Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Geolocalized procedural generation of biomes in real time

Bc. Michal Mráz

Supervisor: doc. Ing. Jiří Bittner, Ph.D.
Field of study: Open Informatics
Subfield: Software Engineering
May 2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Mráz**	Jméno: **Michal**	Osobní číslo: **457055**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra počítačů**		
Studijní program:	**Otevřená informatika**		
Specializace:	**Softwarové inženýrství**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Geolokalizované procedurální generování biomů v reálném čase

Název diplomové práce anglicky:

Geolocalized Procedural Generation of Bioms in Real-Time

Pokyny pro vypracování:

Zmapujte existující metody pro procedurální generování modelů různých biomů. Soustřeďte se na metody umožňující generování terénu a vegetace zvoleného typu biomu v reálném čase. Navrhněte metodu umožňující geolokalizované generování biomů na základě volně dostupných dat o dané lokalitě jako jsou data z Open Street Maps, digitální elevační mapy, satelitní snímkování nebo meteorologická data.
Implementujte aplikaci umožňující generování geolokalizovaných biomů v reálném čase. S využitím této aplikace navrhněte a implementujte jednoduchou geolokalizovanou hru. Pro implementaci využijte herní engine Unity nebo Unreal. Cílovou platformou pro aplikaci budou přenosná zařízení jako jsou mobilní telefony a tablety. Vyhodnoťte rychlost generování biomů v závislosti na jejich vizuální komplexitě a použitém hardwaru. Vyhodnocení proveďte v nejméně třech různých lokalitách. Vytvořenou hru podrobte základnímu uživatelskému testu.

Seznam doporučené literatury:

[1] Niese, T., Pirk, S., Albrecht, M., Benes, B., & Deussen, O. (2022). Procedural Urban Forestry. ACM Transactions on Graphics (TOG), 41(2), 1-18.
[2] Ecormier-Nocca, P., Cordonnier, G., Carrez, P., Moigne, A. M., Memari, P., Benes, B., & Cani, M. P. (2021). Authoring consistent landscapes with flora and fauna. ACM Transactions on Graphics (TOG), 40(4), 1-13.
[3] Galin, E., Guérin, E., Peytavie, A., Cordonnier, G., Cani, M.-P., Benes, B., & Gain, J. (2019). A Review of Digital Terrain Modeling. Computer Graphics Forum, 38(2), 553–577.
[4] Pirk, S., Benes, B., Ijiri, T., Li, Y., Deussen, O., Chen, B., & Měch, R. (2016). Modeling Plant Life in Computer Graphics. ACM SIGGRAPH 2016 Courses, 18:1–18:180.
[5] Smith, G. (2017). Procedural content generation: An overview. Level Design Processes and Experiences, 159-183.
[6] Emilien, A., Bernhardt, A., Peytavie, A., Cani, M. P., & Galin, E. (2012). Procedural generation of villages on arbitrary terrains. The Visual Computer, 28(6), 809-818.
[7] Kybartas, B., Bidarra, R., & Meyer, J. J. C. (2016). Procedural generation of populations for storytelling. In Proc. PCG 2015-Workshop on Procedural Content Generation for Games, co-located with the Tenth International Conference on the Foundations of Digital Games (IKBC15), 2015.
[8] Tuncel, M. Procedural Content Generation for Video Games using Open Data. Master Thesis, MFF UK, 2019.
[9] Gasch, C., Sotoca, J.M., Chover, M. et al. Procedural modeling of plant ecosystems maximizing vegetation cover. Multimed Tools Appl 81, 16195–16217 (2022).
[10] Rapp, Daniel, Niebling, Florian, Latoschik, Marc. The Impact of Pokémon Go and Why It's Not about Augmented Reality - Results from a Qualitative Survey. In proceedings of the International Conference on Virtual Worlds and Games for Serious Applications, 2018.
[11] Kati Alha, Elina Koskinen, Janne Paavilainen, Juho Hamari. Why do people play location-based augmented reality games: A study on Pokémon GO. Computers in Human Behavior, Volume 93, Pages 114-122,.2019.
[12] Petr Nahodil. Procedural Scene Generation for Train Simulator. Bachelor Thesis, Czech Technical University in Prague, 2023.
[13] Ondřej Kyzr. Procedural generation of outdoor scenes. Bachelor Thesis, Czech Technical University in Prague, 2023.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jiří Bittner, Ph.D.   Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **14.08.2023**   Termín odevzdání diplomové práce: ______________

Platnost zadání diplomové práce: **16.02.2025**

doc. Ing. Jiří Bittner, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to thank my supervisor for supporting me and providing me with valuable advice and guidance. I would also like to thank my family and friends for supporting me.

Declaration

I declare that I created the presented work independently and that I have listed all information sources in accordance with the Methodological Guideline on Adherence to Ethical Principles in Preparation of Graduation Theses.

I created the thesis with the help of the AI tools Writefull, ChatGPT, and GitHub Copilot.

In Prague, May 2024

Abstract

This work deals with research, design, and implementation of a system that automatically generates game environment around user's physical position in real time. For that, it uses representation of real terrain, streets, and buildings combined with automatically generated biomes and vegetation. The work also deals with the design and implementation of a prototype game, which is used to test the system with real users and gather their feedback.

Keywords: procedural generation, biomes, vegetation, game environment, pervasive game, location-based game, mobile game, gps

Supervisor: doc. Ing. Jiří Bittner, Ph.D.
Karlovo Namesti 13,
121 35 Praha 2,
Czech Republic

Abstrakt

Tato práce se zabývá výzkumem, návrhem a implementací systému, který automaticky generuje herní prostředí kolem uživatelovy fyzické polohy v reálném čase. K tomu využívá reprezentaci reálného terénu, silnic a budov, které kombinuje s automaticky generovanými biomy a vegetací. Práce se také zabývá implementací herního prototypu, který je využit pro testování systému s reálnými uživateli a pro získání jejich zpětné vazby.

Klíčová slova: procedurální generování, biomy, vegetace, herní prostředí, pervazivní hra, location-based game, mobilní hra, gps

Překlad názvu: Geolokalizované procedurální generování biomů v reálném čase

Contents

Figures

Tables

Chapter 1
Introduction

The goal of this work is to explore methods for procedurally generating a geolocalized virtual game environment with biomes and vegetation, which could be used as a basis for a location-based mobile game. We want to find what options are available, how these options are suitable for real-time generation, and what existing solutions are there. Then we are going to design and implement a procedural biome generation system, create a simple mobile game prototype, test it with users, and gather their feedback. We will also profile the prototype so that we can find out where the bottlenecks of our current solution are and what to focus on in the future.

The word **geolocalized** here means that we are using the user's real-time location to determine his location in the game environment.

A **location-based game** is a category of game in which the player's physical location drives the game's progression. These games are commonly played on smartphones that utilize GPS to determine the player's location. **Procedural generation** is a way to create data algorithmically instead of manually. Using it, we can create a vast environment that combines manually created content with randomness. Procedural content generation (PCG) has the ability to enhance the replayability and adaptability [1] of the game environment. In our case we use PCG because of it's adaptability, to provide game environment adjusted to the geographic location of the user.

Biomes are areas with particular environmental conditions that support the survival of a specific group of flora and fauna. In games biomes contribute to a more varied environment, they serve as natural landmarks and also can be connected to gameplay.

In the context of this work, **real-time computation** refers to the continuous updating of the game's state and visual representation of the game while the player interacts with it. This is often referred to as *soft real-time* [22], meaning that the system performance would be degraded if results were not produced in a timely manner.

1.1 Biomes

Different biomes can have different content, challenges, and variations of gameplay. Examples of biomes can be forests, deserts, jungles, oceans, or mountains, or they can be also entirely fictional. Biomes can differ in the vegetation that grows in them, in weather, available resources, or animals that live there, etc.

Here are a few examples of how biomes have been used in computer games in recent years.

Minecraft (2011) [39] is a sandbox survival game featuring a procedurally generated 3D world where players collect resources and craft items. Minecraft uses biomes to create environments that differ in their flora, fauna, resources, generated houses, shapes of terrain, enemies, audio, and sky color. Biomes separate the world into parts that offer distinct gameplay.

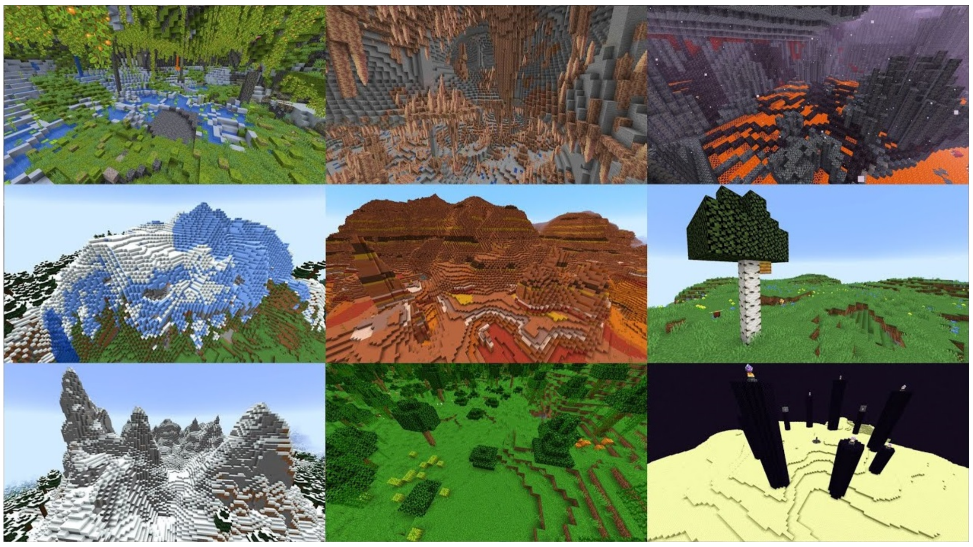


Figure 1.1: Some of the biomes in Minecraft. From left to right, top to bottom: Lush Caves, Dripstone Caves, Basalt Deltas, Frozen Peaks, Badlands, Plains, Jagged Peaks, Sparse Jungle, The End [29]. Source: Mojang Studios [23]

Terraria (2011) [38] is an RPG sandbox game which uses biomes similarly to Minecraft, but it takes place in a 2D world. The biomes contain different enemies and resources that are critical for progression.

No Man's Sky (2016) [40] is a survival game with procedurally generated planets. Each planet has one biome, and that encourages the player to explore the other planets too. The biomes influence how dangerous each planet is and what flora and fauna are there. Different terrain variations influence the appearance of the planet; the types are, for example, pangean, continental, swamp, archipelago, island chains, oceanic, and reef. Each biome has unique resources that can be harvested there. There are 11 types of biomes such as lush, barren, frozen, toxic, volcanic, and marsh [2].

Subnautica (2018) [42] is a survival game set on an ocean planet with various biomes such as deep-sea trenches, kelp forests, and coral reefs. The biomes offer different resources and challenges for the player. Exploring the different biomes is a core gameplay aspect of Subnautica.

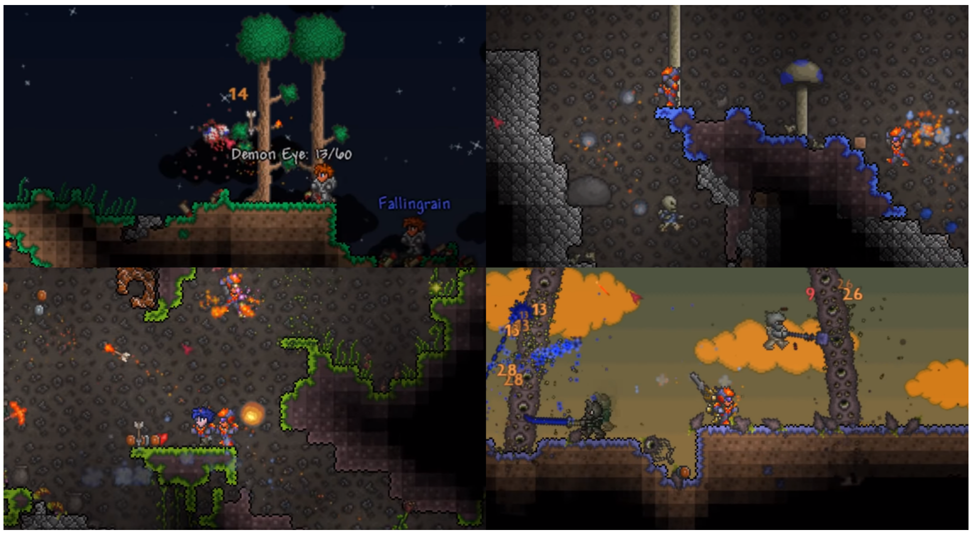


Figure 1.2: Some of the biomes in Terraria. From left to right, top to bottom: Forest, Underground Mushroom, Undeground Jungle, Corruption [32]. Source: Re-Logic

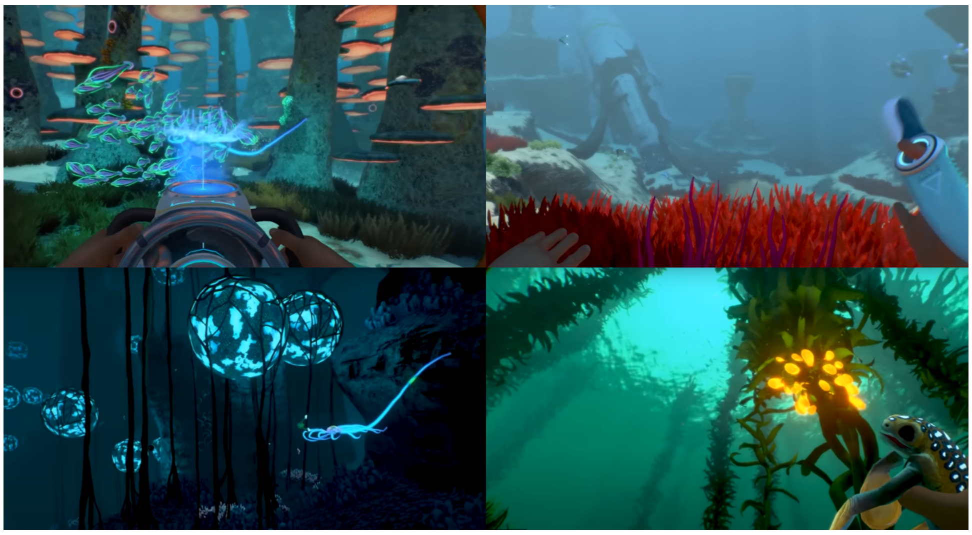


Figure 1.3: Some of the biomes in Subnautica. From left to right, top to bottom: Mushroom Forest, Grassy Plateaus, Grand Reef, Kelp Forest [30]. Source: Unknown Worlds Entertainment

RimWorld (2018) [41] is a colony building and management game with different biomes such as tundra, desert, and tropical jungle that affect the gameplay. Depending on the biome in which the player started his colony, he can face unique challenges, for example, low temperatures, short growing season, heatwaves, lack of trees to harvest wood from, or dangerous wild animals and diseases. The game takes place on small 2D procedurally generated maps in different parts of the world. There is only one biome on the whole map, but maps in different parts of the world can have different biome.

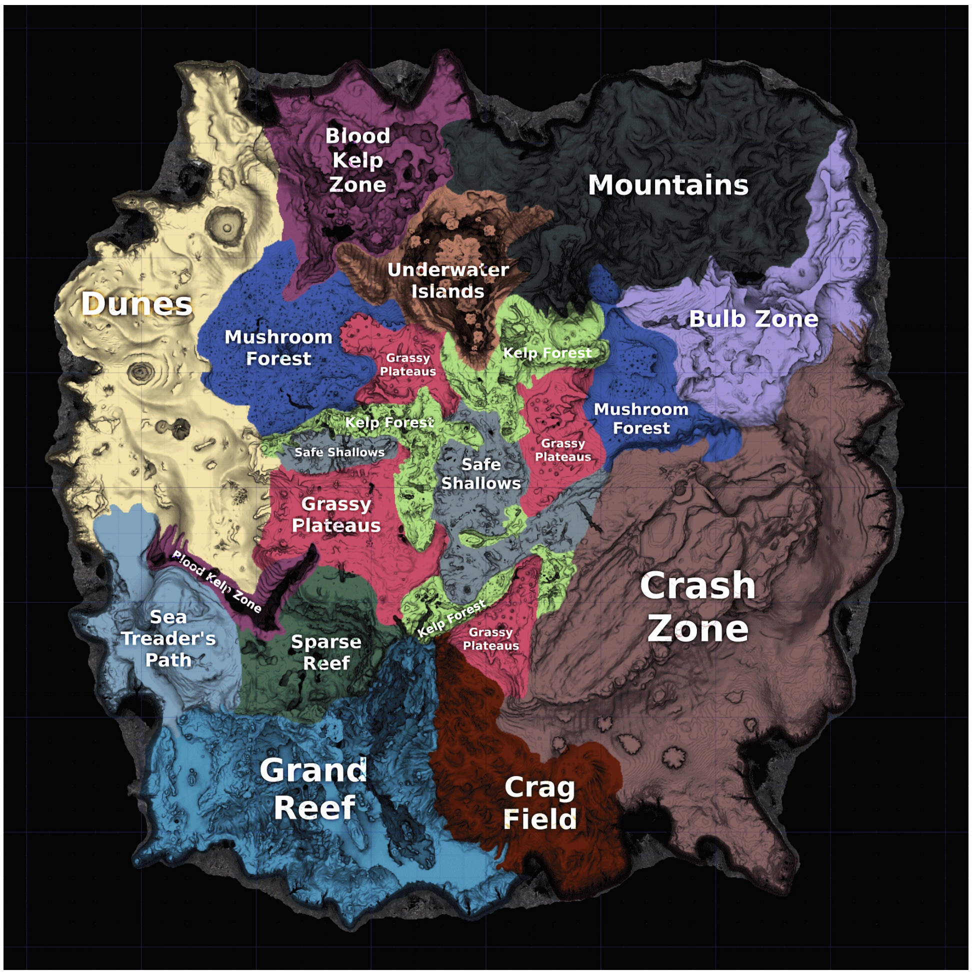


Figure 1.4: The surface biomes from the game Subnautica. Each biome acts as a small-scale ecosystem reflecting those found in real world. Every biome offers unique flora and fauna to explore and various resources to gather [6]. Source: Subnautica Wiki [5]

In general, biomes are used in computer games to create a varied and interesting environment, add challenges and variety to the gameplay, and inspire players to explore.

1.2 Location-based games

Almost everyone has a smartphone nowadays. It has a potent GPU, a GPS module, and often also mobile internet and is always readily available in one's pocket. That is why it makes sense to craft immersive interactive experiences for these devices such as location-based games. Here are some examples of well-known location-based games from recent years:

Pokémon Go (2016) [43] is a mobile game where the player moves in the game world by walking in the real world and where he locates, captures,

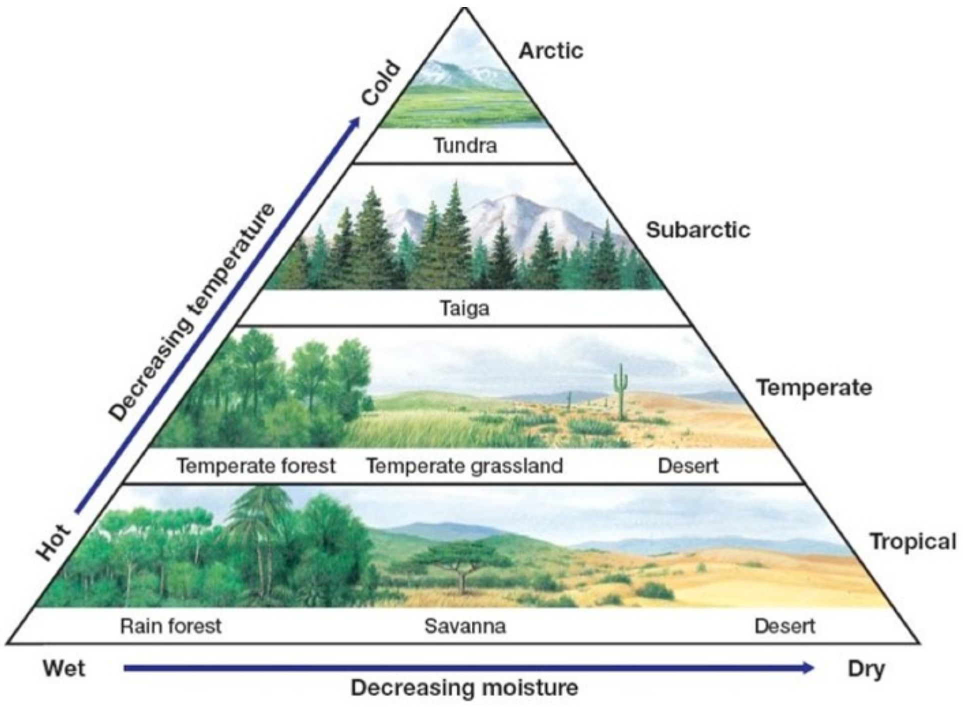


Figure 1.5: An overview of Earth biomes and how they change depending on changing moisture and temperature. Source: Mellisa A. [7]

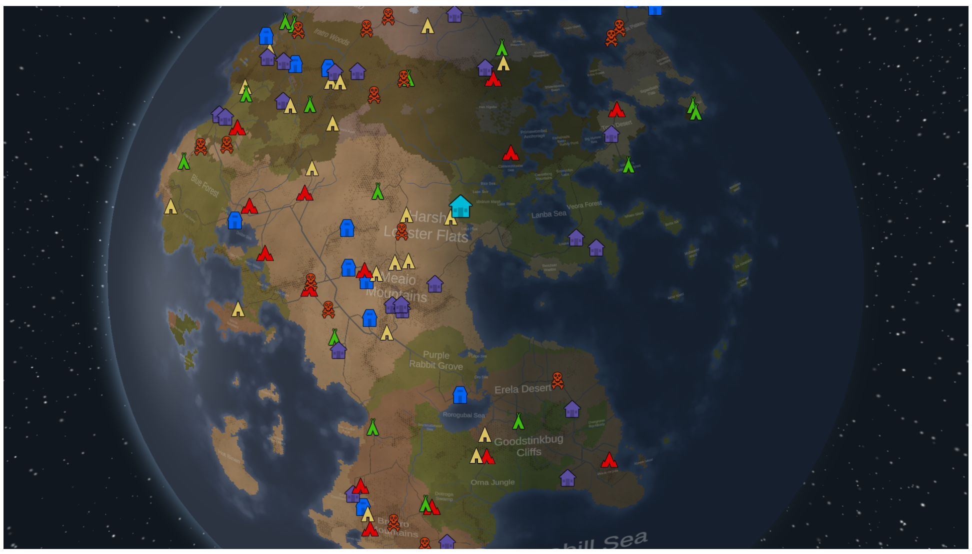


Figure 1.6: Generated planet in RimWorld with different biomes visible. Source: Ludeon Studios

trains, and battles virtual creatures, which appear on the in-game map as if they were in the player's real-world location.

Ingress (2013) [44] is the predecessor of Pokémon Go also by the developer Niantic. Ingress is a game about territory control. Device GPS is used to interact with portals in user's proximity. Portals are placed in places of

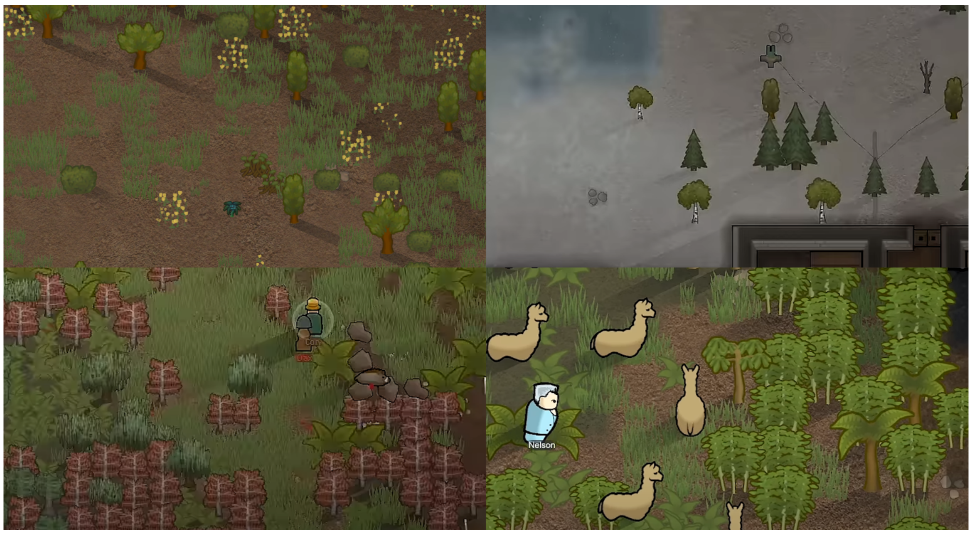


Figure 1.7: Some of the biomes in RimWorld. From left to right, top to bottom: Temperate Forest, Boreal Forest, Cold Bog, Tropical Rainforest [31]. Source: Ludeon Studios

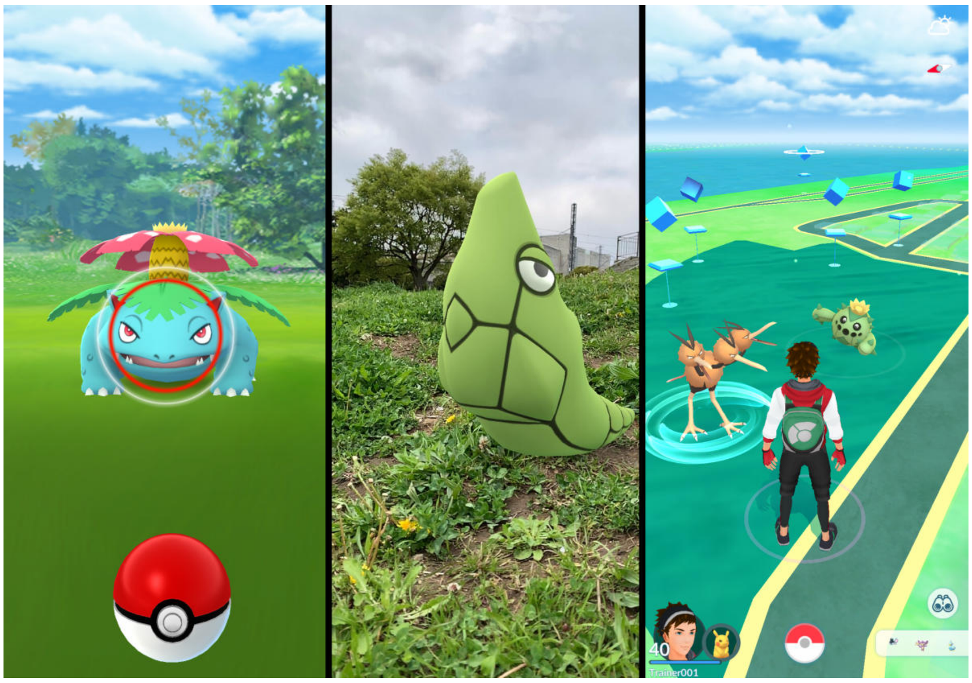


Figure 1.8: Pokémon Go screenshots. From left: catching Pokémon, taking photos of Pokémon in augmented reality, walking on the world map. Source: Niantic [33]

interest such as statues, community hubs, unique architecture etc. [3].

Geocaching (2000) is an outdoor activity in which participants use a GPS device to hide and seek containers, called "caches", at specific locations marked by coordinates all over the world. The players use the coordinates to find the

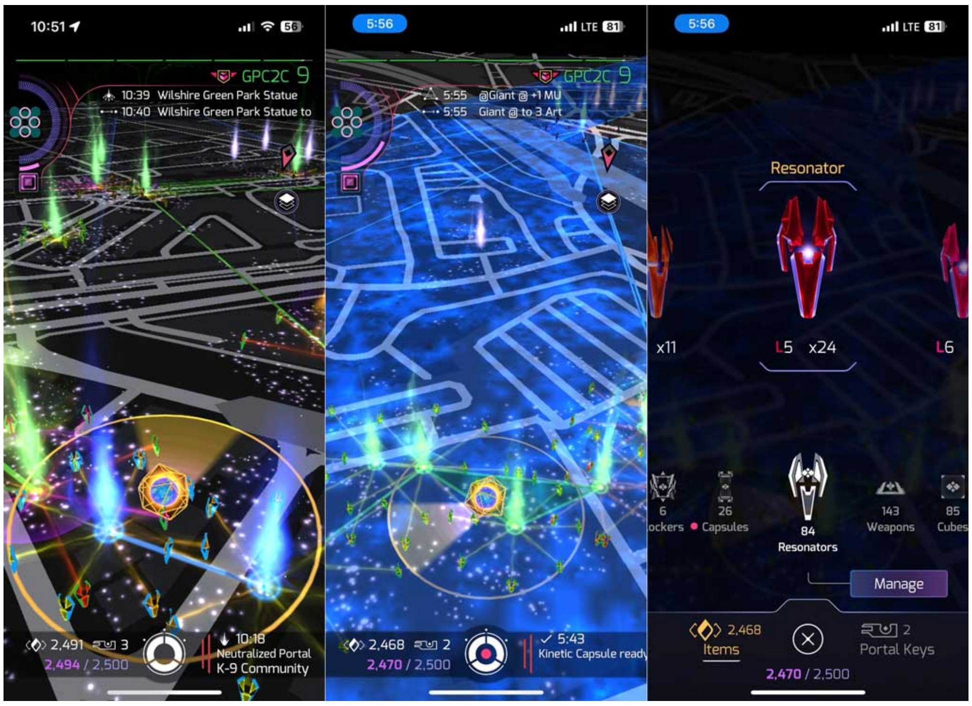


Figure 1.9: Ingress screenshots. Grey portals are unclaimed, green and blue portals are claimed by the green and blue player factions. The orange circle repesents the player's area of influence. The blue area is the are claimed by the blue faction. In the last screenshot the user is selecting a resonator to put on a portal to claim it or to upgrade it. Source: Niantic [33]

caches. Those can be found in parks, trails, urban areas, and even underwater. The caches usually contain a logbook where finders log their visit and they can also trade small items through the caches with other players.

Chapter 2
Related work

2.1 Why do people play location-based games?

Perhaps the most well-known location-based game is Pokémon GO. To understand why players are drawn to such games, we will discuss a study and survey by Alha et al. [16], which was distributed in Finnish Pokémon GO groups on Facebook.

These were some of the reasons why respondents **started playing the game**: They found the game interesting and fun, liked its novelty, and felt that it was different from other games. Some saw funny pictures of the game on social media. They mentioned the importance of positive characteristics of playing, such as spending time outside, getting physical exercise, or exploring new areas. Some played because it was a good opportunity to meet new people, and some were excited by the "treasure hunt"-like gameplay. The nature of the game was described as being casual, easy to play, and easy to install.

These were some of the reasons why respondents **continued playing the game**: progression, achieving personal goals, joy of discovery, collecting. People still found the positive aspects of playing important: exercise, being outdoors, and the game being a reason to walk. This was important for the depressed. The game mechanics became one of the main reasons to continue playing. Social features such as meeting other people, teaching others the game mechanics and comparing progression were important. The game provided surprises and was rewarding.

And these were the most important reasons why people **stopped playing**:

1) The current situation of the respondent - lack of time, lack of money, poor weather, having reached his personal goals in the game, his phone not working, not having internet, etc.

2) Slow progression - the leveling curve was seen as being too steep, people mentioned slow advancement to the next level and little new content. The survey authors also mention that it would be good if the developer added more short term goals for the player so that the player can celebrate "small victories" more often instead of grinding for the long-term goals.

3) Bugs - bugs were the third most important reason to quit and the survey authors say that it is important to focus more on quality control.

Furthermore, the respondents did not like the game content being concentrated mainly in city centers. Some saw the game as shallow and simple. Some found the game to be unrewarding, random in its rewards, lacking a challenge, and too competitive. The main reason for Pokémon GO achieving a wide player base was previous experience with the Pokémon franchise, where the characters are simple and attractive. An interesting thing about Pokémon GO is that it enables binge playing (apart from what is common in the free-to-play mobile games genre), meaning it does not limit the time the player can spend with the game per session. The reason why it can do this is that it includes enough long-term goals that even with constant playing it takes a long time to finish them.

2.2 Why Pokémon GO is not mainly about augmented reality (AR)

A definition of augmented reality: "Augmented reality is an interactive experience that enhances the real world with computer-generated perceptual information. Using software, apps, and hardware such as AR glasses, augmented reality overlays digital content onto real-life environments and objects" [17].

A discussion of the results of a qualitative survey by Rapp et al. [18] states that although Pokémon GO is sometimes called an AR game, AR is hardly ever used in the game, AR features suffer from bad usability and actually make playing the game harder. When players describe the game as an AR game, they mostly mean the link between the real and virtual world in the game, but that is done by the geolocation feature. The main effect of the AR feature can be seen in traditional and social media, where the app was used to take funny pictures of Pokémon.

2.3 Physics based approach to biome generation

Fischer et al. in their paper AutoBiomes [15] deal with creation of a tool that automatically creates a virtual environment using a pipeline that consists of four steps: generation of rough base-terrain, climate simulation, biome-based terrain refinement, and asset placement. What is interesting is that the terrain changes based on the biome selected for that area. For the computation of the biome distribution, they use a physics-based approach that is in contrast to often used noise methods. They developed a climate simulation but had to keep it computationally relatively simple to keep the computation fast. The goal of this approach was to add physically plausible realism to the terrain. They say that if they used noise functions instead, the result would be less realistic and would require more fine-tuning. In the simulation, they compute temperature, wind, precipitation, and in the end biome classification.

But the question is whether this approach would be good for us because we think that when we keep the terrain elevation around the user the same as

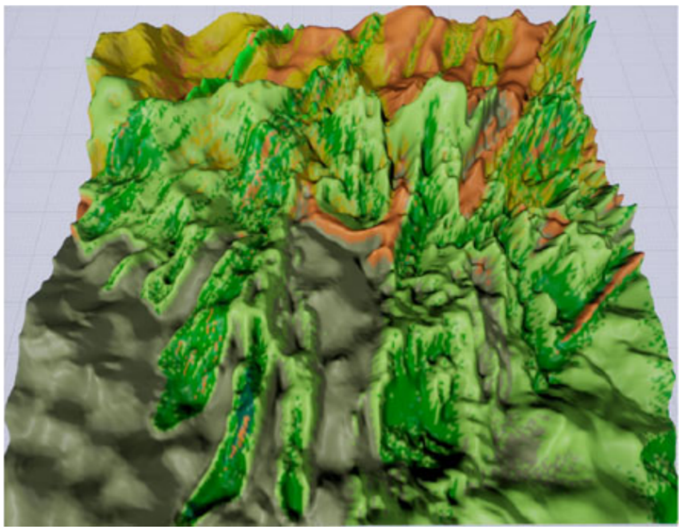


Figure 2.1: An example of the result terrain and biome distribution of AutoBiomes. Source: Fischer et al. [15]

in the real world, it helps him to orient better in the environment and better plan his walking path through the environment. In future work, the terrain could be modified locally, on a small scale, according to its biome so that the overall shape of the terrain is not changed, and at the same time the biomes get more detail.

2.4 Ecosystem simulation

In a paper by Ecormier-Nocca et al. [14] they describe their approach to plant generation, where input is provided by the user. The input consists of a digital elevation model terrain description and other information for computing environmental resources for plants, namely minerals map, terrain orientation, latitude, altitude and extreme temperatures at sea level, describing the targeted climate. They extracted yearly moisture, extreme temperatures, and sun-light maps of the terrain from this input.

They recognized that animals have an important impact on the appearance of the environment, they create trails, clearings, and even affect erosion. The key to simulating this is the competition for resources. They track the resources for each species using a graph, along with resource location and accessibility. In their result, visible animal trails are present.

2.5 Who and when should generate the environment?

Smith et al. [1] ask whether the environment should be generated on a client or on a server and whether online, while the player is playing, or offline when

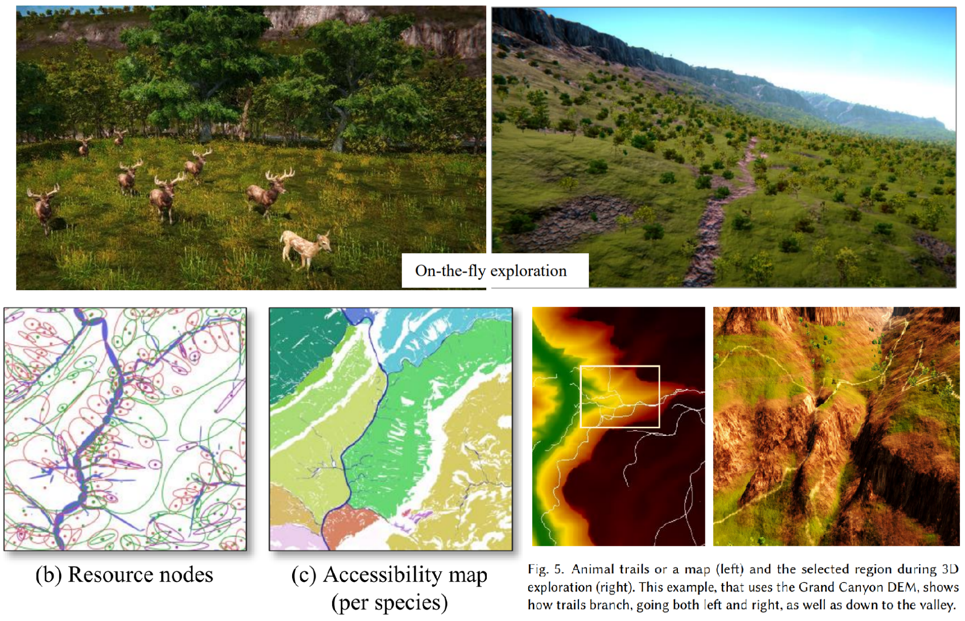

On-the-fly exploration
(b) Resource nodes
(c) Accessibility map (per species)
Fig. 5. Animal trails or a map (left) and the selected region during 3D exploration (right). This example, that uses the Grand Canyon DEM, shows how trails branch, going both left and right, as well as down to the valley.

Figure 2.2: Animal trails were generated using a graph that maps resources and species-specific accessibility of the environment. Source: Ecormier-Nocca et al. [14]

the game is being loaded or when it is being developed. If the generator runs online and frequently, then performance is a big concern and it can result in us needing to compromise quality. On the other hand, if the generator runs offline, there are lower demands on its speed, and it has to store and load the generated content efficiently.

We will be generating parts of the environment around the player while he is playing, so we will be doing online generation. If we generated environment on a server, it would mean less work for the client, better battery life, and more data sent over the network. The server would know what the environment around the player looks like (which could be useful if we wanted the server to influence the gameplay or to add some multiplayer functionality in the future), but the work the server has to do for every player would probably add up. If we generated the environment on the client, we could possibly have an almost offline client apart from downloading streets, buildings, and elevation data. The client would deplete its battery faster and we could be more limited by performance.

2.6 Player searching in the environment

Smith et al. [1] discuss how PCG facilitates exploration, allowing players to navigate and explore the environment, discover surprises, and enjoy unique experiences similar to those in Minecraft. Additionally, players might encounter smaller PCG elements within the environment, similar to those in the Borderlands game. The key takeaway is that generators should create content that is both unexpected and exciting for players to discover. The introduction of surprise and diversity presents challenges. It might involve integrating minor, manually crafted content into the generator or meticulously designing grammar rules and weights.

2.7 Urban forestry

Niese et al. [4] recognized in their work that vegetation in cities is heavily affected by pruning and also by the surrounding urban area. They used a neural network for learning plant distribution in land lots. The neural network learns from satellite images and adjusts the parameters of a planting strategy within a parameter space. They identified several common vegetation placement strategies, which can be seen in figure 2.3. We used the Semi-Random strategy as an inspiration for our work.

2.8 Digital terrain modeling

Galin et al. [9] recognize that terrain is crucial for 3D scenes. Their work is an overview of current techniques for modeling and authoring terrain. They talk about procedural modelling, physically-based simulation and example-based methods that use scanned real world terrain. They divide terrain representations into elevation models and volumetric models that allow overhangs.

The elevation model can be in the form of function representation or discrete heightfields representation or layered representation. The function representation requires little memory, but the evaluation of the function can be computationally expensive. The discrete heightfields representation is the most common; it is a regular 2D grid of points where each point has a height value assigned. The accuracy of this representation is limited because of the regular spacing of the points. The terrain between the points is reconstructed using bi-linear interpolation. In our work, we use this representation. Higher-order interpolation is more computationally intensive and also requires using points from larger neighborhood for the computation. This representation is more memory-intensive than function representation. Layered representation describes different layers of terrain as ordered functions. It is used to model sediments and to simulate erosion.

Volumetric models allow terrain with overhangs, caves, and arches. There are 3 possible representations listed in the work: function representation, voxel representation, and hybrid representation. Function representation is seldom used in practice due to its complexity. In voxel representation the space is divided into a 3D regular grid, where each cell is assigned a material. The disadvantages of voxel representation are high memory cost and difficult modeling of gentle slopes. We see the hybrid representation as a combination of voxel and layered representations. It divides the terrain into columns and stores the materials in the columns as intervals of constant material. On the surface, the terrain is smoothed by a convolution.

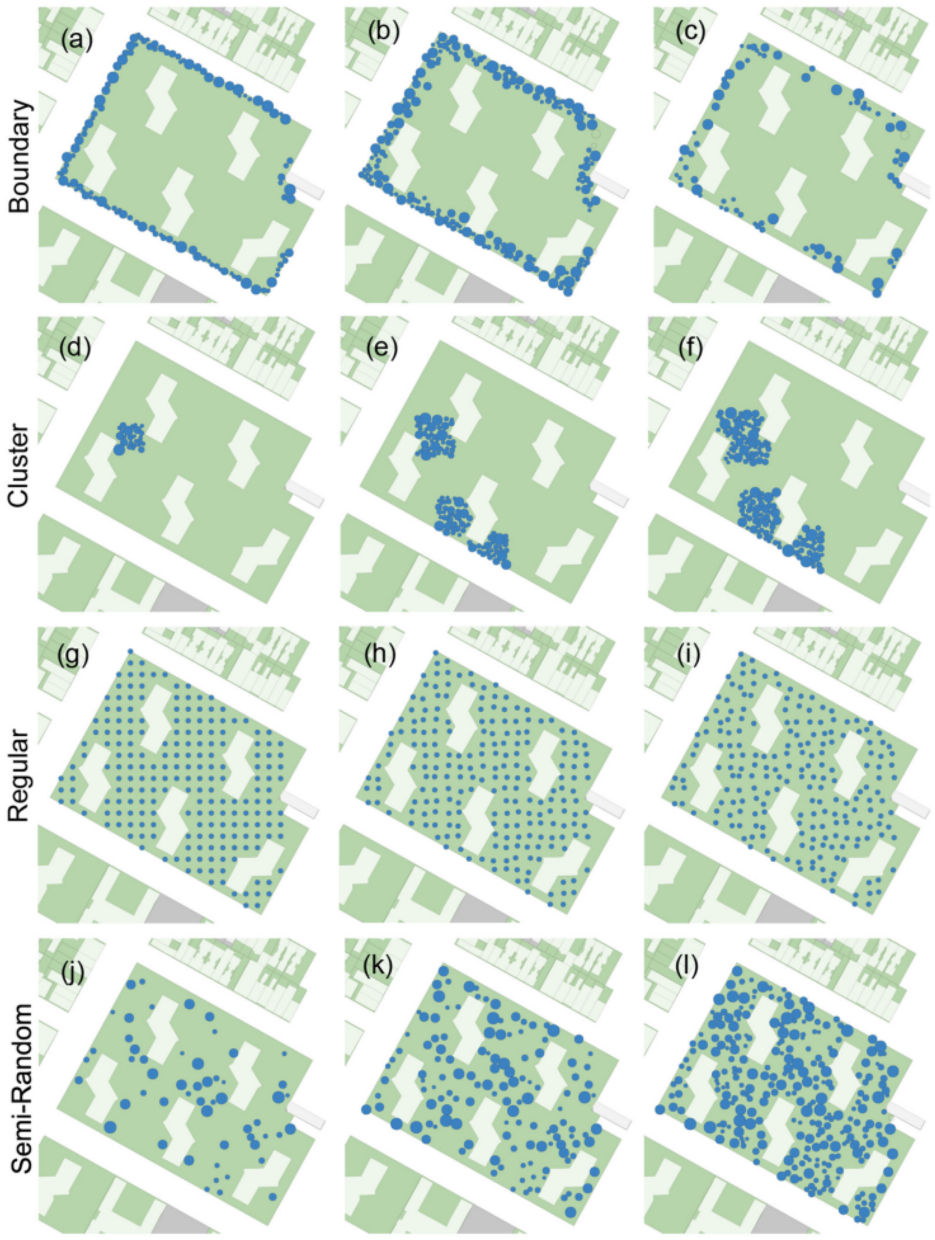


Figure 2.3: Vegetation placement strategies. Source: Niese et al. [4]

In the paper by Galin et al. procedural generation refers to the creation of terrain without the simulation of physical processes or the incorporation of real-world data.

Subdivision schemes iteratively add more and more detail to input terrain. Faulting introduces many faults with different angles to a flat terrain and displaces the terrain in one direction from the fault up and in the other direction down. Noise functions can be used to generate infinite terrains. Using multiple noises with differing scales and amplitudes, one can construct

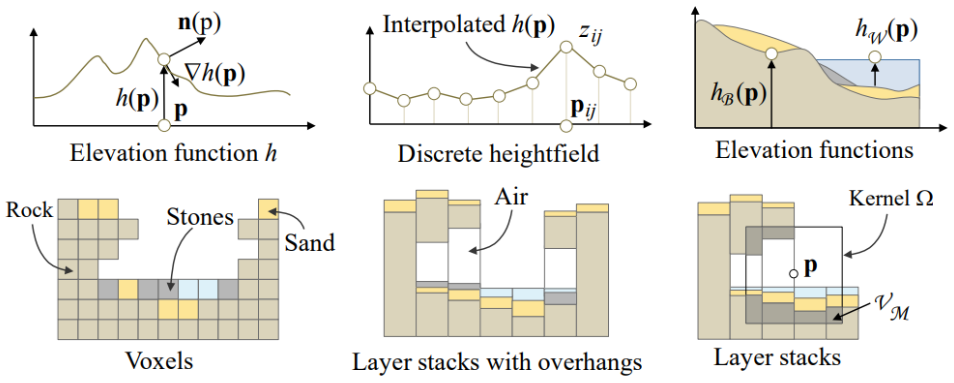


Figure 2.4: Types of terrain representation. Source: Galin et al. [9]

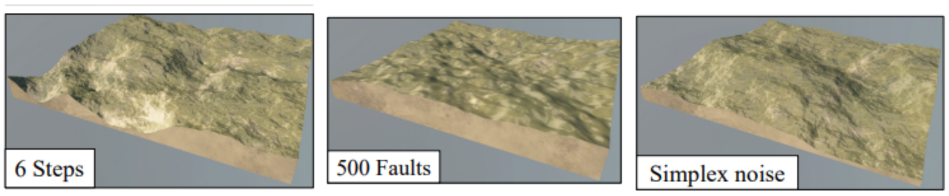


Figure 2.5: Subdivision scheme, Faulting, and Noise. Source: Galin et al. [9]

terrain that looks real. When adding additional noises, the new noise usually has a higher frequency and a lower amplitude. By this procedure, we can create a fractional Brownian motion, which is also referred to as turbulence.

In our work, we procedurally generate the terrain only partly because we use real terrain input in the form of discrete heightfields, and we increase its detail using interpolation and random displacement.

2.9 Modelling plant life

Pirk et al. [10] describe the different modules that make up a plant, how branches tend to grow towards light and bend towards it. They grow against gravity. They tend to avoid each other. They compete for space and resources. From the view of an ecosystem, the plants are modules and they too compete for resources. The authors also show how plants are pruned near buildings, and how they bend towards light when near a building. Wind also affects the shape of the plant. A branch can break when the acting forces exceed some threshold. They describe procedurally generating plants using a procedural model with some parameters such as growth rate, gravitropism, and phototropism. They use a fitness function to maximize the similarity of an input plant and the generated plant. In the end, they explore user assisted plant modeling that combines L-systems and user-provided sketches.

2.10 Overview of procedural content generation

Smith et al. [1] says that PCG is is in its essence a data compression method, it can generate vast environments and it helps with replayability and adaptability. A small team of game developers can avoid hand-crafting a world when they use PCG. Content can be generated by running a **simulation** on some initial content, or the environment can be **constructed** by using some pre-made pieces, or by using **grammars**, or by **optimization** - searching for some optimal combination of components utilizing some evaluation function, or by setting some constraints and then running a solver that searches for all possible solutions.

2.11 Generation of villages

Emilien et al. [11] present an original solution to procedural generation of different types of villages on rough terrain. The geometry of buildings adapts. They state that a village is defined by a road network, parcels of land, and 3D building models. They generate all of these.

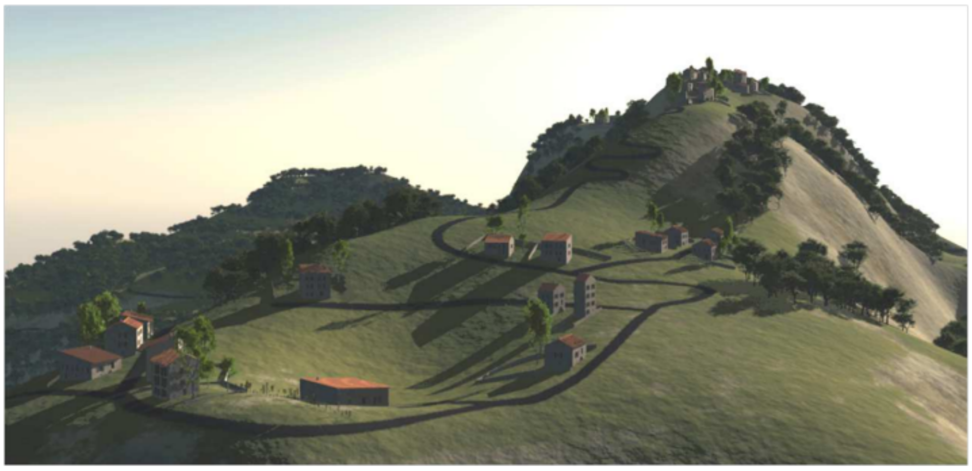


Figure 2.6: A generated highland settlement. Source: Emilien et al. [11]

They achieve this through a three-step process. First, they progressively generate houses seeds and roads that connect them, this creates a village layout. New houses appear near roads, and the road network is also extended when a new house appears. In the second step they create parcels of land using an anisotropic land conquest method - each house seed claims a part of a road, and from there iteratively conquests the land. The conquest has some cost, and each parcel of land is assigned some fund. The conquest cost cannot exceed this fund. In the third step they generate buildings adapted to the terrain using an *Open Shape Grammar* that they introduced. The authors use interest maps to determine which parts of the land are more favorable to settlers, they also consider water availability, fortifications, and other factors.

In our work we do not generate villages but some simpler form of village generation could be considered for future work.

2.12 Procedural generation of populations

Veld et al. [12] state that procedurally generated worlds are often devoid of people. They present a method for generating a socially connected population on any terrain. They praise Dwarf Fortress [45] for creating worlds with diverse population, simulating history, creating towns, populations, and relations. Their work differs from Dwarf Fortress in that it also allows the influence of designer intent. They simulate on the level of population, not individual characters. They simulate migration, collapse, and relations of populations. A large amount of characters is generated. They state that the tool is good for generating narrative and quests for RPG games. They were inspired by history simulation in Dwarf Fortress and complex relations in Crusader Kings II [46] that change during the game. For generating settlements, they got inspired by the work by Emilien et al. we mentioned before.

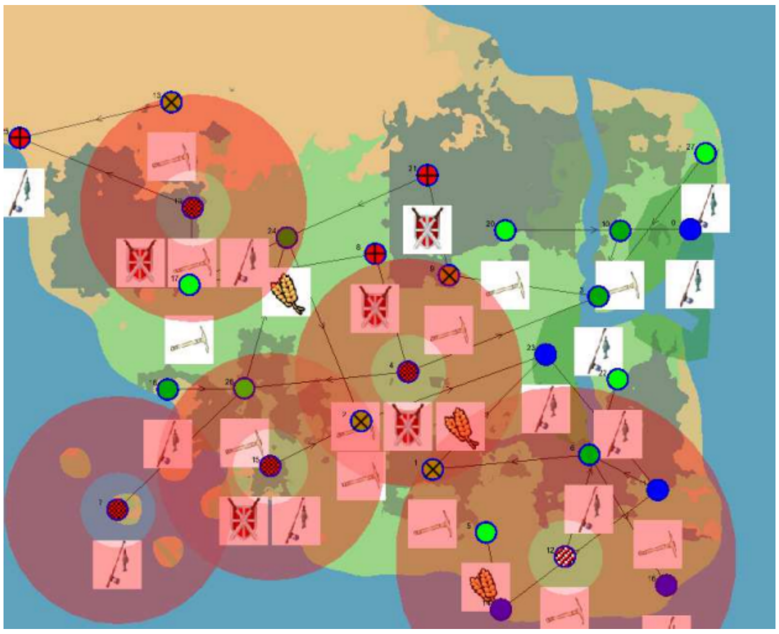


Figure 2.7: A generated world with population. We can see mining settlements, farming settlements, military settlements, fishing settlements, and relatios. Source: Veld et al. [12]

The input of the method is a landscape, and the output is characters within a population. Settlements consists of districts, districts have needs for resources. A settlement has the needs of its districts. A character resides in a district and has relations with characters and settlements. The district generates products, has needs for products, and allows its parent settlement to use some types of relations. A product is a resource such as fish or wood. These are present if water is near or woods are nearby. A relation applies to a set of settlements and has a preferred distance and a maximum distance. Relations between settlements are a strong basis for relations between individual characters. Using relations, settlements exchange resources. The simulation of the world is done in iterations.

They also introduced village prototypes. Using this, a designer can, for example, encourage a village to be a fishing village by setting a rule that its villagers will be more effective at fishing. As a result, the village prefers to be close to the sea. Or the village prototype could be a farmer village, etc.

2.13 Generating plants maximizing vegetation cover

Gasch et al. [24] present a procedural modeling method for placing plants in an environment. They attempt to maximize vegetation cover while being subject to a set of constraints such as terrain characteristics or plant species. The method has three steps.

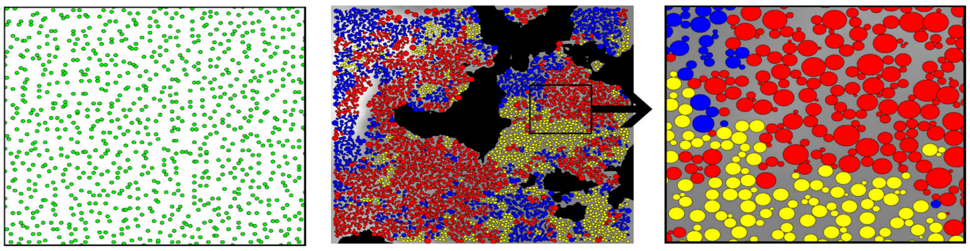


Figure 2.8: Plant distribution. From left: initial seed distribution, plant species assigned, and sizes adjusted so that the sizes are maximum. Source: Gasch et al. [24]

The first step is the distribution of seeds, where they are placed in a regular grid and then their position is offset using a uniform distribution sample, where the maximum offset is half of the spacing size in the grid. In the second step, a species is assigned to each seed. This is done first for a few seeds using the abiotic features of the terrain. The species of the remaining seeds is determined by evaluating the influence of the species of the neighboring seeds. This is done using a k-nearest-neighbor classifier with $k = 3$. The abiotic features the authors considered include only the slope and height of the terrain. And third, constraints are applied. They use a set of inequalities that represent different constraints of the individual plants. The inequalities are solved using linear programming. Using this, they obtain the maximum size of each plant within the constraints, maximizing the vegetation cover. Each plant has its circle with the same size as the plant, and the circles of different plants cannot intersect. The circle of a plant has both maximum and minimum possible sizes depending on the plant species.

In our work we also work with minimum and maximum plant sizes; the Variable-Radii Poisson-Disc Sampling method also generates plant positions with circles with variable radii that do not overlap.

2.14 Dynamic generation of terrain chunks

Nahodil P. [27] presents a system for the procedural generation of a 3D environment along a train track. The terrain is meant to dynamically generate and unload as the train travels through the environment. To achieve this, he divides the environment into chunks. The chunks to be rendered are put into a priority queue and ordered based on their distance from the camera.

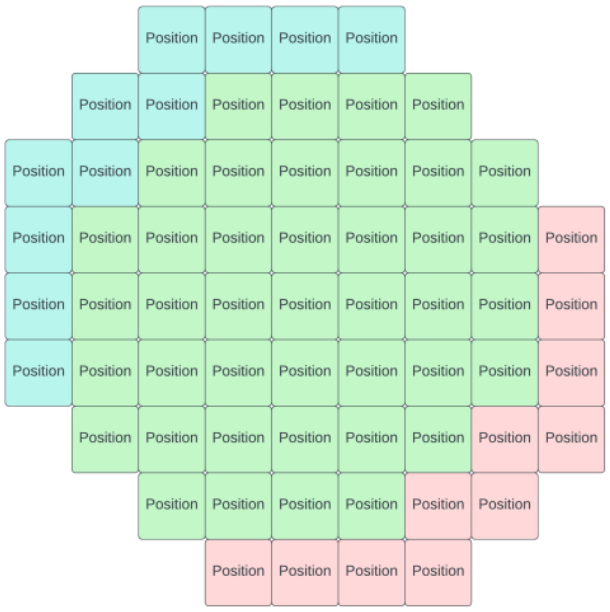


Figure 2.9: Chunks in a circle around the camera. Light green are the generated chunks, dark green are the chunks that are being generated, and red are the chunks that are too far away and are being destroyed. Source: Nahodil P. [27]

We used this work as inspiration when creating our chunk management system. But our chunks are being generated and unloaded in a circle around the player character, not around the camera.

2.15 Terrain adjustment under roads

Kyzr O. [28] presents a tool for Unity that allows procedural generation of outdoor scenes. The scenes can contain roads, paths, erosion, water bodies, and rivers. The user can choose what he wants to include. The terrain is generated using a noise function, and erosion is applied. The environment is divided into chunks, and the generation result is saved into a texture. The computation is done using a compute shader and the texture size on highest settings is 4096x4096 pixels. The texture dictates the height of the terrain. The roads generated by the tool follow the control points and can carve through terrain or make it higher.

We used the way the roads affect the terrain, carve into it or make it higher, as an inspiration for our work.

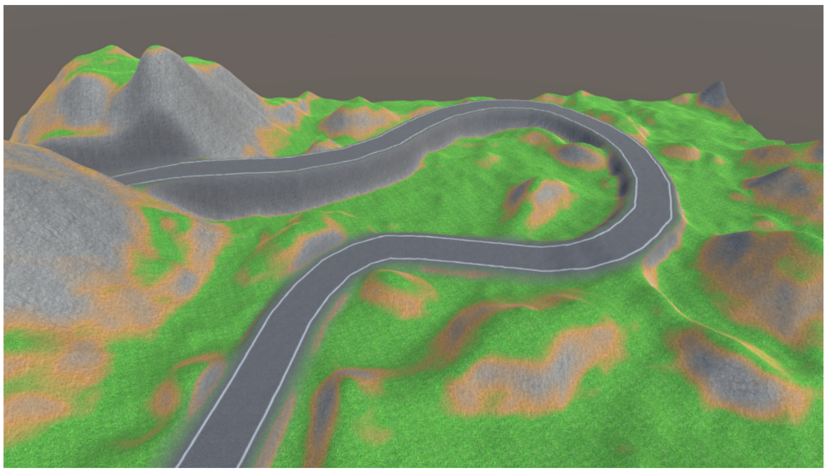


Figure 2.10: A generated road where the surrounding terrain is adjusted, either moved up or down. Source: Kyzr O. [12]

Chapter 3
Procedural generation of biomes

In this chapter, we discuss our approach. It includes dividing the world into chunks, creating terrain, creating streets and buildings, dividing the world into areas that are assigned biomes later, creating vegetation, and creating quest items.

3.1 Design

First, we discuss the general design of our system for procedural generation of environment to provide an overview of what we are doing and how. We will discuss the details in the implementation section.

Because we want our geolocated environment to cover the whole surface of Earth, we split the environment into parts called "**chunks**" and we generate in real-time only those chunks that are close to the user. We also want the generation to be stable, so that the environment is the same every time it is generated for a given GPS location. That is useful because different users will see the same generated environment in the same location, and it also helps us to make connections between chunks seamless.

For creating terrain, we use real world elevation data of the terrain in the form of a discrete height-field representation that is the most used for terrain [9]. From that we create a terrain mesh.

For creating street and building models, we again use real-world data. In the input data, the streets are represented as polylines and the buildings as polygons. We create the streets directly from the polylines, adding appropriate width, and we create the buildings by triangulation of the polygons and extrusion upwards.

We want the biomes to be spread uniformly in the environment in order to keep all biome types at a reasonable distance from the user. This is done to aid the gameplay, when, for example, the player needs to gather a resource from a specific biome. We divide the environment into areas using a Voronoi diagram. To create the diagram, we create its input points. We call them *biome centers*. We generate these points in a regular grid and uniformly randomize their positions within their grid cells. The result of this are areas with visibly straight edges. We can fix this by distorting the edges using a noise function [13] to achieve a more natural look.

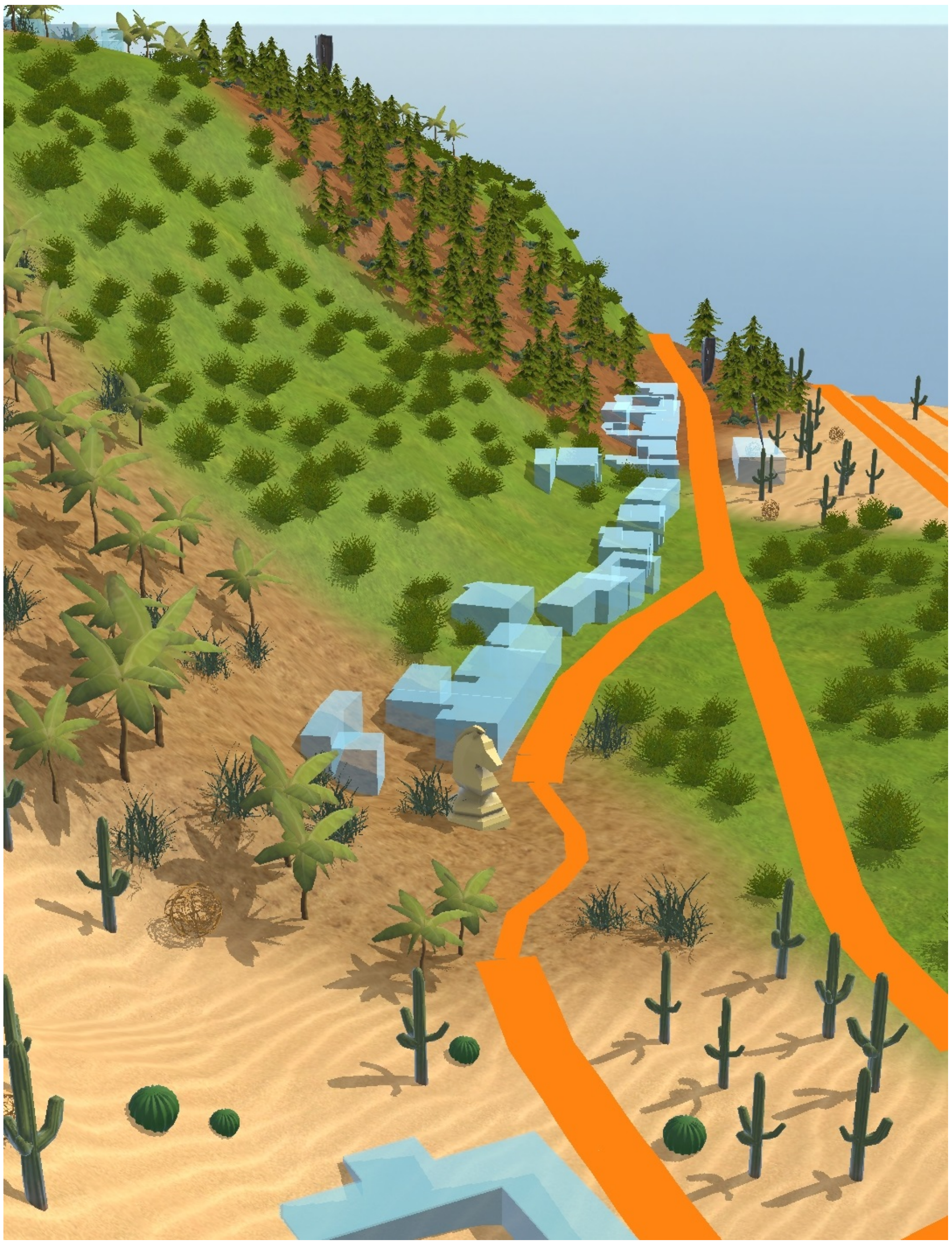


Figure 3.1: Example of our generated environment. In the image we can see orange streets, blue buildings, the knight chess piece represents the player character. We implemented 4 different biomes and the system could easily work with more in the future. The closest biome to the camera is the Desert, behind it is the Oasis, next is Plains and behind it Forest.

Using noise functions we generate two variables for every biome center - temperature and moisture. Each of the biomes that can be generated has some value range of these variables assigned. Using the variables we assign a biome to each biome center and to its area. We can determine the biome of any point by comparing the distances (which we distort using a noise function so that we have a distorted Voronoi diagram similar to the one shown in figure 3.2) to nearby biome centers. The biome of the point is then the same as the biome of the closest biome center.

The terrain is represented by a mesh, the mesh is divided into rectangular cells, and each cell is made of two triangles. The biome of a cell is determined by finding the biome of a middle point of the cell.

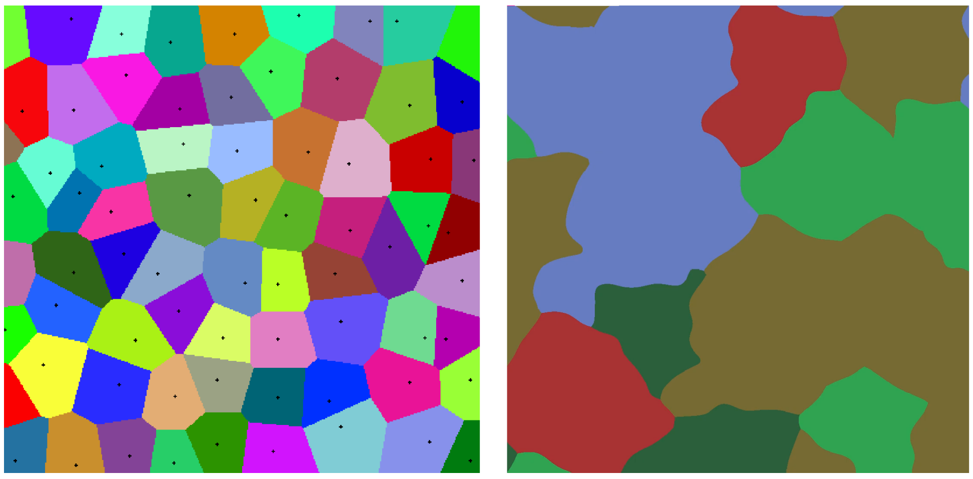


Figure 3.2: A Voronoi diagram with input points visible and a Voronoi diagram distorted using a noise function. Source: Stadnik V. [25], Rudi dev [13]

For generating positions for vegetation placement in the chunk we got inspired by the Semi-Random placement strategy, shown in figure 2.3 and used by Niese et al. [4] and we also use the Variable Radii Poisson-Disc Sampling they used. It creates a set of points where every point is the center of a circle with a random radius, and these circles are not allowed to intersect.

Not all of the generated points are used for creating plants, some are discarded. Which points are discarded is random, it is affected by settings, by a noise function that creates meadows, and also by the biome of the point because each biome has its own vegetation density value. In addition, each biome has a set of plants that can grow in it and every plant is assigned some probability of growing in the biome.

Each terrain cell is assigned a biome texture based on its biome. The textures of neighboring cells are blended together in a shader.

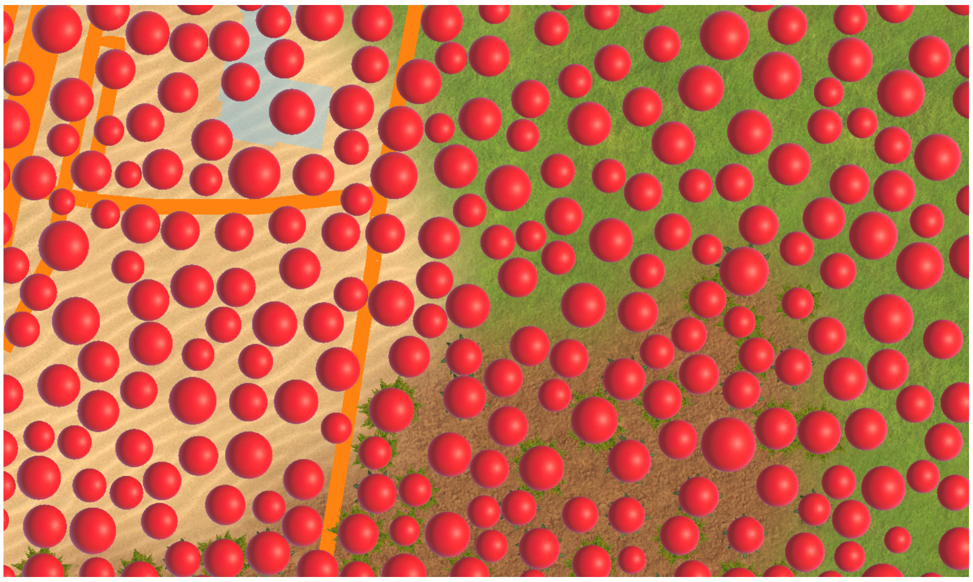


Figure 3.3: Visualization of the result of our implementation of the Variable Radii Poisson-Disc Sampling. The radii of the red spheres correspond to the radii of the circles used in the sampling.

3.2 Implementation

In this chapter, we discuss the implementation of the biome generation system in more detail. The git repository with the source code can be found at `https://gitlab.fel.cvut.cz/mrazmic7/diploma-thesis-public`.

3.2.1 Division of Earth surface into chunks

We divide the surface of the Earth using the Geohash [26] system. Using this system, parts of the surface are assigned ids. The way it works is simple and is explained in an article [20]. The surface of Earth is represented as a rectangle and we recursively divide it in half. With each division, we write into the id number a zero or a one. We use integer identifiers at zoom level 32. We use an NGeoHash [35] package to get the id of a point's area or to get the neighbors of an area.

3.2.2 Terrain

We obtain the input data with the elevation of the terrain in a discrete height field representation that Galin et al. [9] states is the most common representation of the terrain. From that we create a triangle mesh. To make the terrain more detailed, we create additional vertices by interpolating existing vertices and adding a random height offset.

To get elevation data of chunk terrain we use Google Elevation API [8]. The API takes as input the latitude and longitude of a set of points and returns their elevations. There is a maximum of 500 points that can be in

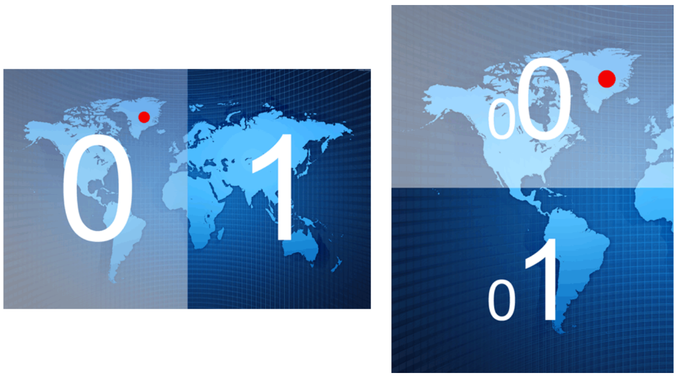


Figure 3.4: Integer Geohash area id creation visualized. Source: Whelan P. [20]

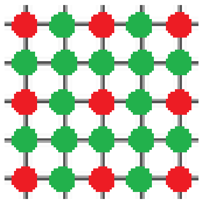


Figure 3.5: Visualization of adding new vertices to terrain. The red points are the input points, the green points are the interpolated points.

one request and the spatial resolution of the SRTM dataset [34] (the API uses this dataset as a source of its elevation data) is about 30 meters. So in a chunk we create a grid of 22 times 22 points with spaces of about 30 m between and request the elevation of these points from the API. We use the returned elevation data to create a terrain mesh.

3.2.3 Converting between GPS and Unity coordinates

For the conversion we defined two important GPS points - one we call the "reference origin" and the other the "world origin". When converting, we use the double data type in our code and convert using the reference point. From the result we subtract the world origin and convert it to float data type. This can be then used in unity as a transform. We continually recalculate the world origin so that it is never further away than 50 km from the player. That should be a reasonable distance for not experiencing the negative effects of float data type inaccuracy.

For the conversion we assume the latitude of the reference point - that results in distortion in some parts of the world but the procedurally generated environment is continuous and stable. Let's expand on this a bit:

For the conversion we at first assume an ideal sphere with the radius of the Earth. We know that one degree of latitude corresponds to some distance in meters. But for longitude it is more difficult because it depends on the latitude of the point. So we do a little hack and we assume the same latitude everywhere - the latitude of the reference point. That means that the further we get from the latitude (or negative latitude) of the reference point, the more inaccurate the conversion becomes. But the conversion still works, the distances are just distorted. That means we assume a **cylinder** where the latitude is mapped accurately to meters and the longitude to meters is mapped accurately only at the latitude and negative latitude of the reference point. As we get closer to the equator, the longitude meters become more squashed than in reality and as we get closer to the south and north poles, the longitude meters become more stretched than in reality.

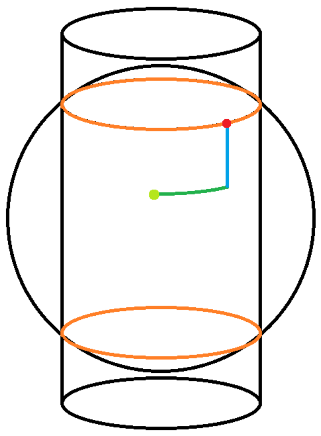


Figure 3.6: Visualization of how our GPS to Unity coordinates conversion works. The red point is the reference point, the green point is the one we are converting. The green line is a part of a latitude circle, the blue line is a part of the longitude circle projected onto a cylinder. We move on the green line using longitude or x coordinate and on the blue line using latitude or z coordinate. The orange circles are the latitude circles at the latitude of the reference point.

3.2.4 Streets and buildings

We get the input data for building and street generation as a vector description, a building is represented by a polygon made of points, and a street is represented by a polyline.

We download OpenStreetMap (OSM) data from `http://overpass-api.de`. Into the request we add the GPS boundaries of the area in which we want to download the OSM data. We download the data in XML format. In these data, we care only about the Node, Way, and Relation XML nodes. Nodes are points with latitude and longitude information attached and ways reference these nodes by their IDs. We extract the ways that represent streets and buildings. We use the Relation nodes that represent multipolygon building, and from that we use only the outer way. This is a quick solution, it could be improved in the future. For example, support for making holes inside the polygon using inner ways of the Relation node could be added.

The way that represents a street contains node IDs in order. We render the streets using the LineRenderer Unity component. We differentiate the width of the created street based on what kind of street it is, because there are several kinds of streets in OSM (there they are called highways). We divide the street into smaller segments and place the endpoints of the segments onto the terrain using physics raycasting.

The way that represents a building contains IDs of nodes of the buildings outline, these can unfortunately be written in the OSM in clockwise or counterclockwise order. We detect in which order they are written using an algorithm that sums angles at each node of the polygon and based on the final sum we decide. Then we triangulate the polygonal surface using the ear clipping triangulation algorithm. We also create walls of the buildings by extruding the outline. That is done by creating additional triangles and points that are then inputted into a MeshFilter Unity component. The resulting GameObject of the building then contains two objects, one contains the walls MeshFilter and the other the roof MeshFilter. We then put the buildings onto the terrain. For every point of the building outline we check using a Unity physics raycast at what height it intersects with the terrain mesh. We take the values of the lowest height at put the building at that height. We assign random colors to the buildings.

We create building mesh from a polygon describing its layout using an Ear clipping algorithm [19] and we create its walls by extruding its layout upwards. The building is placed onto the terrain again using physics raycasting, the height of the lowest point of the layout is used.

3.2.5 Biomes

We generate points in a regular grid, slightly randomize their positions and use these new points as input to create a Voronoi diagram. We call these points "**biome centers**." The result of this is areas with visibly straight edges. We can fix this using Perlin noise [13] to achieve a more natural look. In every chunk we generate 4 biome centers, each positioned in one quarter of the chunk. To achieve continuity of biomes across chunk borders, a chunk also generates biome centers of its neighbor chunks. We determined that generating two additional biome centers in a direction is enough, but that means to generate all four biome centers of every neighbor chunk anyway.

We achieve stable randomization of the biome center position by setting a seed for the random generator beforehand. The seed is based on the latitude and longitude of the initial biome center position. When creating the seed, we scale latitude and longitude differently so that biome centers, that are close to each other, are not offset similarly.

To a biome center we assign a biome. Each biome type has some temperature and moisture range assigned. Temperature and moisture values of a point are generated using two Perlin noise functions. As input to the Perlin noise, we use latitude and longitude, which we also scale. Experimentally, we arrive at the scaling values we used. We scale the input for temperature by 100 and the input for humidity by 200. Using the resultant temperature and moisture we assign a biome to the biome center.

We determine the biome of a point using distances to the nearest biome centers. The distances are additionaly distorted using Perlin noise. Biome of the point is the same as the biome of the nearest biome center. We combine

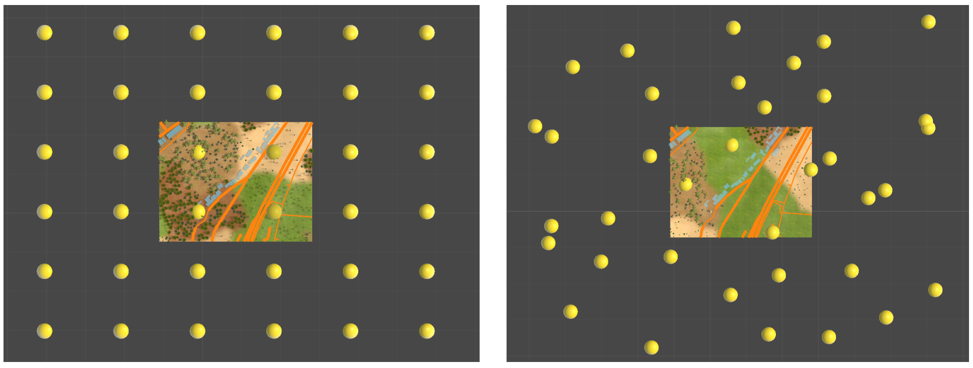


Figure 3.7: Biome centers, represented using yellow balls, before and after being randomly offset. We can see one chunk and its four biome centers. This chunk also generates biomes centers of its neigbors so it can independently generate its biomes without the other chunk having to be loaded.

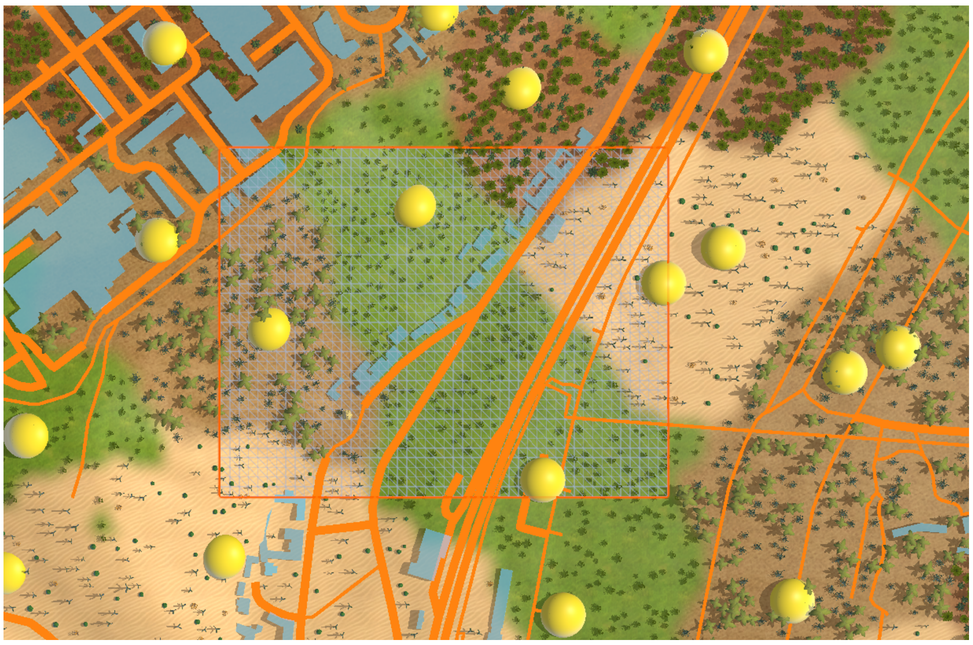


Figure 3.8: Biome centers visualized using yellow balls. The chunk shown in previous figures is now selected, its borders are orange and its triangles and cells (made of 2 triangles) are visible.

bigger and smaller offset. For creation of each offset we use two perlin noises. For the first offset we scale the noises by value 1150, for the second offset by value 2300. We arrived at these values experimentally. We also transformed the range of the offsets from range $< 0, 1 >$ to range $< -1, 1 >$. Before applying the offsets to distances, we scale them by terrain cell size and the bigger offset by number 5 and the smaller offset by number 2.5. In the following code you can again see how we apply the values we got using Perlin noise.

```
Vector3 movePointForColoringByBiomes(Vector3 point)
```

```
{
    GpsVector gps = Gps.instance.ConvertUnityCoordsToGps(point);
    Vector2 cellSizes = new Vector2(cellWidthInX, cellHeightInZ);

    int seedX, seedZ;
    float gpsScale, offsetsScale;
    Vector2 randomAddition = Vector2.zero;

    seedX = 723;
    seedZ = 189;
    gpsScale = 1150.0f;
    Vector2 biggerOffsets = generateOffsetUsingPerlinNoise(gps, seedX, seedZ, gp

    seedX = 500;
    seedZ = 1000;
    gpsScale = gpsScale*2;
    Vector2 smallerOffsets = generateOffsetUsingPerlinNoise(gps, seedX, seedZ, g

    offsetsScale = 5.0f;
    randomAddition = biggerOffsets * cellSizes * offsetsScale;

    offsetsScale = offsetsScale /2;
    randomAddition += smallerOffsets * cellSizes * offsetsScale;

    point += new Vector3(randomAddition.x, 0, randomAddition.y);

    return point;
}
```

3.2.6 Generating vegetation

For generating potential positions for vegetation placement in the chunk we got inspired by the Semi-Random placement strategy used by Niese et al. [4] and we also use the Variable Radii Poisson-Disc Sampling. It creates a set of points where every point is the center of a circle with a random radius, and these circles are not allowed to intersect. This approach creates a natural-looking distribution of points. For the implementation we got inspired by Lague S. [36].

We accelerate the placing of vegetation by creating a grid of cells where each side of a cell is equal to square root of minimum possible circle radius and thus in every cell there can be at maximum one plant. Sadly the circles can overlap on chunk borders but it is not very noticeable. This could be fixed in the future so that placing trees is seamless across the borders.

Some of the generated points are used for creating vegetation. Which points are selected to use depends on the vegetation density parameter, lushness parameter (both set in the application settings) and on the point's biome. Point's biome is determined by which biome center is the closest to it, but

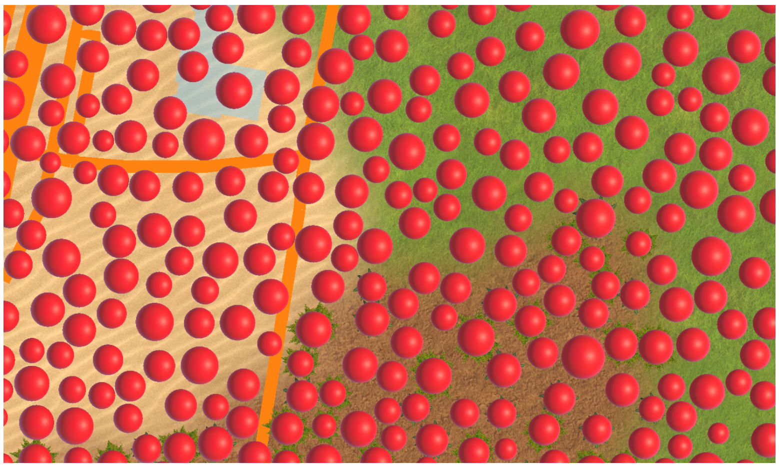


Figure 3.9: Visualization of the result of our implementation of the Variable Radii Poisson-Disc Sampling. The radii of the red spheres correspond to the radii of the circles used in the sampling.

the distances are slightly distorted using a Perlin noise. The same strategy is used also for determining the biome of terrain cells. Most biomes have more than one plant species that can be generated in it, each species has a set probability. Lushness is used to create "meadows" - we use a Perlin noise to select areas where there is no vegetation generated. The lushness parameter determines the size of these areas.

3.2.7 Terrain textures

We logically divide the terrain into cells, where a cell consists of 2 triangles. When determining the biome of a cell, we measure the distance of the middle of the cell from nearby biome centers and we slightly distort the distances the same way as before.

To achieve continuity of biomes between neighboring chunks, each chunk generates also some biome centers of its neighboring chunks, 2 biome centers in every direction. By visualizing how big the area that a biome center can influence is, we arrived at the number 2.

We programmed a surface shader for the terrain that blends the biomes of its cells. The texture of a terrain pixel is influenced by biome textures of cells that are within a distance of 2 cells from the pixel. To achieve continuity of cell biome texture blending between chunks, each chunk has to generate some additional cells from its neighbor chunks. That means that each chunk generates 2 cells beyond the chunk borders in every direction. We use a surface shader so that shadows can be displayed on the terrain. When programming the shader we got inspired by Beider V. [37].

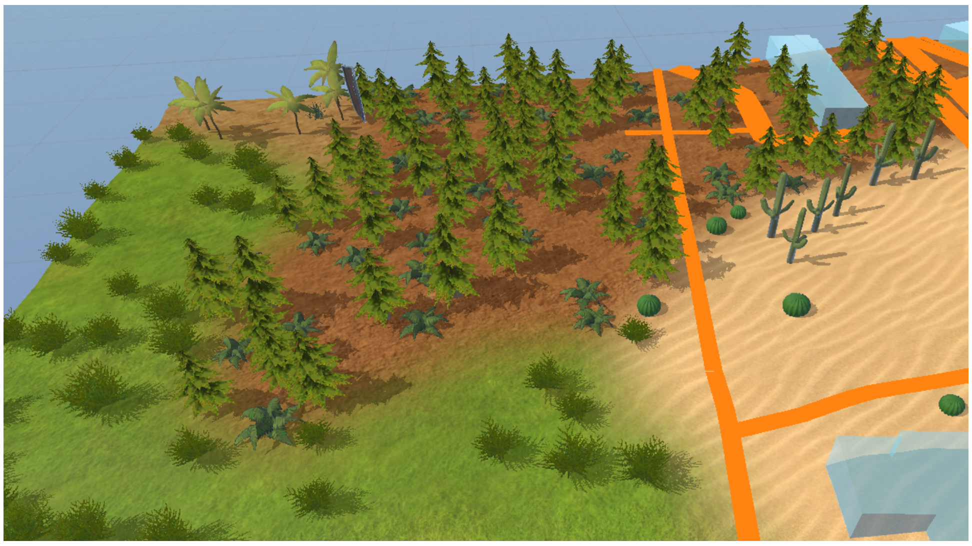


Figure 3.10: Example of terrain with blending biome textures.

3.2.8 Terrain smoothing under streets

To smooth the terrain under streets we got inspired by Kyzr [28], where he describes road generation on terrain. The road is able to cut through terrain or to tighten it. In our approach, we choose a smoothing radius and go through every vertex of the street, and check the height of every terrain vertex within the smoothing radius of a street vertex. We average these heights and level the terrain vertices within the smoothing radius around the street vertex by interpolating the old heights and the average height using a factor alpha. This factor is determined using a smoothstep function - it takes the distance of a terrain vertex from the street vertex (with maximum distance being the smoothing radius) and converts it to range from 0 to 1, where the values around 0 and 1 are smoothed.

3.2.9 Chunk terrain normals stitching

The terrain was visibly not continuous on chunk borders before. Every terrain vertex has a normal. On chunk borders there are multiple vertices in the same position, but each of them has different normal. We fixed this by the following method: When a chunk is created, it checks what neighboring chunks exist around it. Then it gets the normals of vertices that are on mutual borders and averages all of the normals of vertices that share a position. Than it saves the averaged normals to every affected chunk.

Chapter 4
Game prototype

The player character changes its in-game location according to the change of the device's GPS location and rotates based on data received from the device's compass. The movement of player is interpolated to eliminate stutters that were present when we used the sensor data directly. The camera controls are adapted for usage on a touchscreen, swiping rotates the view around the player character and pinching zooms the view.

4.1 Using and simulation GPS data

We get the device's current GPS sensor readings through Unity. The position changes quite frequently and there can be big jumps. We create our own GPS position called game GPS and we get it by interpolating between the old game GPS and the current sensor readings so that the movement is smoother. We also added a threshold, if the position changes by more than a few hundred meters, the game GPS changes immediately to the new position. Player orientation is changed based on the data received from the device's compass. It is also interpolated to prevent unwanted twitching. We are essentially "bridging" the device GPS and in the game is used the game GPS.

We can also change the source of the GPS data to a simulated GPS, where we control what location is simulated and we can smoothly change it using a virtual joystick. The options for enabling and changing the simulated GPS are available through a settings window in the prototype. When using the simulated GPS, the game GPS is determined by the simulated GPS instead of by the GPS read from the device sensor.

The player position is the same as the game GPS all the time.

4.2 Chunk generating and unloading

We set a chunk-generating distance, which is a radius of the area around the player in which to generate chunks. Each time we generate the chunk that is closest to the player and currently there is a 1 second delay set before generating another chunk. We set this delay so that the prototype doesn't freeze for too long at a time. We find the candidate chunks for generating by

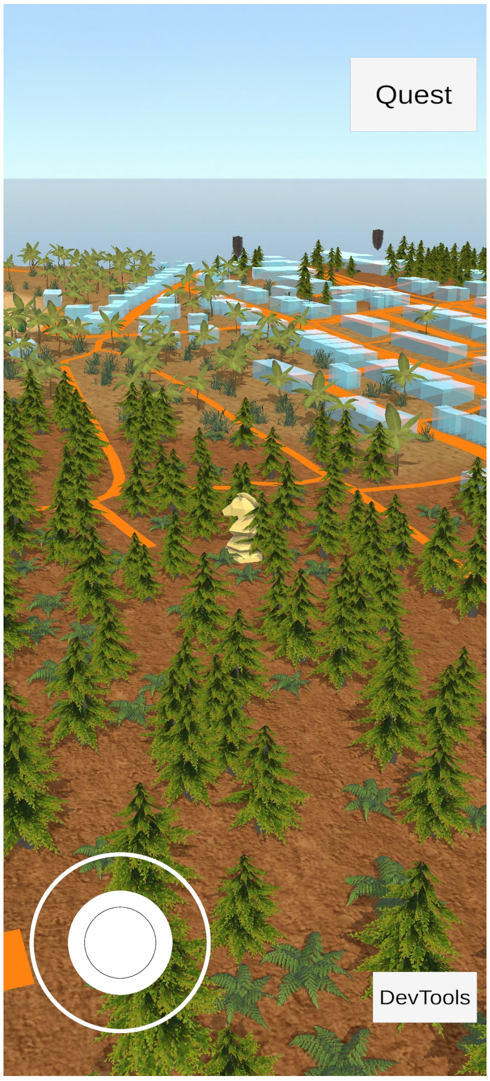


Figure 4.1: Joystick UI and the player character in a forest biome.

recursively looking at geohash neighbors of another chunk. When an existing chunk is too far away from the player (chunk-generating distance we set + 200 meters), the chunk is destroyed to free up resources.

4.3 Camera control

The camera controls are designed for the touch screen, so the user can rotate the camera by swiping the screen with one finger and zooming in or out by pinching in or out with two fingers. There are limits to the angle of the camera so the user can't rotate the view to look straight down or straight up. If we detect that the camera is under terrain or slightly above it, the camera is pushed upwards.

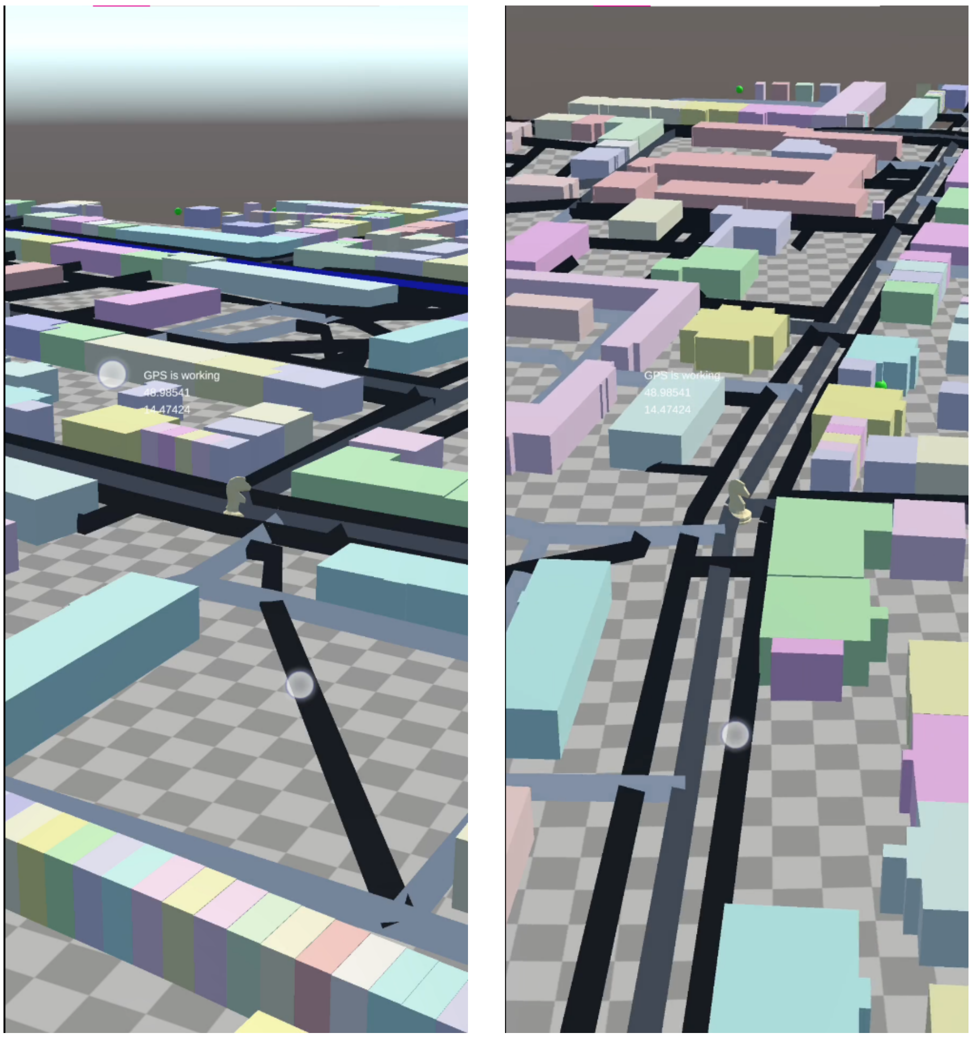
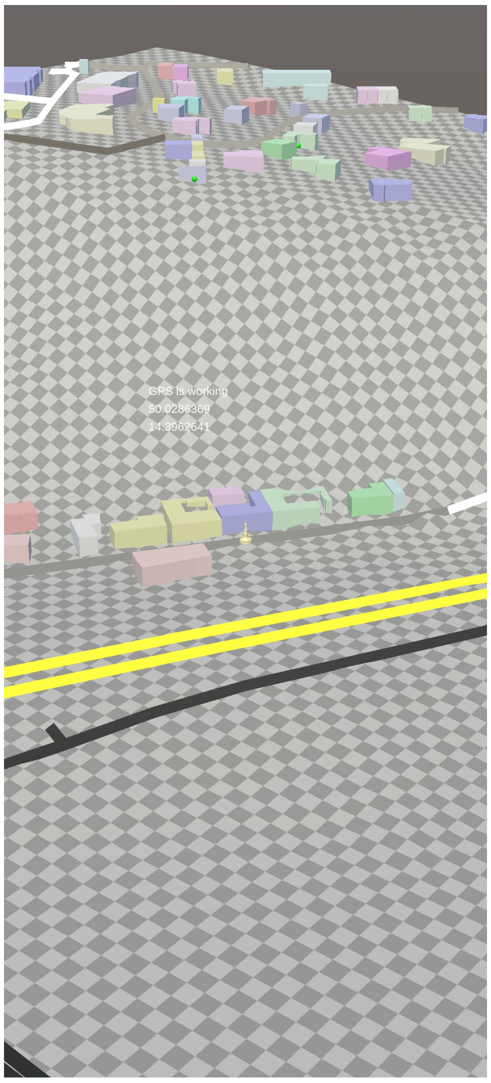


Figure 4.2: Screenshots from early prototype. The white circles are the places where the user touches the screen. (Left) The user is zooming in the view using a two finger gesture. (Right) The user is rotating the view using a one finger gesture.

4.4 Quest

To be able to test the prototype with users, we designed a quest. Into the desert and forest biomes we randomly place the sword and shield items and the player's task is to find them. The quest is accessible through a button in the prototype, a quest window appears with the initial quest description and the player has the option to accept the quest. The text in the quest window changes based on what stage the quest is currently in. There are 3 texts - One is the initial description, the second is when the quest is accepted but not yet completed and the third is after the quest was completed. After the player accepts the quest he is supposed to start walking around, search for



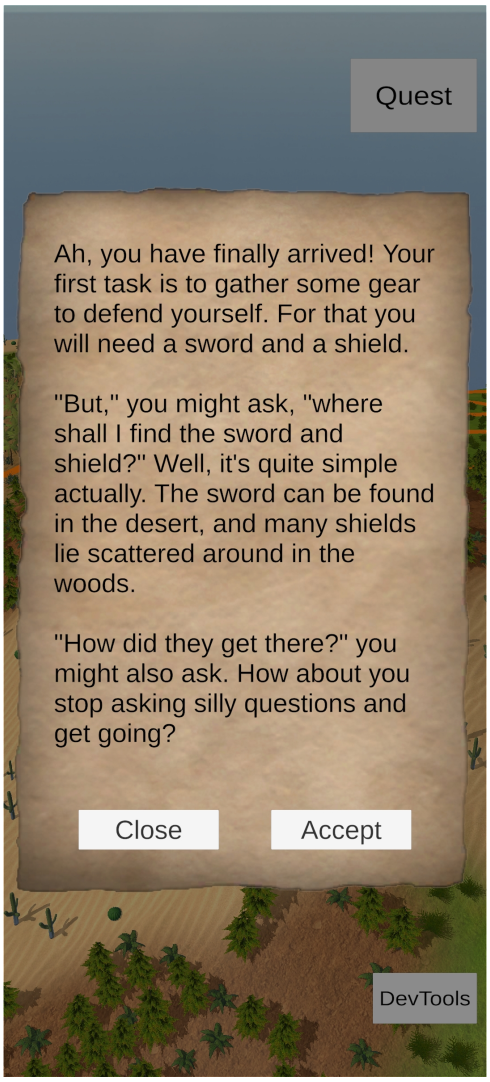
```
GPS is working
50.0286369
14.3962641
```

Figure 4.3: Screenshot from early prototype. Here we can see terrain, buildings and streets without biomes. The buildings are placed on top of the terrain.

the two biomes and there he should look for the items requested. If he clicks on the item and is within 100 meters distance of it, the item is picked up. Otherwise a message appears that says he is too far away. There is a UI that tracks the quest objectives, it tells the player the name of the quest, what items is he supposed to find and how many items he already found. After he picks up enough items, he can complete the quest.

4.5 Settings

There is a settings window accessible through the button "DevTools". In this window there are toggles to enable or disable different parts of the UI, to enable using building heights from OSM when generating building objects,



Ah, you have finally arrived! Your first task is to gather some gear to defend yourself. For that you will need a sword and a shield.

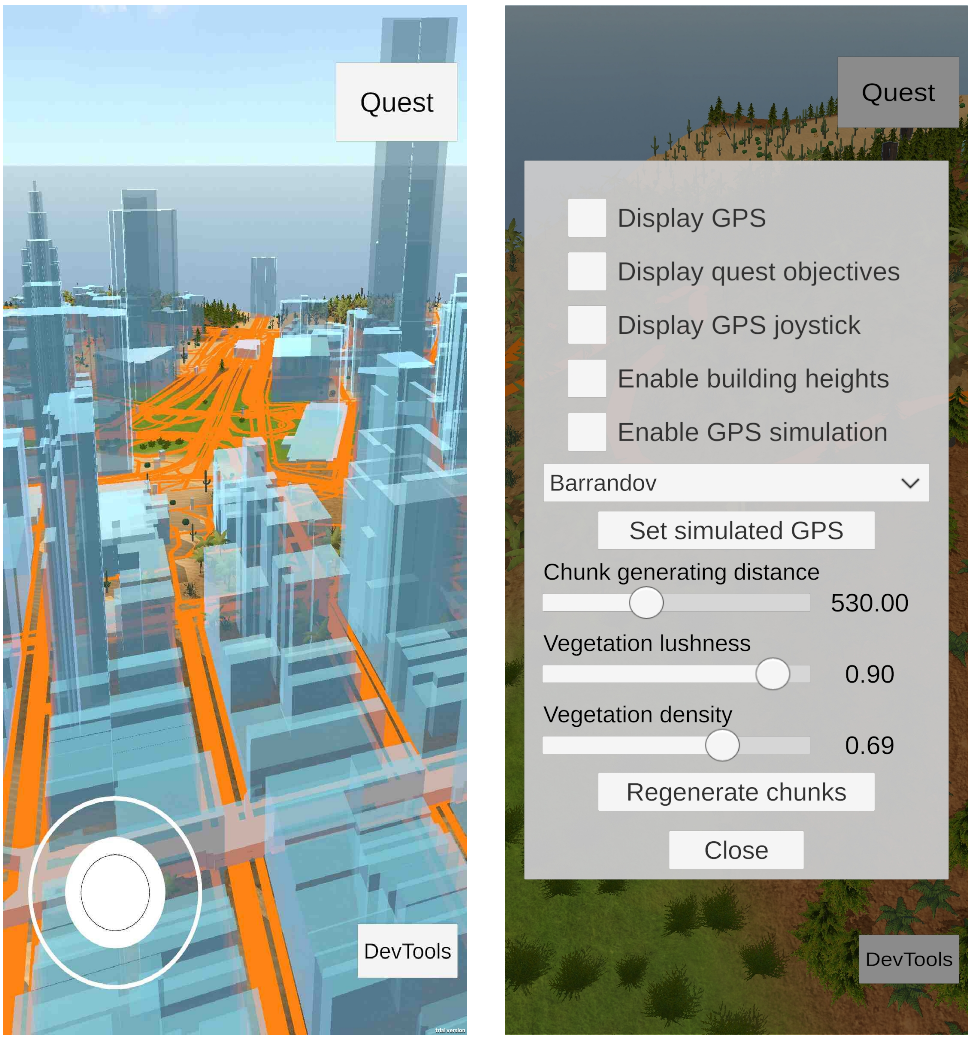
"But," you might ask, "where shall I find the sword and shield?" Well, it's quite simple actually. The sword can be found in the desert, and many shields lie scattered around in the woods.

"How did they get there?" you might also ask. How about you stop asking silly questions and get going?

Figure 4.4: Quest window

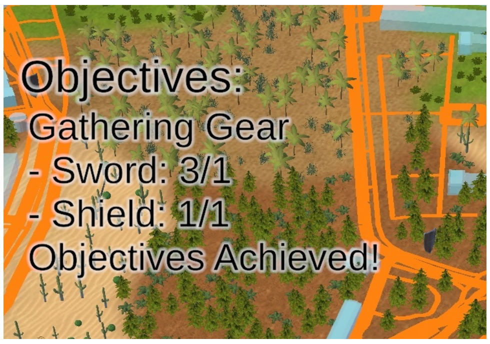
a toggle to enable GPS simulation, a selection of positions to set as the simulated GPS, a slider that changes chunk generating distance parameter, a slider that changes lushness parameter, a slider that changes vegetation density parameter.

Chunk generating distance affects the distance in which we will generate chunks and unload them. Lushness affects how big the "meadows" areas where no vegetation grows are. Vegetation density affects how many of the potential vegetation positions generated by Variable Radii Poisson-Disc Sampling will be used in the end.



Display GPS
Display quest objectives
Display GPS joystick
Enable building heights
Enable GPS simulation
Barrandov
Set simulated GPS
Chunk generating distance 530.00
Vegetation lushness 0.90
Vegetation density 0.69
Regenerate chunks
Close

Figure 4.5: From left: enabled building heights, settings window.



Objectives:
Gathering Gear
- Sword: 3/1
- Shield: 1/1
Objectives Achieved!

Figure 4.6: Quest objectives UI

Chapter 5
Results

5.1 Screenshots

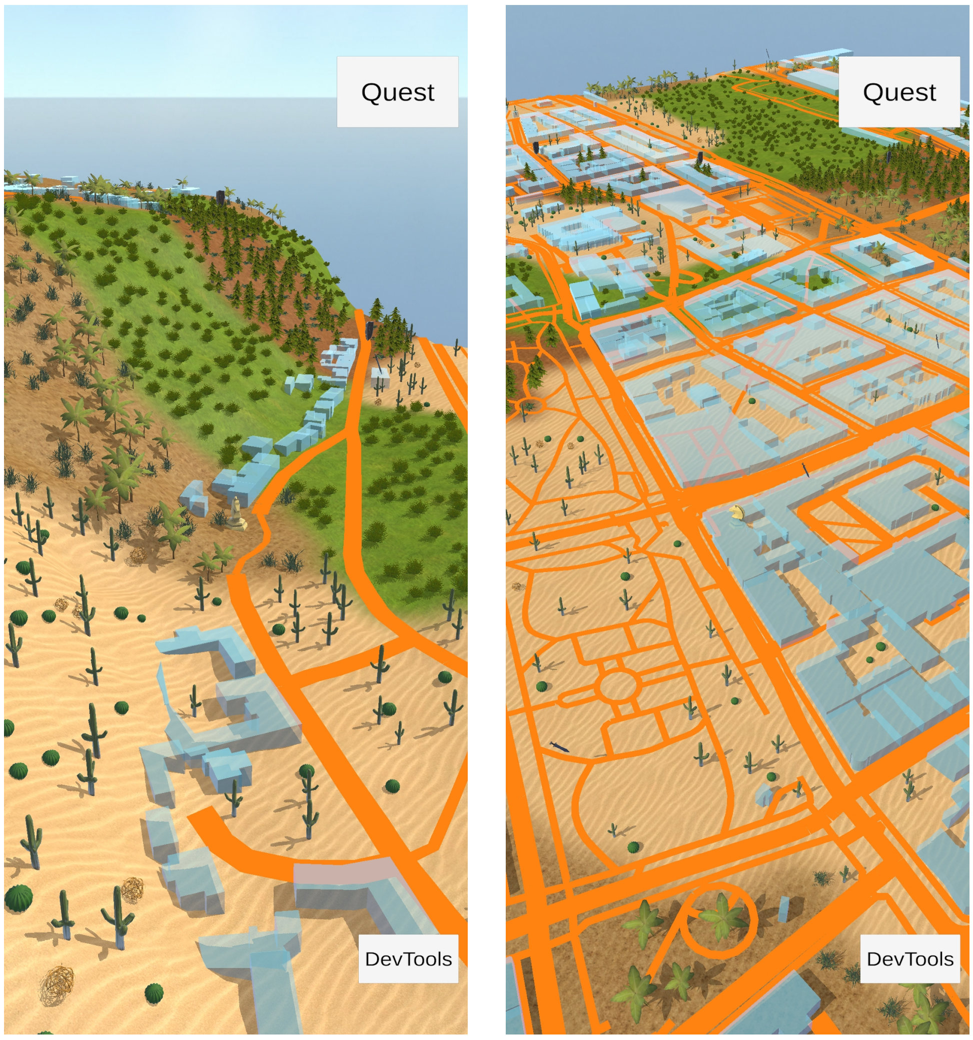


Figure 5.1: From left: Barrandov Terraces, Charles Square.

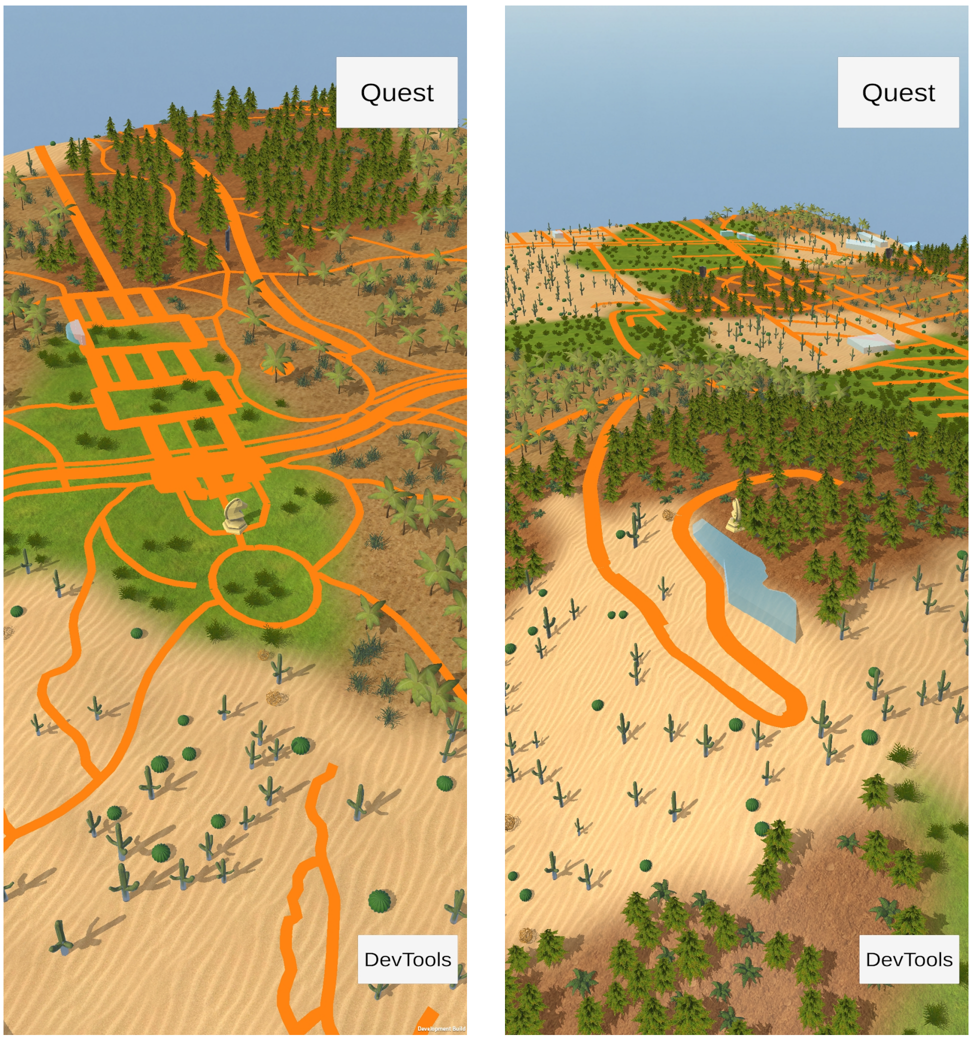


Figure 5.2: From left: Central Park, Santo Domingo.

5.2 User Testing

In total we tested the game prototype with 3 users and a few other users tried it. On the next two pages is the testing document, which contains an introduction for the prototype testing for the user, some instructions for the one conducting the test and a questionnaire. We tried to not reveal too much about the game and how it is controlled during the testing, we let the users to discover it on their own and it was mostly successful.

We will number the responses where it makes sense, the numbers correspond to the questions in the questionnaire, which can be found in the appendix.

5.2.1 Testing with the first user

1) He liked the game very much, 2) he found the controls to be fluent, the compass to be quite sensitive. 3) He would somehow highlight the nearest pickable object to be able to find it more easily. 5) He praised that the game was a good reason to walk outside and the game wasn't difficult. 7) He had difficulty finding the items, but he also says that this is probably the goal of the game, to search for the items, so he doesn't view it as a mistake. 8) He would like to how some indication of where the nearest item is, like in geocaching. 9) The biomes are used for indicating where items could be, also specific enemies could spawn in different biomes. 10) He liked the vegetation models. 11) Thanks to the 3D terrain he could predict whether he will go uphill or downhill, useful. 12) He could predict the difficulty of path he chose thanks to the 3D terrain. 13) He would maybe change the textures of streets and buildings, but on the other hand it is good that they are distinct. 14) A relaxing game that gets you outside. 15) Has experience with Pokémon GO and Geocaching. 16) For such game it is important that it is simple and there is not too many things. 17) It is good that the game gets people moving. 18) He likes discovering new places, he likes discovering new paths through areas he already knew. 19) He would like to see an adaptation of LARP (Live action role-playing game) - meeting real people, exchanging resources with them in the game, fighting them.

5.2.2 Testing with the second user

1) He found the game to be interesting, and says it has potential. 2) He liked the walking aspect because it provides exercise, he can be in nature and it provides opportunities to be social. 3) He found the environment in the game to be better to orient in than Google Maps because it is in 3D. 4) When he found the items, he felt fulfilled and happy. 5) He rates the game 8 out of 10 points, he would like to see more quests. 6) He got stuck when he was searching for a shield, he found a forest but it was behind the Vltava river and he couldn't easily get there. 7) He liked that the game is good for health, he didn't like that the game doesn't save progress. 8) He would add progression and progression saving, multiplayer. 13) He would add a snow

biome and he would make the streets black or grey, to not be so "flashy". He says that the items he is supposed to find should be well visible instead. 15) He tried Pokémon Go, it was a new style of game. 16) He says that having lots of updates and not being pay to win is important for such a game. 17) He likes that he can make new friends through the game, doesn't like that he needs to have internet to play it and it is also not suited for the disabled.

5.2.3 Testing with the third user

He would like to be able to click outside of a window to close it. He would like to be able to click a button titled "yes" when the quest asked him a question instead of clicking "close". He would like to see some introduction to the game instead of just directly being presented with the first quest, something along the lines of "Welcome to our kingdom". 1) He likes the game, he likes the visual style of it. He wasn't able to see the items in the distance and that is wrong. He made some suggestions for what to add into the game: shame some locations where you can hide your items so that you don't lose them if your character is killed. Also he suggested expanding the first quest by going to a village and asking there where you could find the items. You would have to search around in the game and ask instead of directly being handled information where to find the items. You could also find the items by chance. 2) He likes that the game makes you walk. 3) He knew where to walk, he didn't see where the character is facing. 4) He thinks the desert biome is prevalent, as a result he found the sword much more easily than the shield (which is in the forest biome). 5) He rates the game 8/10, likes the fantasy or medieval theme. 6) He got stuck when searching for the shield, the forest biome was too small. 7) He liked the appearance of the game, liked the desert biome, he didn't like that the items were not distinctive enough. 8) He would add villages, animals, enemies, the safe space for items in a pub or in a village. 9) He suggested that the vegetation could blend more between the biomes, now it is not possible that for an example a tree from a forest would grow on the edge of desert biome. 10) He rates the biomes and vegetation 9/10 points, he would make the desert less prominent. 11) 3D terrain he rated 8/10. 12) The paths helped him to orient himself, he didn't see buildings because there were none in the park where we tested. 13) He would change the appearance of the buildings, the transparent blue doesn't fit the medieval style. 14) He would describe this game as Pokémon Go in medieval times. 15) He has experience with Pokémon Go and Harry Potter: Wizards Unite from the same company. He also has experience with a game called Grim Soul: Survival. It is not a location-based game, but he mentioned some of its features that could be added to our game. 16) He says that in such a game the theme is important, also it is important to have some distant goal and a storyline. 17) He likes that the game makes you walk, what he doesn't like is that it drains the battery of your device and although you are outside, you are still on the phone. 18) In a game of this type he likes to make progress and he also likes when the game has interesting updates.

5.2.4 Testing with some other users

A user praised the text of the quests, he thinks they are very fun and that we should continue what we are doing. He said that he didn't use the buildings at all for orientation, he only used the streets. He would like to be able to move the map without having to walk. Probably to search for the items that are far away from his character this way. He is also missing a map scale to be able to estimate how far he will have to walk. One user said that he would change the appearance of the streets to have some texture.

Before we added the quest stages, there was only the introduction text of the quest. One user who tested the prototype at that time was disappointed, he felt he didn't get his dopamine surge when almost nothing happened after he completed the quest, only a little message saying "quest complete" appeared. Another user was disappointed when no flashy effect appeared after picking the quest item. The item just... disappeared after being picked up. Clicking on the quest items was sometimes difficult, we experienced it with multiple users. We should fix that. It was interesting to see that multiple users tried to pick the items up by walking over them. And when being shown that they can pick the items by clicking, they were surprised by the distance (up to 100 meters) from which the items can be picked up. Maybe we could shorten the pick up distance.

5.2.5 User testing results discussion

During the testing we realized that it would be probably better if the items showed up only when the user came near because the users had difficulty finding the items, that were far away because they were quite small at that distance. We could maybe also limit more the render distance so that the user has to walk more around. Now the users search for the small almost not visible items in the distance and and then they walk right towards them. The searching is difficult and there is not much element of surprise included.

The benefits of the game according to the users are: makes you walk, exercise, good for health, makes you orientate yourself, you can socialize with others, make new friends, you can get to know your surroundings better.

The cons of the game are: it needs an internet connection, it drains the battery fast, it is not suited for the disabled.

It looks like for some reason the desert biome is the most prevalent of the four, 2 users noticed it. It has something to do with how we combine two Perlin noise functions together or it could be caused by the interplay of how we scale the Perlin noise functions and how far away from each other the biome centers are.

Generally the users praised the appearance of the game, the visual style, the models. What some of them would change are the streets and buildings appearance. We agree with that, the colors of buildings and streets should be more subtle, a texture should be added and they could be merged into the terrain texture as seen in some other games (for example Dragon Quest Walk).

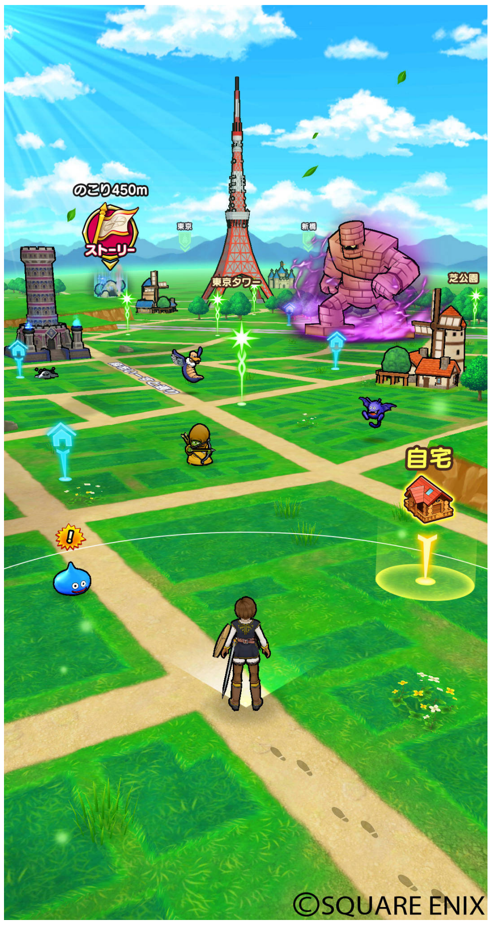


Figure 5.3: The buildings and streets in the game Dragon Quest Walk are represented as textures. Source: Square Enix [21]

During the testing, we noticed that after a player accepts the quest, he is no longer able to see the introduction text of the quest so he has to remember where to find the sword and the shield. We recognized this mistake and helped the users during testing.

5.3 Performance

For testing the performance of our system we created several test cases. The basis for every test case is the default configuration 5.1 and on top of that some variables are modified. We measured the frame times during generation of environment and plotted them into a graph.

We listed the variable domains in table 5.2, location details in table 5.3, and device specifications in table 5.4. Screenshots of the testing locations can be found in section 5.1.

Parameter	**Value**
Vegetation density	69%
Vegetation lushness	90%
Chunk generating distance	530 m
Device	PC
Location	Charles Square

Table 5.1: The default configuration

Variable	**Domain**
Vegetation density	0%, 50%, 100%
Device	PC, OnePlus Nord 2, Xiaomi Redmi 6
Location	Charles Square, Barrandov Terraces, Central Park
Chunk generating distance	250 m, 700 m, 1000 m

Table 5.2: Variable domains

Location	**City**	**GPS**
Charles Square	Prague	50.076104, 14.418764
Barrandov Terraces	Prague	50.026896, 14.393643
Central Park	New York	40.774031, -73.970967

Table 5.3: Location details

Device	PC	OnePlus Nord 2 5G	Xiaomi Redmi 6
CPU	AMD FX-8320	Mediatek Dimensity 1200	Octa-core 2.0 GHz Cortex-A53
GPU	NVIDIA GTX 1050 Ti	Mali-G77 MC9	PowerVR GE8320
RAM	16GB	8GB	3GB
OS	Windows 10	Android 12	Android 9
Resolution	2160 x 1080	2400 x 1080	1440 x 720

Table 5.4: Device specifications

We measured the average frames per second (FPS) after the generation was done and entered the measurements into a table 5.5.

Test case - Default configuration with ...	Average FPS after generation completed
Vegetation density = 0%	115
Vegetation density = 50%	108
Vegetation density = 100%	106
Device = PC	107
Device = OnePlus Nord 2 5G	30
Device = OnePlus Nord 2 5G, Chunk generating distance = 1000 m	30
Device = Xiaomi Redmi 6	8
Location = Charles Square	107
Location = Barrandov Terraces	117
Location = Central Park	95
Chunk generating distance = 250 m	141
Chunk generating distance = 700 m	100
Chunk generating distance = 1000 m	80

Table 5.5: Measurements of average FPS in different testing cases after environment generation completed. The FPS on mobile is capped at 30 by Unity.

In the default configuration we did additional measurements about download and processing the OSM data, because that is the most demanding part of the system. The average amount of downloaded OSM data for a single chunk was 345 kBytes, downloading it took on average 971 ms and processing it took on average 1009 ms. In total 7 chunks are downloaded when using the default configuration, so the true demands are download time around 971 ms, processing time around 7063 ms, and 2.415 mBytes.

We also roughly measured (using Stats in Unity in the Game window) how many triangles are being rendered when we put the camera in such position that all chunks are visible. In the default configuration there are 292.5k triangles rendered, in the default configuration with location changed to Barrandov Terraces, 1.4M triangles are rendered, and in the default configuration with location changed to Central Park, 1M triangles are rendered. From this we can see that vegetation contributes most triangles to the scene triangle count because the area around Charles Square is mostly composed of buildings.

Graphs with frame times for the test cases are shown below. There is an intentional delay of one second before the system sends a request to download data from another chunk (otherwise the program would just freeze for a long time without the user being able to do anything in that time). After the chunk data are downloaded, the program freezes briefly while the chunk content is generating. The large spikes in the graph mostly correspond to generation of one chunk, but sometimes more than one spike joins together because the download of data took longer than usual.

In the first test case, we measured using the default configuration and we altered the vegetation density to be 0%. That means no vegetation was generated. We can see 7 spikes as the individual chunks were loading, and the longest time chunk generation took was 1s. After the last chunk has finished loading, the spikes end and the average FPS after that is listed in table 5.5.

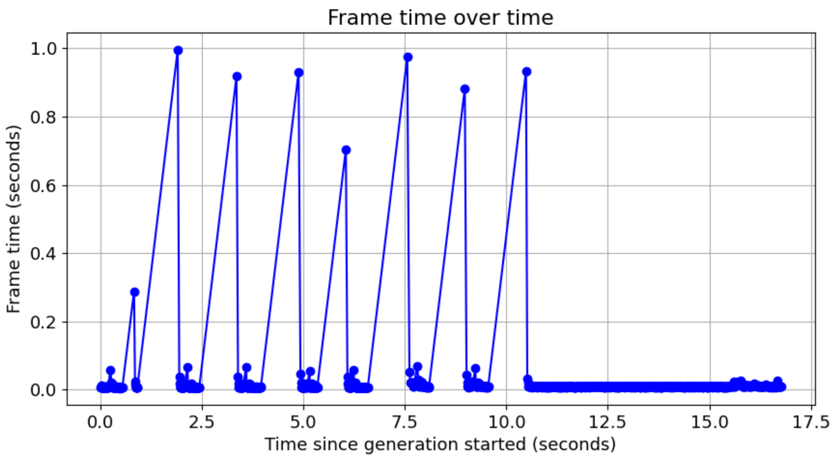


Figure 5.4: Test case: Default configuration with Vegetation density = 0%

When vegetation density was higher, the longest time chunk generation took reached almost 1.4 s.

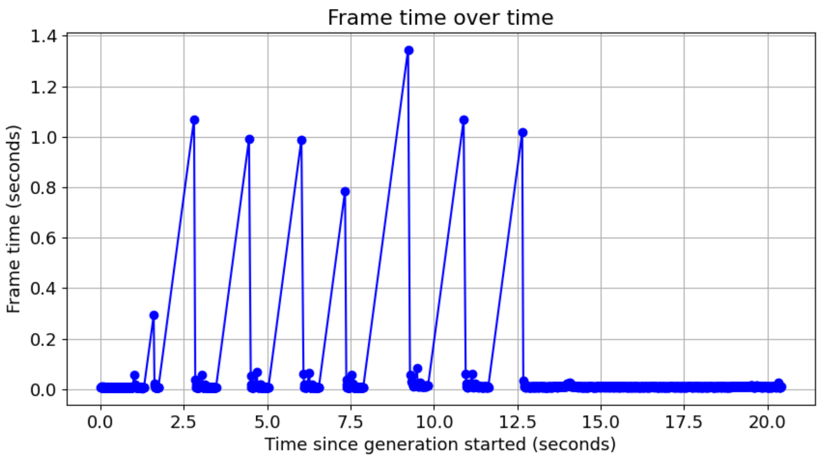


Figure 5.5: Test case: Default configuration with Vegetation density = 50%

With vegetation density set to 100%, the longest chunk generation time was around 1.1 s.

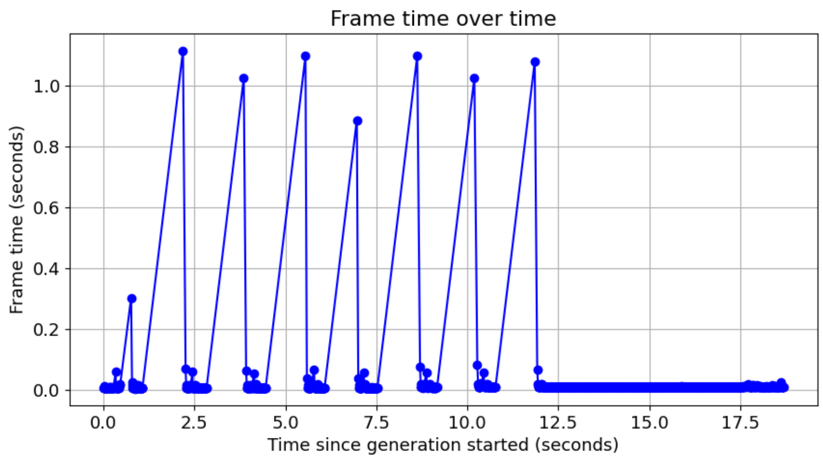


Figure 5.6: Test case: Default configuration with Vegetation density = 100%

Here, two chunks were generated in one frame. This test case has the same configuration as the default configuration because the device used is a PC.

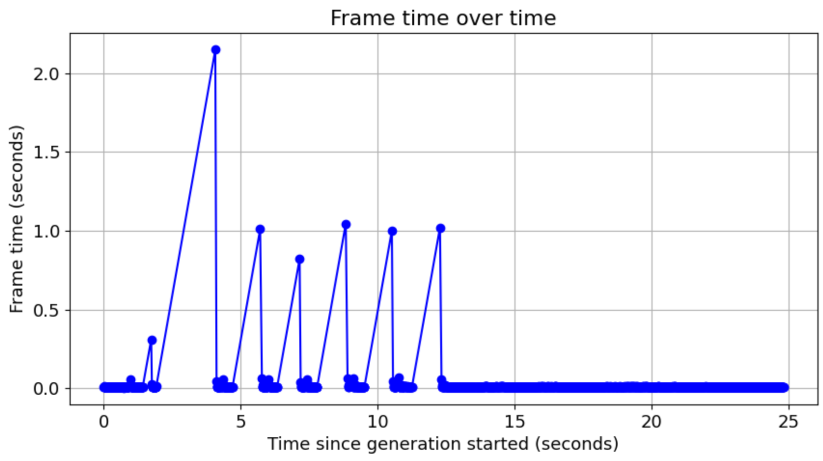


Figure 5.7: Test case: Default configuration with Device = PC

On the faster android phone with which we tested the generation was surprisingly fast. Generating a chunk took about 0.4 s. That means that when users tested our game prototype, the game environment was generated within 10 seconds. When they played the game and moved farther from the location where they began, generation of a new chunk took 0.4 s.

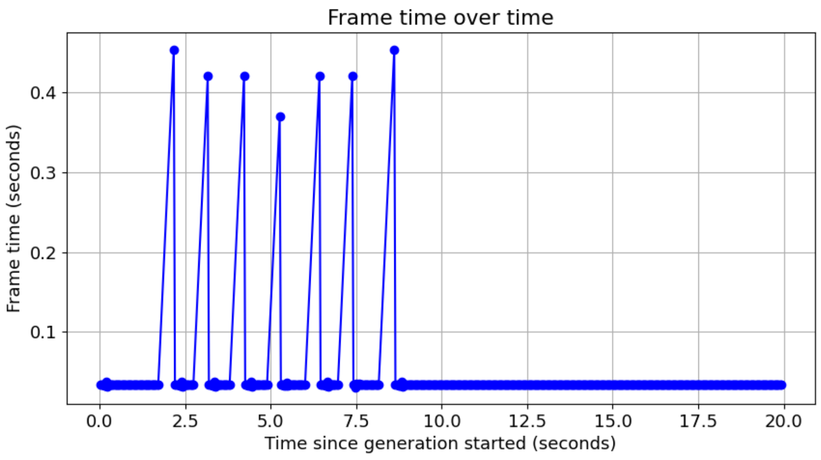


Figure 5.8: Test case: Default configuration with Device = OnePlus Nord 2 5G

This test case used an inexpensive android phone released in 2018. Generation of a single chunk took around 4 seconds, which is the slowest we measured.

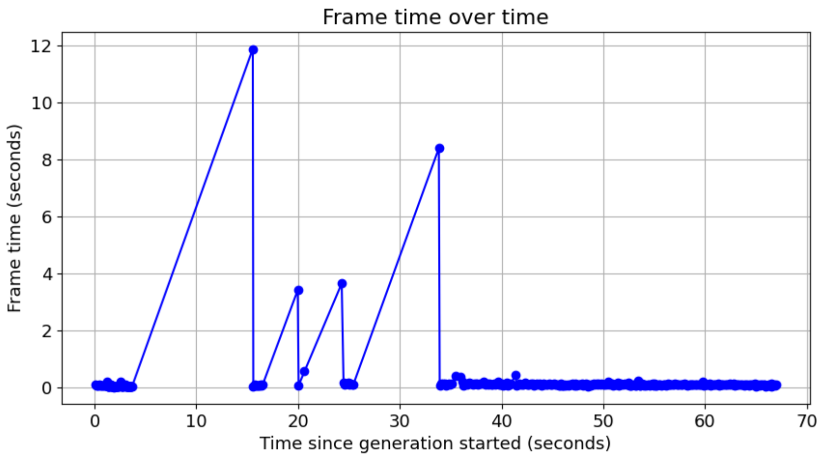


Figure 5.9: Test case: Default configuration with Device = Xiaomi Redmi 6

Here, the location is Charles Square, it is again the same configuration as the default configuration.

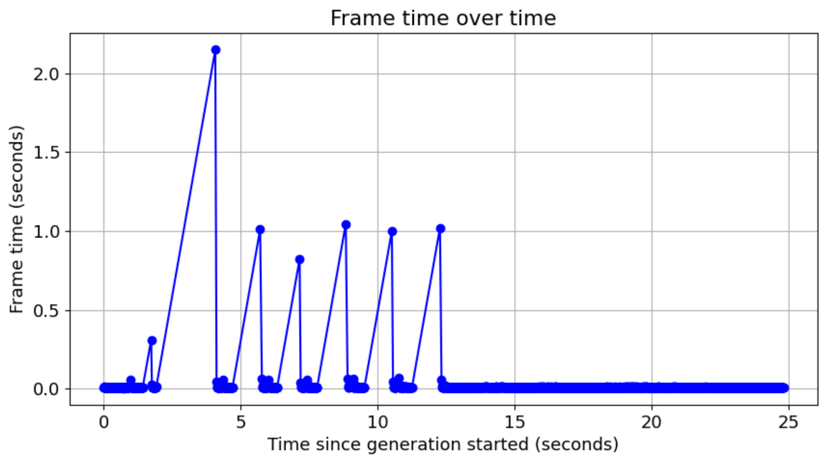


Figure 5.10: Test case: Default configuration with Location = Charles Square

Chunk generation speed in Barrandov Terraces is surprisingly fast, mostly around 0.4 s. We suppose this is because there is much less OSM data to be downloaded and processed than in the Charles Square.

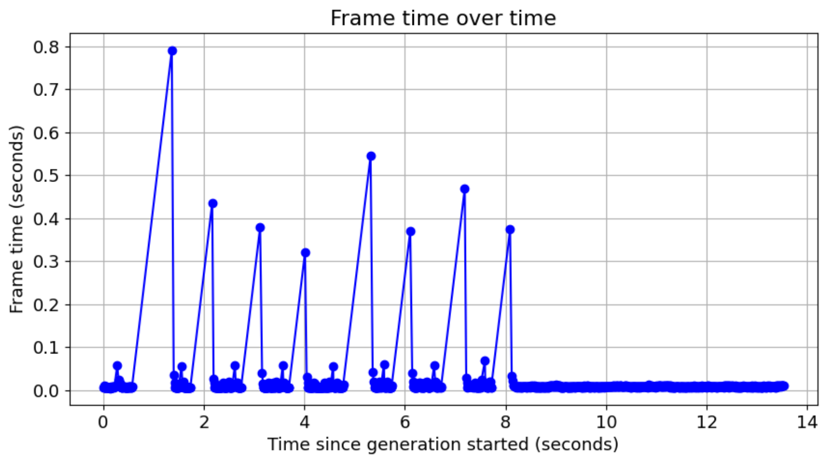


Figure 5.11: Test case: Default configuration with Location = Barrandov Terraces

The chunk generation speed in Central Park is also faster than in Charles Square, we again suppose this is because there is less OSM data to be processed than in Charles Square. The immediate surroundings of this location are mostly greenery without many buildings or streets.

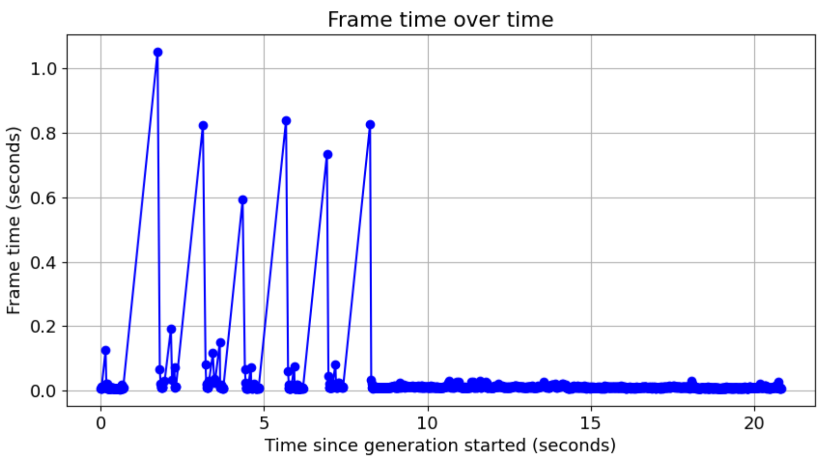


Figure 5.12: Test case: Default configuration with Location = Central Park

Here, we see only two spikes because more chunks did not fit within the limited distance of 250 m.

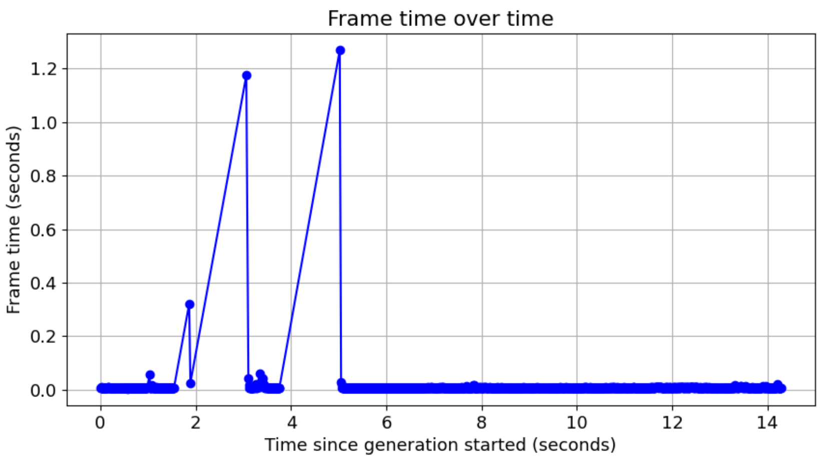


Figure 5.13: Test case: Default configuration with Chunk generation distance = 250 m

Here we see more chunks being generated with the chunk generation distance set to 700 m.

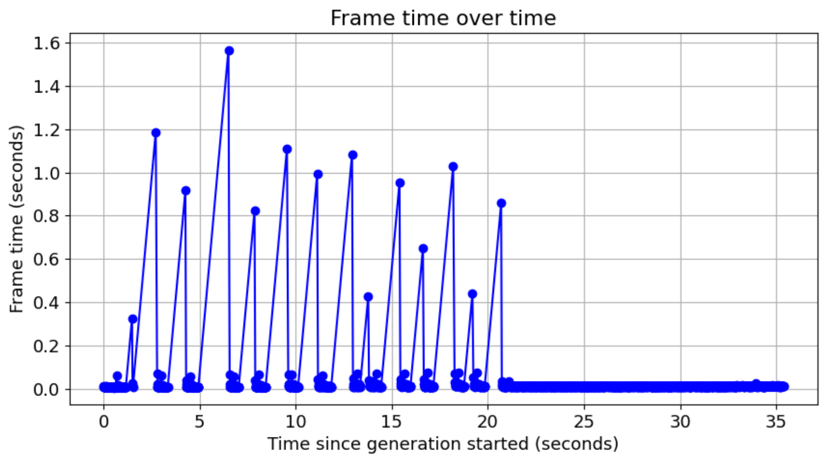


Figure 5.14: Test case: Default configuration with Chunk generation distance = 700 m

When we set the chunk generation distance to 1000 m, we see even more chunks being generated. The generation now takes 40 s to finish.

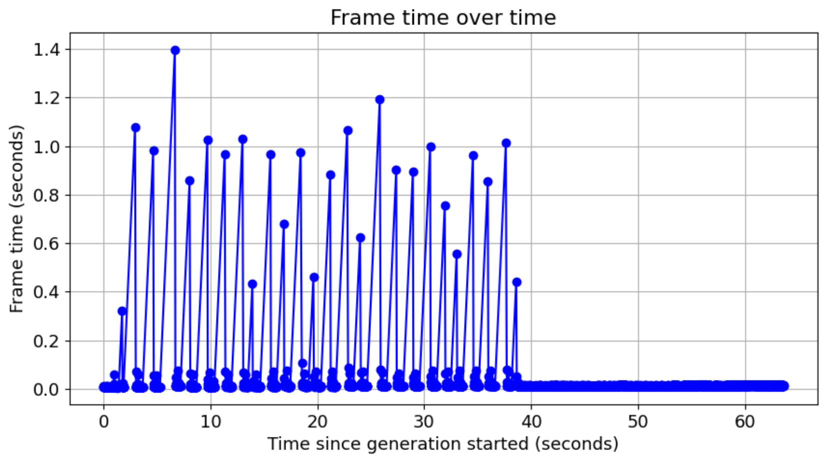


Figure 5.15: Test case: Default configuration with Chunk generation distance = 1000 m

We added one final test case with the more performant Android phone and large chunk generation distance. The environment was completely generated in 30 seconds. The one big spike appears probably because of the data download being delayed.

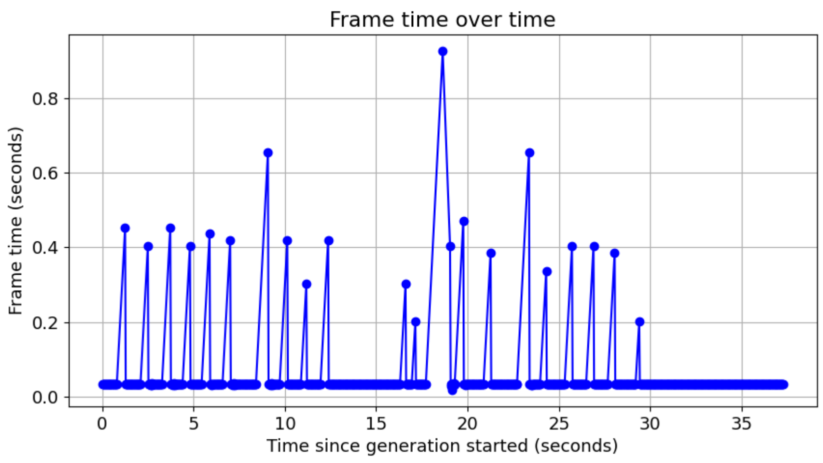


Figure 5.16: Test case: Default configuration with Chunk generation distance = 1000 m and Device = OnePlus Nord 2 5G

5.3.1 Profiling

We profiled the program using the default configuration shown in table 5.1. We did basic Unity profiling and then deep profiling. We focused on the frames that were the slowest and summarized the result of the standard profiling in figure 5.17 and the results of the deep profiling in figure 5.18.

```
|======================================|
| 49% Generate buildings and streets   |
|     | 6% Add components              |
|     | 4% Instantiate                 |
| 44% Destroy                          |
|======================================|
```

Figure 5.17: Profiling result, each function has its percentage of total computation time. The function hierachy is visualized using indentation.

We can see that building and street generation and game objects destruction have the greatest performance impact. GameObjects are destroyed when the freshly generated vegetation collides with street or building colliders. GameObjects are also destroyed when a chunk gets too far away from the player.

In deep profiling result, we see that almost all of the computation time is taken by the Generate buildings and streets function. What is surprising is that smoothing terrain under the streets takes 8% of time, loading XML document 12% of time and Poisson-disk sampling takes 46% of time. It seems that our sampling implementation should be improved. We see that getting sample from normal distribution is also quite demanding, consuming 17% of time. Identifying biome of a point takes in total almost 10% of time.

```
|==============================================================|
| 99% Generate buildings and streets                           |
|    | 98% Create game objects from OSM                        |
|    |    | 63% Generate biome                                 |
|    |    |    | 46% Variable radii Poisson-disk sampling      |
|    |    |    |    | 17% Get sample from normal distribution |
|    |    |    |    | 14% Is candidate position valid         |
|    |    |    |    | 3% Sin and cos functions                |
|    |    |    |    | 3% Vector2 operations add and multiply  |
|    |    |    |    | 1% Populate integer array               |
|    |    |    | 11% Generate vegetation                       |
|    |    |    |    | 3% Add Component                        |
|    |    |    |    | 3% Instantiate objects                  |
|    |    |    |    | 3% Identify biome of point              |
|    |    |    | 6% Color grid cells by biome                  |
|    |    |    |    | 6% Identify biome of point              |
|    |    | 8% Smooth terrain under streets                    |
|    |    | 12% Load XML document                              |
|    |    | 5% Process building way                            |
|    |    | 4% Extract node GPS                                |
|    |    | 3% Process street way                              |
|    |    | 1% Determine if way is street                      |
|    |    | 1% Determine if way is building                    |
|==============================================================|
```

Figure 5.18: Deep profiling result, each function has its percentage of total computation time. The function hierachy is visible from the indentation.

Now we know what to focus on when trying to optimize the program in the future. We should focus on improving our implementation of Poisson-disk sampling and on normal distribution sampling because we think there we will be able to easily achieve the biggest performance boost.

Chapter 6
Conclusion and Future Work

6.1 Conclusion

In this work we created a system for geolocalized procedural generation of a game environment, and we also created and tested with users a simple game using this system. The system divides the Earth surface into areas for which it generates terrain, streets and buildings based on real data and also generates biomes and vegetation not based on real data but designed to support playability. The player character moves using the GPS and compass sensor data of the mobile device, the player can move and zoom the camera using gestures, pick up items and complete a simple quest.

We showed how to create a system that combines two popular concepts in games. Those are procedural generation of biomes (as seen for example in Minecraft) and location-based gameplay (as seen for example in Pokémon Go). This combination is a novel idea and based on the responses from the users in the user testing we see that it has a potential, players find it fun and they ask for more content they could play. We thought that our generation system is quite slow, but when looking at the Frame time over time graph of the mobile device we tested on, it is actually quite good. The environment is loaded within 10 seconds after launching the game and then generating one new part of the world takes only 0.4 seconds for which the game freezes. This generation could be further optimized or split using a coroutine into multiple frames.

6.2 Future work

Future work could focus on making the generation faster based on information we discussed in the profiling section. We could add additional biomes and plants. We could identify some additional data in OpenStreetMap or use some other points of interest source to create game objects at interesting real world places. Representation of water areas should be added, multipolygonal buildings should be better supported, currently we don't support making holes in polygons and we don't support combining a polygon from multiple ways. The streets and buildings should be made more subtle, textures could be added to them and we could think of ways of integrating them into the terrain texture. On the other hand pickable items should be made better visible. The desert biome should be less represented in the game. The player character should be improved so that the player sees better where it is heading. The direction of the character should be dependent also on the current movement direction character and not only on compass readings.

Bibliography

[1] Smith, G. (2015). Procedural Content Generation: An Overview.

[2] Biome. (n.d.). No Man's Sky Wiki. Retrieved May 15, 2024, from https://nomanssky.fandom.com/wiki/Biome

[3] Ingress (video game). (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Ingress_(video_game)

[4] Niese, T., Pirk, S., Albrecht, M., Benes, B., & Deussen, O. (2022). Procedural Urban Forestry. ACM Transactions on Graphics, 41(2), 1–18. https://doi.org/10.1145/3502220

[5] Subnautica Wiki user Bioness. Subnautica Labelled Map Zones and Textures. (n.d.). Subnautica Wiki. Retrieved May 15, 2024, from https://subnautica.fandom.com/wiki/Crater_Map?file=Subnautica_Labelled_Map_Zones_and_Textures.png

[6] Biomes (Subnautica). (n.d.). Subnautica Wiki. Retrieved May 15, 2024, from https://subnautica.fandom.com/wiki/Biomes_(Subnautica)

[7] Melissa, A. (2010, November 11). Summary#6 World Biomes. http://anamelissa-scienceclass.blogspot.com/2010/11/summary6-world-biomes.html

[8] Get Started | Elevation API | Google for Developers. (n.d.). Retrieved May 15, 2024, from https://developers.google.com/maps/documentation/elevation/start

[9] Galin, E., Guérin, E., Peytavie, A., Cordonnier, G., Cani, M., Benes, B., & Gain, J. (2019). A Review of Digital Terrain Modeling. Computer Graphics Forum, 38(2), 553–577. https://doi.org/10.1111/cgf.13657

[10] Pirk, S., Benes, B., Takashi Ijiri, Li, Y., Deussen, O., Chen, B., & Radomir Měch. (2016). Modeling plant life in computer graphics. https://doi.org/10.1145/2897826.2927332

[11] Emilien, A., Bernhardt, A., Peytavie, A., Cani, M.-P., & Galin, E. (2012). Procedural generation of villages on arbitrary terrains.

The Visual Computer, 28(6-8), 809–818. https://doi.org/10.1007/s00371-012-0699-7

[12] B. Veld, B. Kybartas, R. Bidarra, & Meyer, J. J. C. Meyer (2016). Procedural generation of populations for storytelling.

[13] User Rudi dev. (2022, January 27). Biome generation explained | Voronoi diagrams into biomes. YouTube. https://www.youtube.com/watch?v=D9_FB6hWnME

[14] Ecormier-Nocca, P., Cordonnier, G., Carrez, P., Moigne, A.-M., Memari, P., Benes, B., & Cani, M.-P. (2021). Authoring consistent landscapes with flora and fauna. ACM Transactions on Graphics, 40(4), 1–13. https://doi.org/10.1145/3450626.3459952

[15] Fischer, R., Dittmann, P., Weller, R., & Zachmann, G. (2020). AutoBiomes: procedural generation of multi-biome landscapes. The Visual Computer, 36(10-12), 2263–2272. https://doi.org/10.1007/s00371-020-01920-7

[16] Alha, K., Koskinen, E., Paavilainen, J., & Hamari, J. (2019). Why do people play location-based augmented reality games: A study on Pokémon GO. Computers in Human Behavior, 93, 114–122. https://doi.org/10.1016/j.chb.2018.12.008

[17] Augmented Reality: The Future of Manufacturing. (n.d.). SAP. Retrieved May 2, 2024, from https://www.sap.com/cz/products/scm/industry-4-0/what-is-augmented-reality.html

[18] Rapp, D., Niebling, F., & Latoschik, M. E. (2018). The Impact of Pokémon Go and Why It's Not about Augmented Reality - Results from a Qualitative Survey. 2018 10th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games). https://doi.org/10.1109/vs-games.2018.8493442

[19] Eberly, D. (2002). Triangulation by Ear Clipping. https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf

[20] Whelan, P. (2011). Geohash Intro - Big Fast Blog. Retrieved May 15, 2024, from https://bigfastblog.com/geohash-intro

[21] Fuller, A. (2019, June 3). Square Enix Announces Dragon Quest Walk. RPGamer. https://rpgamer.com/2019/06/square-enix-announces-dragon-quest-walk/

[22] Intel. (n.d.). Real-Time Systems Overview and Examples. Intel. Retrieved May 21, 2024, from https://www.intel.com/content/www/us/en/robotics/real-time-systems.html

[23] Mojang Studios. (2021, December 5). Minecraft [Screenshots by YouTube user MaxStuff]. Retrieved May 21, 2024, from https://www.youtube.com/watch?v=f_fWH8eJJgg

[24] Gasch, C., Sotoca, J. M., Chover, M., Remolar, I., & Rebollo, C. (2022). Procedural modeling of plant ecosystems maximizing vegetation cover. Multimedia Tools and Applications. https://doi.org/10.1007/s11042-022-12107-8

[25] Vadim S. Simple Approach to Voronoi Diagrams. (2015, March 4). CodeProject. https://www.codeproject.com/articles/882739/simple-approach-to-voronoi-diagrams

[26] Geohash. (n.d.). Wikipedia. Retrieved May 15, 2024, from https://en.wikipedia.org/wiki/Geohash

[27] Nahodil, P. (2023). Creating 3D scenes for a train simulator.

[28] Kyzr, O. (2023, May). Procedural Generation of Outdoor Scenes. https://dcgi.fel.cvut.cz/en/theses/2023/kyzrondr/

[29] Biome. (n.d.). Minecraft Wiki. Retrieved May 21, 2024, from https://minecraft.fandom.com/wiki/Biome

[30] Biomes (Subnautica). (n.d.). Subnautica Wiki. Retrieved May 21, 2024, from https://subnautica.fandom.com/wiki/Biomes_(Subnautica)

[31] Biomes. (n.d.). Rimworld Wiki. Retrieved May 21, 2024, from https://rimworldwiki.com/wiki/Biomes

[32] Biomes. (n.d.). Terraria Wiki. Retrieved May 21, 2024, from https://terraria.fandom.com/wiki/Biomes

[33] Shanklin W. (2023, June 29). Image of Pokémon Go. In article "Pokémon Go" developer Niantic is laying off 230 employees. Engadget. https://www.engadget.com/pokemon-go-developer-niantic-is-laying-off-230-employees-180438129.html

[34] Earth Science Data Systems, N. (n.d.). SRTM. Earthdata. https://www.earthdata.nasa.gov/sensors/srtm

[35] klink, jemerick. NGeoHash package. Retrieved from https://www.nuget.org/packages/NGeoHash

[36] Lague S. (2018, November 23). [Unity] Procedural Object Placement (E01: poisson disc sampling). YouTube. https://www.youtube.com/watch?v=7WcmyxyFO7o

[37] Beider V. (2022, August 6). Godot - Rimworld style tilemap shader tutorial. YouTube. https://www.youtube.com/watch?v=91fkApi8RUQ

[38] Re-Logic. (2011). Terraria [Video game].

[39] Mojang Studios. (2011). Minecraft [Video game].

[40] Hello Games. (2016). No Man's Sky [Video game].

[41] Ludeon Studios. (2018). RimWorld [Video game].

[42] Unknown Worlds Entertainment. (2018). Subnautica [Video game].

[43] Niantic. (2016). Pokémon Go [Video game].

[44] Niantic. (2013). Ingress [Video game].

[45] Bay 12 Games. (2022). Dwarf Fortress [Video game].

[46] Paradox Development Studio. (2012). Crusader Kings II [Video game]. Paradox Interactive.

Appendix A

Content of electronic appendix

Folder	Content
bin	.apk installation file for Android
src	source code of the generation system and the game prototype
latex	source code of the thesis
images	screenshots
readme	user manual
thesis	this pdf file

Appendix B

User manual

B.1 Installation

In the electronic appendix in the folder bin, you can find a .apk file. Install this on an Android device. The app needs permission to read the GPS location of the device and needs a connection to the Internet to download data from the Google Elevation API and overpass-api.de.

B.2 Using the app

The app should detect the GPS location of the device and start generating the environment around the location. The character of the player moves according to the GPS and compass sensor data of the device. The view can be zoomed in by pinching the screen with two fingers and rotated around by swiping with one finger. After pressing the button Quest, a quest window appears and a quest can be accepted. You are then expected to start walking. The quest instructions help you find the needed quest items. You can see how many you have collected so far in the UI. After collecting enough, you can press the quest button again. Now the content of the quest window is different, and you can complete the quest.

B.3 Settings

The settings can be found under the DevTools button. A settings window appears. Here, you can enable some of the UI elements, enable using building heights. Next, you can enable GPS simulation. The app keeps track of two GPS sources, one is the device GPS sensor (real GPS), and the other is the simulated GPS. You can set the initial simulated GPS by selecting a location in the dropdown and clicking the Set simulated GPS button. After that, you can override the real GPS by the simulated GPS by enabling the Enable GPS simulation toggle. You can use the joystick UI to easily modify the simulated GPS. If simulated GPS is enabled, the character will move around when you touch the joystick. The last option in the dropdown sets the simulated GPS to copy the GPS of the current real GPS. This is useful when you want to explore the surroundings of your current physical location without having to walk. (We purposely hid this option from the testers of our app to keep their motivation to walk.)

The Chunk generating distance slider sets how far the chunks will be generated and after what distance they will be unloaded. The Vegetation lushness slider sets the size of areas where no plants grow. Vegetation density sets what how many plants are considered for generation before some further biome-specific rules are applied. The button Regenerate chunks discards the currently loaded chunks. Then, new chunks should be generated with the current settings.

Appendix C
User testing document

C.1 Introduction

You will be testing a prototype mobile game. The game environment is three-dimensional and combines a representation of real terrain, houses and roads with generated biomes and vegetation. You move around in the game environment by walking in the real world. The test will take place outdoors and will require an estimated walk of several hundred meters. The testing will consist of two phases - in the first, you will try out the app and attempt to complete a task, which you will find under the "Quest" button. In the same time, you will say out loud what you are doing and why you are doing it, as well as your current impressions. In the second phase you will fill in a questionnaire where you will answer several questions.

We may take audio or video recordings during the testing to improve the quality of the data collected. All data collected will be anonymised before publication and it will not be possible to identify the test participant from the data.

Please pay close attention to your surroundings while testing so as not to compromise your personal safety.

C.2 What we will observe during the testing phase

How intuitive are the controls? Do users understand the biomes and their meaning? How well can users navigate the game world? Do users understand where to look for items to complete a task? We will track any hiccups as users progress through the application and observe why they happened.

C.3 Questionnaire

1. What is your impression of the game?
2. What do you think about controlling the game by walking?
3. How well were you able to navigate the environment?

4. What was it like for you to find the items needed to complete the task?
5. How much did you enjoy the game?
6. Did you get stuck anywhere during the game?
7. What did you like and what did you not like?
8. What would you improve or add?
9. What do you think is the importance of biomes and vegetation in the game?
10. How did you like the biomes and vegetation?
11. How did you like the three-dimensional terrain in the game?
12. How did the roads and buildings help you find your way around?
13. How would you change the roads and buildings?
14. How would you describe this type of game?
15. What experience do you have with similar games?
16. What do you think is important in a game like this?
17. What do you think are the advantages and disadvantages of this type of game?
18. What do you enjoy about this type of mobile game and why do you keep coming back to it?
19. Would you be interested in playing a game of this type and what would you be able to do in your ideal game?