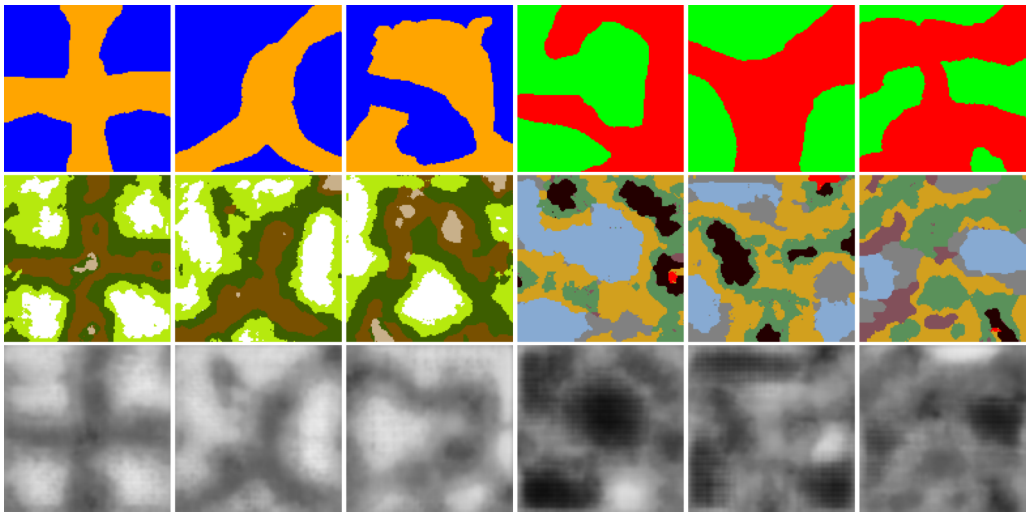

Figure 5.25: 6 samples of low detailed input sketches. Every column features an input sketch, a generated biome and a generated height-map. (TS from left to right: 0.86, 0.72, 0.77, 0.75, 0.83, 0.73)

approach can be utilised for many different applications all the way down to very accurate structuring, if necessary. Figure 5.25 shows some more examples of input sketches that are rather low detailed.

These sketches can actually be drawn in just a few seconds but still produce results of great quality. The rough conditioning of the areas of the maps still works great. Worth mentioning, the detail and diversity of the biomes actually looks very natural too. Merely

the underworld theme examples show some irregularity as can be seen in the forth column. Here the *ice* biome is predominant and the large blob in the centre of the image doesn't really feature a lot of variety and does not really fit the height-map, which shows low elevation in that region. The height-maps in this case also show some artefacts, mainly in the *rock* and *ice* biomes which in theory should be higher elevated. This is probably due to the colours being fairly similar in range and might be corrected by using a different colour palette.

All of the maps, be it of low or high detail, do still exhibit the rather stable Traversability-Score of above 0.7. which determines that the conditioning can be done very thoroughly or rather imprecisely.

5.3.2 Connectivity

Many popular video games feature the so called dungeon crawl scenario. In games that are part of this genre, the entire map will be divided into many different *rooms* which each have their own sub-map a user can explore. These *rooms* are connected through passageways. [4] We wanted to find out whether our approach will be able to generate such possibilities too.

To explore this, input sketches that line up at the edges have been drawn. As can be seen on the four maps at the left of Figure 5.26, the area of play ranges all the way to the borders in several parts that match the adjacent maps.

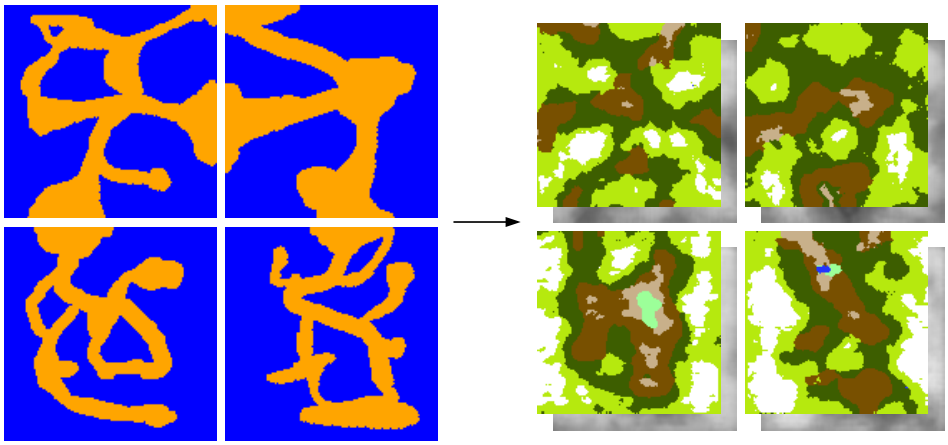

Figure 5.26: Example of maps' area-of-play connecting. Left: 4 input sketches; Right: the corresponding 4 generated outputs (TS: 0.75, 0.81, 0.71, 0.78)

The generated output maps resemble the characterised features. The bottom two maps do not share any direct link with each other but both have traversable regions towards the top of them. The biome-maps of the top row, equal to the input sketches, do have their traversable biomes stretching all the way to the middle, enabling them to connect.

These maps could therefore be concatenated together. This has also been explored in a similar manner by *Ping and Dingli* in their paper, already been introduced in Section 2. There, the authors claim that their network has learned that the maps should in some instances generate these pathways alias doors which then offer a possibility of being connected together, but do not really present any results that this could work well when concatenated [26].

We showed that a user could design their levels in this room-based approach as well and still have lots of creativity over how each of the actual maps look like. A slight difference in the exact biomes or heights at the traversable sections between each map or room would not be of too large importance because in most dungeon crawler games, passing from one room

to the next would be interrupted by a short loading screen anyway. It might even be desired to have an entirely different theme, like the underworld theme, in one room of the game. In a top-down look onto the play-area, players should just get the feeling of leaving a map at the top and entering a different room at the bottom. This can also be achieved in our approach since we decided that the accessibility of the map matters primarily.

5.4 Limitations

It can not be expected that users always feed a system with meaningful input. It is therefore useful to analyse possible edge-cases and how the network performs with dealing with them.

5.4.1 Edge Cases

Some examples of generated maps with possible attempts are discussed in the following paragraphs. The top row of Figure 5.27 shows rather meaningless input sketches which can also be given to the network as input.

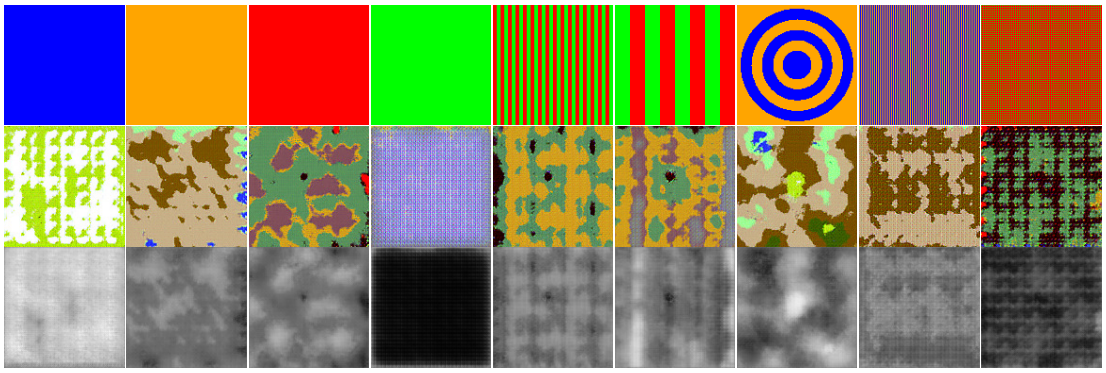

Figure 5.27: Raw examples of edge case inputs. Each column from top to bottom: Input-sketch, biome-map, height-map.

As can be seen in first four columns, when an entirely empty map has been fed, the model does not really generated very meaningful examples. Merely the entirely red map shows actually meaningful terrain. This is due to some overfitting. The dataset features some almost fully red maps and during the training process it has been learned that this output should correspond to these maps. All the other single-colour maps do feature lots of artefacts. It is still interesting to see that with a blue input sketch, which stands for untraversable terrain, a very highly-elevated terrain and only *ice* and *mountain* biomes have been generated. Similarly, with the orange traversable sketch, only the traversable biomes have been generated. This shows, that the model is indeed trying to match terrain to the input but just struggles how exactly it should do so.

When the maps have been detailed a little more, like in columns 5 to 8, the generated output looks at least somewhat sensible. It is still rather unclear and displays lots of artefacts as well as repetition like seen in the biome-map of column 5. The last two sketches are very detailed height-maps with every other pixel being a differing colour. It is clearly visible that the system struggled a lot in assigning meaningful terrain to these rather meaningless sketches.

Since we already introduced the possibility of combining the two themes in Section 5.1.4, users might also try and misuse the sketch by applying the non traversable terrain of one

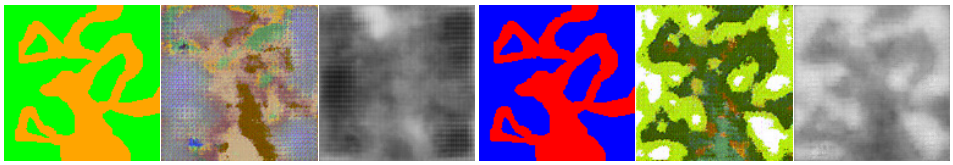

Figure 5.28: Raw examples of combining overworld traversable regions with underworld non-traversable regions and vice versa.

theme to the traversable terrain of the other. How this performs is shown in Figure 5.28. The generation did struggle a lot and displays lots of artefacts and unclear biomes.

5.4.2 Improvements

Currently, the sketches have only been generated using a graphics editor like *Adobe Photoshop*[2] or *Photopea*[3]. To improve the usability of the entire system, an actual user interface could be developed. This for example has been done by the authors of *Sentient World*, where the coarse sketch of the biomes can be drawn. In addition, after each of their iterations, some example results of maps are displayed to a user. A desired map can then be chosen with which further training or detailing can be undertaken. They then display their full resolution height-map that has been generated to the user who can export that map [18].

Some numerical feedback like the Traversability-Score of the maps could also be displayed. This way, a user immediately knows how their sketch performed. Some hints to how the map could be improved might also be beneficial to users. It would even be possible to show an actual rendering of the output maps directly in such a developed GUI.

[2]Adobe Inc., Adobe Photoshop, 2021, https://www.adobe.com/products/photoshop.html (Accessed 12-12-2021)

[3]Ivan Kuckir, Photopea, 2021, https://www.photopea.com/ (Accessed: 12-12-2021)

Chapter 6

Conclusion & Future Work

Our approach analysed the suitability of Generative Adversarial Networks, in particular Pix2Pix, for creating low poly maps. Designers can condition the synthesised maps structurally as well as artistically with minimal effort by drawing only a rough sketch of the regions that are desired to be accessible. The generated maps for the creation of 3-dimensional scenes. The results deliver both great quality and diversity and show the possibilities that GANs offer in the field of procedural content generation and computer games. Final maps have been rendered to display the actual quality of the generated biome and height-maps and to give more understanding about possible applications of this approach. Despite the generated maps being rather small in size, we showed that by concatenating multiple of them, a larger world could be generated.

Exploring whether a similar approach could be applied to larger dimensions would be interesting. More biomes, themes or even entirely different structure like rooms, dungeons or caves would also be interesting to investigate. One aspect that could also definitely be implemented would be a standalone programme that automates most of the tasks that have been exerted manually like the post-processing of the maps or mapping splatmaps for the rendering. A neat graphical user interface, possibly where the drawing is done in, could also improve the entire project and additional methods that are used for conditioning could be investigated. For example, the control over the structure of the maps as well as the mood of the scene could be further simplified by implementing Natural Language Processing that enables users to create the maps just by textual description. Additionally, it would also be interesting to explore whether more detailed scenes, for example with trees, roads, or other features, could be synthesised.

Bibliography

[1] Ryan Abela, Antonios Liapis, and Georgios N Yannakakis. A constructive approach for the generation of underwater environments. *Sixth FDG Workshop on Procedural Content Generation in Games*, 2015.

[2] Christopher Beckham and Christopher Pal. A step towards procedural terrain generation with gans, 2017.

[3] Ali Borji. Pros and cons of gan evaluation measures, 2018.

[4] Nathan Brewer. Going rogue: A brief history of the computerized dungeon crawl, July 2016. https://insight.ieeeusa.org/articles/going-rogue-a-brief-history-of-the-computerized-dungeon-crawl/ . Accessed: 2021-12-13.

[5] Terrance DeVries, Adriana Romero, Luis Pineda, Graham W. Taylor, and Michal Drozdzal. On the evaluation of conditional gans, 2019.

[6] Andrew J. Elliot. Color and psychological functioning: a review of theoretical and empirical work. *Frontiers in Psychology*, 6:368, 2015.

[7] Roland Fischer, Philipp Dittmann, René Weller, and Gabriel Zachmann. Autobiomes: procedural generation of multi-biome landscapes. *The Visual Computer*, 36(10):2263–2272, 2020.

[8] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015.

[9] Erik Geslin, Laurent Jégou, and Danny Beaudoin. How color properties can be used to elicit emotions in video games. *International Journal of Computer Games Technology*, 2016:1–9, 01 2016.

[10] Edoardo Giacomello, Pier Luca Lanzi, and Daniele Loiacono. Doom level generation using generative adversarial networks. In *2018 IEEE Games, Entertainment, Media Conference (GEM)*, pages 316–323. IEEE, 2018.

[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[12] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[13] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[16] Lawrence Johnson, Georgios Yannakakis, and Julian Togelius. Cellular automata for real-time generation of. 09 2010.

[17] George Kelly and Hugh McCabe. A survey of procedural techniques for city generation. *ITB Journal*, 14(3):342–351, 2006.

[18] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Sentient world: Human-based procedural cartography. In Penousal Machado, James McDermott, and Adrian Carballal, editors, *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pages 180–191, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[19] William A. Mattull. Toward improving procedural terrain generation with gans. Master's thesis, Harvard Extension School., 2020.

[20] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[21] Ian Parberry. Designer worlds: Procedural generation of infinite terrain from real-world elevation data. *Journal of Computer Graphics Techniques*, 3(1), 2014.

[22] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Gaugan: Semantic image synthesis with spatially adaptive normalization. In *ACM SIGGRAPH 2019 Real-Time Live!*, SIGGRAPH '19, New York, NY, USA, 2019. Association for Computing Machinery.

[23] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[24] percarprimoz. Procedural terrain generation. https://github.com/pecarprimoz/procedural-gen-dipl, 2021.

[25] Aladdin Persson. Pix2pix. https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/GANs/Pix2Pix, 2021.

[26] Kuang Ping and Luo Dingli. *Conditional Convolutional Generative Adversarial Networks Based Interactive Procedural Game Map Generation*, pages 400–419. 02 2020.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[28] Isha Salian. 'paint me a picture': Nvidia research shows gaugan ai art demo now responds to words. https://blogs.nvidia.com/blog/2021/11/22/gaugan2-ai-art-demo/, 11 2021. Accessed: 2021-11-24.

[29] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[30] Noor Shaker, Mark J. Nelson, and Julian Togelius. *Procedural Content Generation in Games*. Springer International Publishing, Switzerland, 2016.

[31] Sam Snodgrass and Santiago Ontañón. Learning to generate video game maps using markov models. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(4):410–422, 2017.

[32] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games*, 10(3):257–270, 2018.

[33] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[34] J. Togelius, A.J. Champandard, Pier Luca Lanzi, M. Mateas, A. Paiva, Mike Preuss, and K.O. Stanley. Procedural content generation: goals, challenges and actionable steps. *Dagstuhl Follow-Ups*, 6:61–75, 01 2013.

[35] Thomas Würstle. Realistic biome generation for procedural maps using essential climate principles. Master's thesis, University of Applied Sciences Ravensburg-Weingarten, 2017.

[36] Ying Zhang, Yifang Yin, Roger Zimmermann, Guanfeng Wang, Jagannadan Varadarajan, and See-Kiong Ng. An enhanced gan model for automatic satellite-to-map image conversion. *IEEE Access*, 8:176704–176716, 2020.

[37] Howard Zhou, Jie Sun, Greg Turk, and James M. Rehg. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, 2007.