Graduation report

Development of an Interactive Scenario

for TenneT Virtual Vision

Student Anton Volkholz

Number 417428

Study program Creative Media and Game Technologies - Saxion Enschede

Graduation report

Development of an Interactive Scenario

for TenneT Virtual Vision

Student Anton Volkholz

Number 417428

Email anton.volkholz@me.com

Company Coach Marc Ortner

Guiding Teacher Taco van Loon

Project Interactive Scenario for TenneT Virtual Vision

Study Program Creative Media and Game Technologies - Saxion Enschede

Date June 18th 2019

I. Preface

This report describes the project I worked on as part of my graduation for the Creative Media and Game Technologies study course at Saxion Enschede. The project was completed at “Die Wegmeister”, a marketing agency based in Stuttgart, for their client TenneT, which is a transmission system operator based in the Netherlands.

The work in this project focused heavily on technical art, which is extremely interesting for me, since it combines design with technical tasks such as the development of visually complex particle effects and the interaction with them. Working on 3D art and designing a visually pleasing style, while also focusing on technical tasks to optimize development or creating functionalities is very enjoyable for me.

Previous projects in cooperation with the Game Engineering course at Saxion, as well as the Smart Solution Semester helped me develop my skills and competences in this direction.

The opportunity to work on a project such as this, in a work environment that is professional and calm offered a valuable experience that I am thankful to have had.

I would like to thank my supervisors at the Company and at Saxion for their continuous support and for answering any questions I had throughout this project.

Anton Volkholz

Stuttgart, June 18th, 2019

II. Abstract

This report describes the development workflow of an interactive scenario in Unity that runs on a touch table and visualizes the interdependencies of a power grid. The project was done for the company TenneT, the first European cross border transmission system operator. In this project, within 10 weeks of development, a scenario should be created that visualized these interdependencies to help explain the continuous expansion of the power grid to people from outside the field of energy transmission systems. For this, a time efficient workflow needed to be defined to complete the development within the small time-frame available.

This report focuses on the research process that created the basis necessary for formulating a well-founded workflow approach, which aimed to ensure the completion of the development process in time. It also describes the development process of visualizing an energy grid using particle systems that interact with each other.

In the theoretical analysis existing information is collected and reviewed so a basis for formulating a workflow and development approach can be determined. After comparing the different pipelines Unity offers, the High Definition Rendering Pipeline (HDRP) was selected as the most suitable one, because it offers two exclusive features, the VFX graph and the Decal Projector, both of which were valuable for ensuring the optimal outcome of the project.

The VFX graph is suitable for simulating a large number of particles, because it calculates on the Graphics Card, rather than the Processor. Although this has its drawbacks, regarding the lack of physics-based interaction of particles with their environment, it proved ideal for the particle-based power grid visualization. The underlying environment, which was meant to support the scenario's intention through a visually appealing background that catches the interest of the user, was created using the Gaia Terrain Toolset, developed by Procedural Worlds. The payed plug-in redeems its price through superior quality of the terrains it can help create. With it the background terrain was sculpted in very short time and using the environment textures that come with Gaia the terrain could easily be textured to fit the intended detailed style required by the client.

To optimize the development with multiple particle systems, several tools were created that allow convenient adjustment and management of parameters, as well as the interaction between particle connections. To achieve an interface that gives a good overview of available parameters and development functionalities, the Odin Inspector plug-in, developed by Sirenix, was used.

The development could be completed in time, due to the thought through workflow that was formulated ahead of the development. Having a thorough understanding of the advantages and drawbacks of each tool or pipeline, prevented unexpected problems to arise during later stages of the development, which would have cost a significant amount of time.

III. Table of contents

IV. List of figures

1. Introduction

This report describes the research and development process of a project for my Graduation.

The project was done at “Die Wegmeister”, a marketing agency based in Stuttgart, Germany. The client in this project was the company TenneT, a transmission system operator, based in the Netherlands. It has roughly 3.500 employees and is considered the first European cross border transmission system operator.

Every year in May, an event is held in TenneT’s Showroom in Berlin, called TenneT Virtual Vision. In this Showroom a variety of Stations are presented to visitors, that tell about the history, the present state and the potential future of energy transmission in Virtual and Mixed Reality. During the annual spring event, new Scenarios are introduced to the visitors of TenneT Virtual Vision. The project described in this report provides an interactive scenario for one of the Stations of the Showroom, called Tangible Bar.

In this project, the interdependencies within a power grid are to be visualized in an appealing and understandable way. The scenario should consist of a good-looking environment as a background, in which several grid components are visible, such as transformer stations, renewable and conventional power generators and central areas of power consumption. A particle-based visualization of the power grid should then react to changes made by the user to the different components of the grid. When adding additional power generators for example, the network should react accordingly.

The time frame was quite restricted, with only around eleven weeks available for development. That is why large parts of the project were focused on workflow optimization and creating an efficient development process.

My work in this project was focused on the creation of 3D assets and achieving a good-looking environment as a background, as well as technical development for creating interactive functionalities and interfaces that simplify the development. These include “PathLink”, a system for enabling particles to travel along a conveniently definable path in the scene, as well as “StateShift”, which lets the different particle systems communicate with each other about changes in the scenario. The research prior to development is heavily focused on determining an optimal pipeline and workflow for the project’s development process. Going into production with a solid and efficient workflow is a valuable basis for developing under time pressure.

A second developer focused on creating the user interface for the gameplay and how input is handled and translated to enable the scenario to react to it. Together we developed this interactive scenario using Unity 2018.

2. Methodology

The development approach is based on the design-based research model. After analyzing the projects context, its requirements and problems, a central research question could be formulated. This served as the basis for the following research and development.

In order to get a clear overview of the visual and technical requirements and possible issues or obstacles during development, all existing knowledge is analyzed. In this phase it is important to get a complete understanding of what the development workflow will look like. That means determining which tools will be used, how the main functionalities can be developed and what requirements and restrictions the development process might be confronted with.

To create this theoretical basis for development, information needs to be gathered from websites, forums and documentation on the possible tools, that can be used for development. This is done in a 360-information scan, in order to gather as much data as possible on the topic. Information is collected mainly through desk research, by reviewing articles or forums online, as well as through working with Unity and the required tools directly. In this trial and error approach, a lot of unexpected issues can be solved right away, and even during development this method still applies and helps dealing with any problems that might occur.

After gathering information about all necessary aspects of the project's development, a solution for workflow and development is formulated. This describes what the optimal workflow looks like and how the required components of the scenario can be developed.

The development process is structured in four iterations. After each development phase, based on the conceptualization of a solution for development, the current state is tested, and the progress is reflected on. Feedback from client and project lead is gathered and formulated into additional requirements, to guide further development in the following iteration.

The development process starts with the mockup and prototype iterations, where the focus lies on the scenario's visual appearance and on testing all necessary functionality that needs to be implemented. Later, in the Alpha iteration, both 3D design and functionality are integrated, based on the previously developed prototypes and style mockups. In the Beta Iteration, the feedback from Alpha testing is addressed and implemented and a finalized version is refined.

3. Problem indication and practical analysis

For someone outside the field of expertise, the energy grid and its interdependencies and functionalities might be hard to understand. Due to that, it is even more difficult to explain the necessity of further expansion of the energy grid, as well as the investment in new technologies focused on grid optimization.

The problem TenneT continuously has, is that it is difficult to visualize these dependencies within the energy network in a way that explains and justifies the expansion of the existing power grid. A visualization gives the company the opportunity to make these dependencies understandable and easier to grasp. The project specific problem is to visualize this in a visually appealing way, that feels reactive and organic, while conveying a clear message to the viewer.

The scenario should consist of a background terrain, grid components and buildings on top of that background and lastly an energy grid represented by particle streams. In the User Interface the user should be able to alter the state of different components in the scenario. These are structured in four categories:

- power generation
- power consumption
- power storage
- grid technologies

If changes are made to these categories, e.g. power consumption is increased, the scenario's components should react, such as the growing of a city and therefore a higher power consumption. The particle streams should react as well by changing color or spawn rate accordingly, to visualize a shortage of power in the system, for example.

The background layer should be an appealing environment with high amount of detail, so it serves as a visually pleasing background, while not drawing too much attention away from the scenario's core content. The terrain should visualize an abstracted German landscape, have good surface structure and fitting textures, as well as detail objects like trees, grass or rocks. A water shader should help to visualize the northern seaside. The environment should remain abstract, since a too accurate representation of Germany might cause wrong conclusions for the users, about the layout and development of the actual power grid, which the scenario does not intend to represent.

The component layer should contain all the grid components like power plants or wind and solar parks, transformer stations and convertor platforms, and the urban and industrial buildings that visualize the power consuming cities. These buildings should have a clean and simple style, with light color, in order to be easily visible on top of the detailed environment background. These buildings should react to changes of the above-mentioned categories by appearing and disappearing, e.g. when a city is supposed to grow or shrink.

The grid layer should consist of particles traveling through the grid and reacting to changes made by the user, by changing their color, size or spawn rate. This layer will be the main

feedback to the user of the current state of the scenario. A power shortage or surplus should cause the particles to glow red, for example.

Finally, a functionality layer should enable all the interactive elements in the scenario to work together and react to changes. The scenario will have one definite state at a given moment in time and every component or particle system should be able to react whenever that state is updated.

The project has a visual challenge, regarding the style and perspective of the scenario's main components, as well as the supporting visual background. The underlying terrain should be appealing and represent a heavily abstracted German landscape.

The technical challenge is that an energy grid visualized with particles should have multiple influencers and react to changes dynamically, while feeling organic and reactive. The particle systems need to be connected to each other and since multiple systems will need to be used simultaneously the development with these systems should be as efficient as possible.

Central research question

In order to determine a basis for the direction the research and development should be focused on, a central research question is formulated.

"How does a suitable workflow and development approach need to be formulated, that will ensure a time efficient development process, so that an interactive scenario that uses high quality particle systems in Unity and runs on a touch table can be created within a short period of time?"

Since the project is extremely time restricted, the central problem focuses on establishing an efficient workflow for optimal development within the project. This includes determining a suitable pipeline within Unity and the creation of additional tools and interfaces that will help during development. The workflow should present a suitable basis for the development of the more specific designs and functionalities of the scenario.

4. Knowledge analysis

Before being able to create a suitable workflow, additional information needs to be collected, in order to formulate a well-founded solution for the central research problem.

This is especially valuable due to the little available development time in the project. Through first analyzing the existing knowledge and possibly suitable tools, a good basis for formulating an optimal workflow can be determined. By formulating a well-founded and thought through pipeline early on, a large amount of time is saved in later development, where problems with certain tools or workflows can disrupt the process and progression of the project.

The following questions cover the three main fields of required knowledge and all the information necessary, in order to determine a suitable pipeline and workflow, as well as creating an environment and particle system in the desired style.

Additional research questions

1) How can a visual environment, style and perspective be designed, in order to work well with the technical limitations of the interactive touch table?

This area focuses on the technical requirements and limitations and how the visual component of the scenario can be developed, in accordance with the client's requirements and ideas, all while keeping a stable performance.

2) How would a pipeline need to be defined, so that it is suitable for a flexible development process with Unity particle systems?

Different tools and pipelines in Unity will be reviewed in order to determine a suitable workflow. With the help of documentation and literature about Unity's built-in features, and external plug-ins, a good overview of all available options will be achieved.

3) How can an abstract visualization of an energy grid be designed, that feels organic and responsive and supports the scenario's core message?

Here it is necessary to gather information about how a real power grid functions and reacts to changes. The visualization of this will be further determined during the development of prototypes, where the interactive functionality can be tested more accurately.

5. The 360-information scan

Answering the additional research questions

The additional questions, as formulated above, will be answered by collecting information on each topic and reviewing advantages and disadvantages of possible tools, pipelines and workflows. With the gathered data, answers to the questions will be formulated in the theory conclusions.

1) How can a visual environment, style and perspective be designed, in order to work well with the technical limitations of the interactive touch table?

First, the technical limitations of the touch table need to be specified. These can be determined using the data sheet provided by the client. The table has a display as its surface and can be controlled both by touch input and by five tokens that can be placed on the screen. These tokens have QR codes on the bottom, which can be tracked by cameras underneath the partly transparent screen. The table unfortunately is not available during development, which means that any testing needs to be done in PC compatible builds.

The table screen is 55 inches in diameter with a Full HD resolution, while there are spaces between pixels which make the display transparent enough so that four cameras behind the screen can track the token's QR Code markers. This is the second way of registering input next to the conventional touch interaction. With the tokens being 8.7 centimeters (roughly 3.4 inches) in diameter, they take up a significant amount of space on the screen if all 5 are in use.

The system the touch table runs on is equipped with a Nvidia GTX 960 graphics card and an Intel Core i5 processor. Most important for the performance of the scenario is the GPU due to the extensive visual component of this project. If the graphical side is kept at a reasonable level, the workload on the CPU should not have a substantial impact on the performance. This information was made available in form of a Technical Specification overview by the client. *(Spezifikationen (PC, Touch, Display, Etc.), 2019)*

Since the scenario is played on a table display, the perspective towards the screen is a lot different than with a conventional desktop display. The users will be standing around three of the four table sides, the two wide sides and one end, since the tables other end is predefined for placing the tokens when not in use *(see Figure 1)*.

In order to maximize the usable space left, a vertical or “portrait” perspective would be suitable. This way the users can stand around the remaining three sides of the table, while still being able to have a good perspective on the scenario. Since the scenario should depict an abstract version of the German landscape and energy grid, the northern seaside and southern mountain range and the resulting top to bottom contrast would fit nicely into a vertical frame.

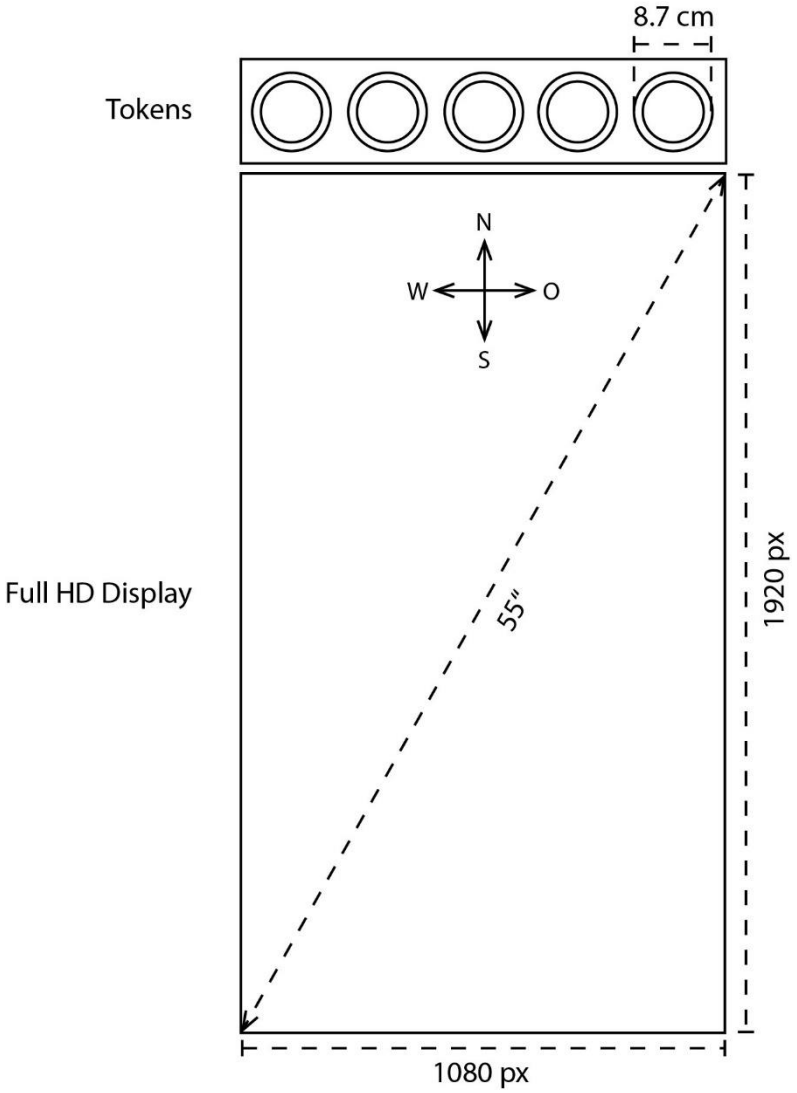


Figure 1: Layout blueprint of the interactive touch table with technical specifications

The problem with a horizontal or “landscape” perspective is that only users standing on the correct side of the table will have a good overview of the scenario, while the rest of the users will have trouble keeping their orientation.

The scenario content should be visible at the same distance and quality in all parts of the screen. That means the buildings and environment on the lower end of the screen, which would appear to be closer to the camera, should be the same size as the buildings on the top, which would be far in the distance, when using a perspective camera. That is why an orthographic camera would be suitable for this situation. Every object is then rendered at the same size on screen, since the orthographic camera has no perspective and any depth perception is lost.

An object in the scene that is equipped with different levels of detail (LOD), would switch texture and model quality to lower levels when the camera would move further away from it *(see Figure 2)*. This is done to optimize performance, because objects in the distance will not be rendered at their full quality. Since every object needs to be visible at the same level of detail when using an orthographic camera implementing different LOD levels will not have a desired effect, if any at all. The terrain's LOD system is built-in, however it would not use its full functionality, since the camera has a static perspective.

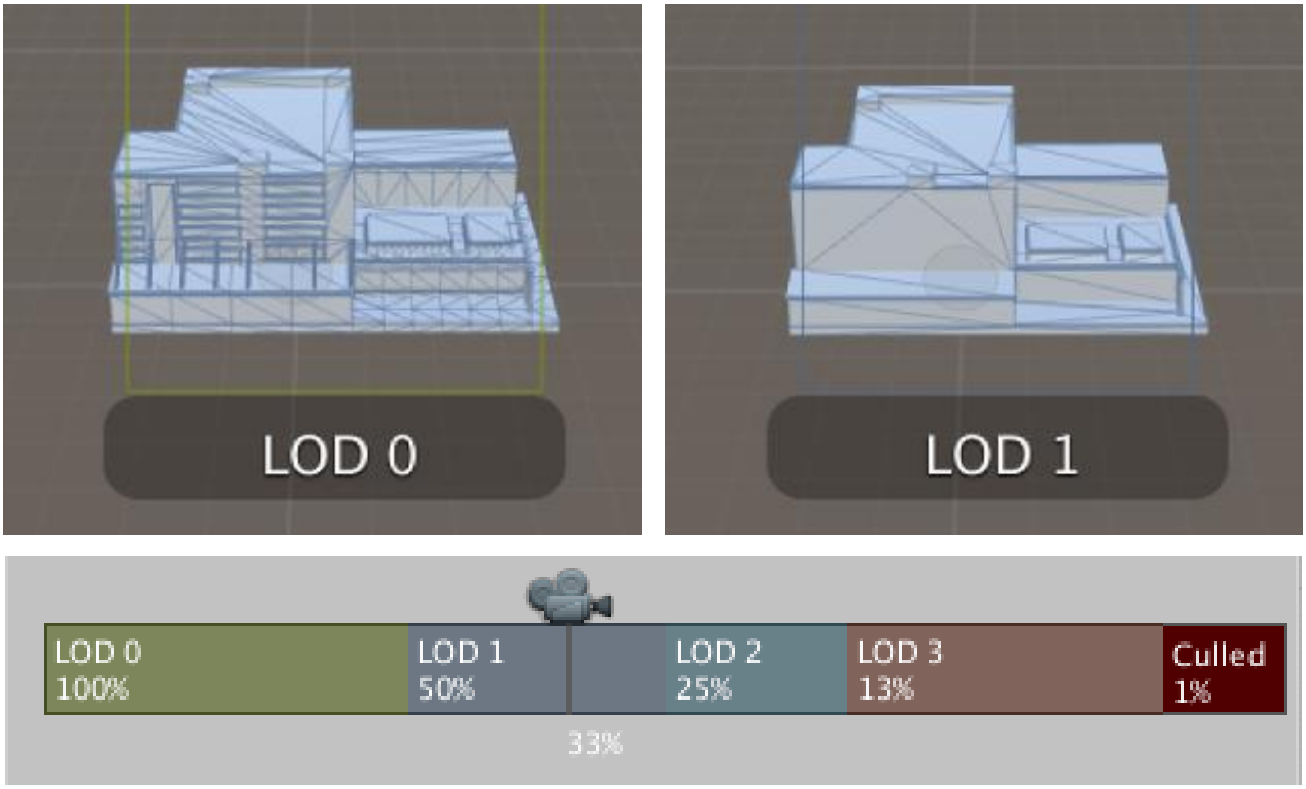


Figure 2: LOD Group setup (Unity Technologies, 2019)

The client wants a simplistic representation of the energy grid. The background should be detailed enough to portrait a nice environment terrain from mountains in the south to the seaside up north, however abstract enough as to not actively portrait the German landscape or power grid specifically. As mentioned earlier, a too accurate representation might lead to users drawing wrong conclusions about which areas the grid expansion will affect.

The grid components should have little detail and a clean style, so they are easily distinguishable from the underlying terrain. The components should have small animations or effects that make them come alive a little more than just being static objects. Additionally, each building should have a small foundation that will blend it in to the terrain a little better. This can easily be achieved using decals, which could be enabled and disabled together with the components. This way the ground texture does not have to be changed in the terrain's textures, but simply by adding a decal projector to the component object. The abstracted style should not get too simple and childish due to a low poly count or simplistic textures. Each component should be easily recognizable by using little detail to create clear shapes. This was established in discussions between the project lead and the client about their requirements and ideas regarding the scenario's look.

There should not be any 2D icon overlay other than adaptive UI dials that the input tokens will generate around them and an informative UI layer on the bottom of the screen. This means all other indications on the scenario's state need to be conveyed through the 3D scenario itself.

Unity has a built-in system for creating simple terrains. It allows Heightmaps to be imported, however only in RAW format, which can be created in Photoshop with some obstacles. It allows a very comfortable painting and blending of multiple textures and, if set up correctly, the mesh painting of trees, rocks or other detail meshes to enrich the background environment *(Unity Technologies, 2019)*. Some of the terrain tools, like the grass and detail mesh painter, tend to have some minor issues with HDRP and especially the built-in tree generator is completely incompatible with the HD Pipeline. This was discovered, by trial and error during the testing of the terrain tool in the High Definition Pipeline. The issue is caused by the Bark and Leaf shaders, which only work in the built-in renderer, and HDRP upgraded versions of these are not available, as of Unity2018.3. The two shaders are required for the terrain to recognize the model as a tree, in order to apply random rotation, height and color values, as well as wind animations *(Unity Technologies, 2019, Unity Terrain Trees)*. The absence of these very helpful sub-tools makes tree placement onto the terrain extremely tedious.

The external terrain generation tool *Gaia*, created by Procedural Worlds, allows very modular editing of terrains. It has tools for procedurally texturing terrains depending on the surface structure and slope, as well as for placement of detail meshes like plants, and even procedural villages *(Procedural Worlds, 2019, Introduction to Gaia)*. The most valuable tool, however, is the terrain stamper, in which any heightmap can be imported and even converted to an appropriate format, and then stamped anywhere on the terrain. The option to rotate, scale and change the height of the stamp is incredibly valuable for rapid terrain generation and for adjusting the surface in polishing, for example *(Procedural Worlds, 2019, Stamper Introduction)*.

Gaia offers a lot of helpful options and tools for terrain generation and provides a highly efficient workflow, ideal for a development under time pressure. Unfortunately, the tree painting system is very limited in the HD Pipeline and Gaia is only optimized for procedurally filling a terrain with textures and trees, rather than detail mesh painting *(Procedural Worlds, 2019, Spawners – Terrain Trees)*. The rather time consuming and tedious manual placement of trees could be optimized by creating a simple custom tool for randomized transforms, for example.

The Unity Terrain is highly performance optimized with its clustered LOD levels both on geometry and textures and should not impact performance significantly *(Unity Technologies, 2019, Unity Terrain Engine)*. The buildings, as mentioned, should have a clean and simple style. This means a low polygon count and little texture detail, which will not require higher texture resolutions.

The overall load of the environment art will be reasonably low. This leaves a lot of room for the development of the more complex particle streams to visualize the electric grid.

2) How would a pipeline need to be defined, so that it is suitable for a flexible development process with Unity particle systems?

The project is focused on an appealing visual style, which should support the main functionality and message of the scenario. Specific effects for the animated interaction with components would require development of custom shaders, which could be done easily with the Shader Graph, available in all Pipelines. For ensuring a good performance, the Lightweight Render Pipeline (LWRP) would be suitable, since it gives good results even on a mobile device. The Projector system, used to place decals dynamically into the scene, is only available in the HD Pipeline *(Unity Technologies, 2019, Decal Projector)*, just like the Visual Effects Graph.

The High Definition Render Pipeline (HDRP) offers outstanding visual quality as well as vital features like the Decal Projector and especially the Visual Effects Graph. Both these tools are currently not available in other Pipelines. Although an optimal performance could be achieved with LWRP or the Built-In Renderer, the exclusive high-quality features that HDRP offers makes it a very suitable Pipeline for this project. The post processing features and the high rendering quality proved to be useful in designing and polishing the visual style of the scenario.

Unity offers the Shuriken System for all Pipelines, a physics-based particle generator, that calculates mostly on the CPU. The main advantage it has is the option for physics interaction of particles with the environment or each other. This allows simple fluid simulations that interact with the environment, for example, as well as triggering of collision events on any other collider object in the scene *(Unity Reddit, 2019)*. The huge disadvantage when creating high quality visual effects with Shuriken is its limitation on the particle count. With the energy net requiring a large number of particles that should be visible at the same time, this could cause performance issues.

The Visual Effects Graph, available in the HD Pipeline, however, runs on the GPU, which allows millions of particles to be rendered at once, while Shuriken can only handle up to about 400.000 particles *(Brackeys, 2019)*. The VFX Graph additionally has a system for setting and updating parameters at runtime, similar to how Animator or Shader Properties work. The node-based workflow makes a quick, iterative workflow possible and offers convenient overview of the systems functionality. The disadvantage of the VFX Graph is the missing physics and collision interactions with its environment, as well as the availability only in HDRP.

With its flexibility and convenient workflow, the Visual Effects Graph proved the more suitable tool for creating the particle-based energy grid of this project. The variety and adjustability in its use offered great value for the potentially changing design process of the energy grid system.

When working with multiple particle systems that should behave in roughly the same way, it would be very inefficient, if the developer is required to repeat his development steps for each system all over again. Instead it would be extremely effective if changes made to one system could be applied to all of them, while implementing some differences. Additionally, it would be convenient for a developer if all important variables could be set or even automatically generated in a particle systems interface.

This would be especially important for the already mentioned path curves of each particle system. It would take an absurd amount of time to manually tweak each curve, for each dimension, of each path, on each individual particle system. The value of a development tool comes from its ability to save a developer's time, by simplifying processes and optimizing the required workflow. That is why in this particular situation, a development tool that optimizes the design process and path creation of the particle systems is especially valuable.

A great way to quickly create tool interfaces in Unity is by using the *Odin Inspector*, developed by Sirenix. This plug-in can be used to quickly create an interface that a designer can work with and that supports the functionality of the tool. It enables the quick implementation of custom editor layout options by using simple keywords in the attributes of a variable or function *(Sirenix, 2019)*. The tool should focus on the main and most time-consuming task, which is the path creation, and without getting too complex in development, extract a path for the particles to follow. For this, it would be easiest to use handles that the designer can place into the scene and extract the path from.

The tool itself should have access to all necessary parameters of the particle system and be able to change them directly, even when in edit mode. This can be done with the *ExecuteInEditMode* attribute of the tool script's class.

3) How can an abstract visualization of an energy grid be designed, that feels organic and responsive and supports the scenario's core message?

An energy grid consists of power lines that connect junction points and transformer stations in throughout the grid, as well as the distributing network that connects the users to the power grid *(TenneT TSO GmbH, 2019, Projekte NEP)*. In the scenario, it can be nicely visualized by letting particles travel from a point A to a point B as representation of one connection. The grid then consists of multiple such connections, which connect central consumer areas with renewable and conventional power generators, through several junction points *(see Figure 3)*. A grid of connections with more than two path points, and therefore curved paths, can seem too organic and more like veins, rather than a network.

When the entire grid reacts to a new state simultaneously, it is hard to understand the origin of the change that was made. The grid feels stiff and not as dynamic and interactive as it should, even when it reacts quickly.

In order to make the grid feel more organic, a new state could be visualized as spreading through the grid originating from one or more specific points. This way a state could start on certain focus points and travel through all connections until it has reached all points in the grid.

Only a little delay in the spreading of states can already achieve a very dynamic effect and make the grid feel alive as it is dynamically reacting to changes. A too large delay and low spread speed might result in a very sluggish feeling, however, so the speed and delay need to be fine-tuned to assure an optimal balance.

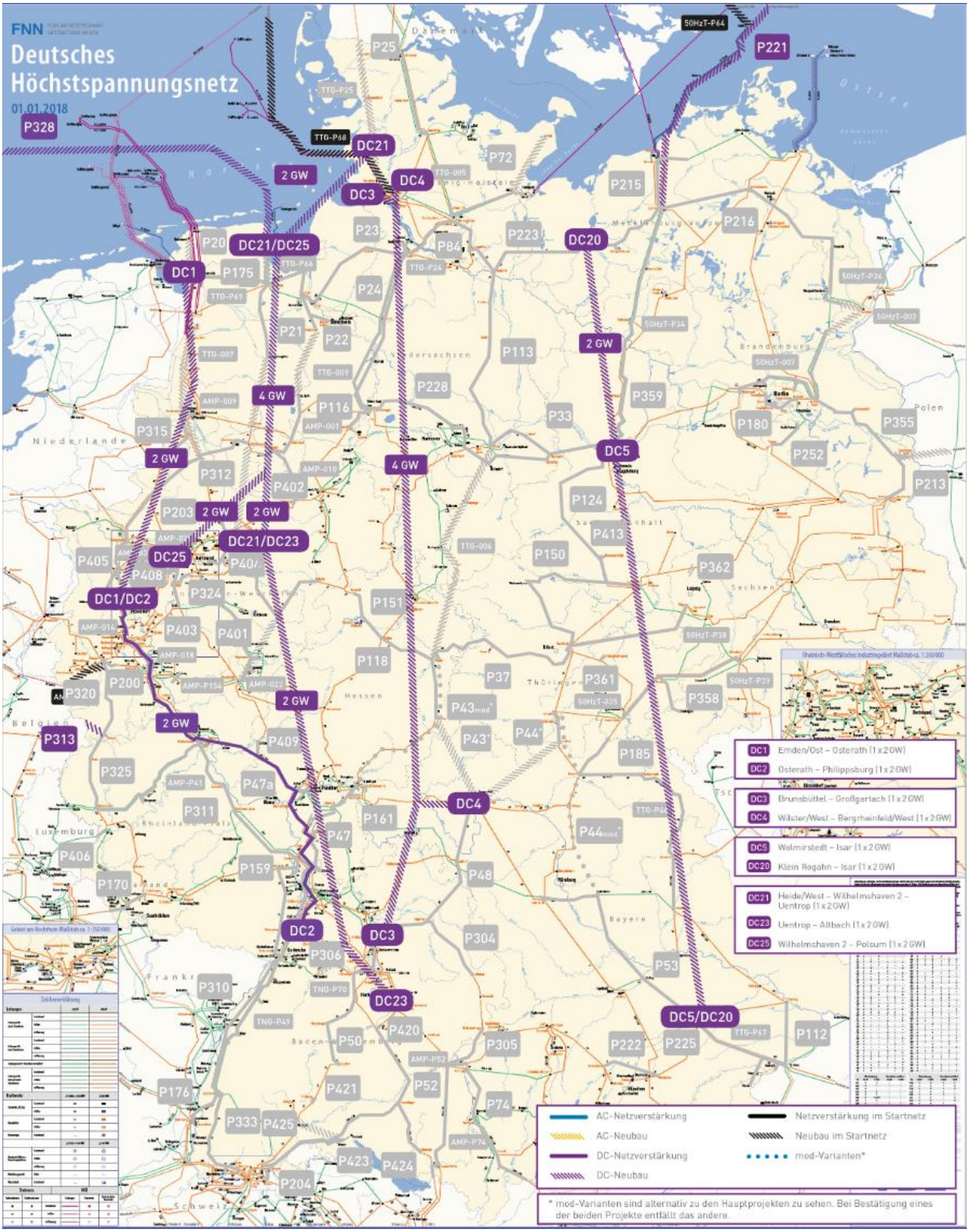


Figure 3: Grid Expansion Plan describing current and future energy grid expansion projects

Theory conclusions

After analyzing and researching these sub questions and reviewing the newly found knowledge, a much better perspective for further conceptualization and development was developed. The collected information and gained knowledge served as the basis for formulating a solution for the central problem.

It became clear that Unity's High Definition Render Pipeline is most suitable to be used to develop the scenario, due to the significant advantages of the HDRP-exclusive Visual Effect Graph, which could visualize the particle-based energy grid. It would be very efficient to create simple helper tools, especially for the design and development of the particle systems and their path creation.

The detailed terrain can be created very conveniently with Gaia, by using the Stamper Tool. With it, textures can be imported as heightmaps and stamped onto the terrain. The surface can be sculpted very accurately using this tool, ensuring the detailed style that the client desired. A simplistic and clean style should be reflected in all 3D models and textures.

Since the tree painter requires a very specific setup, the manual placement of trees would be more suitable. The number of trees required was not too high, and while placing the trees manually would be tedious, it offers more control over the trees' variations in the scene.

The moderate level of overall detail in 3D and environment art will leave room for more extensive visual development of the particle grid, without sacrificing performance on the mid-range specs of the target platform. Because of the low graphical load, regarding 3D models and textures, and since every object should be visible at the same distance and quality, using an orthographic camera setup, there is no need for LOD optimization. The effect would never be used anyway because of the scenario's static perspective.

6. Conceptualization

With the basis that had been established in the Knowledge Analysis and 360 Information Scan, a thought through and well-founded solution for the central research and development problem can be formulated.

The scenario content is structured in four different layers. Starting with the environment layer, which serves as a background for the scenario. It consists of the terrain and water surfaces, the textures and detail meshes on this terrain and the perspective onto the entire 3D environment.

The next layer will contain all grid components and buildings in the scene. These will be placed into the environment terrain. Grid components include renewable and conventional power generators, transformer stations and centers of power consumption.

The last visual layer contains the particle streams that will visualize the energy grid. The different particle systems will connect all grid components from the component layer and transport the particles, representing electricity, from the generators to the users.

The final layer consists of all the functionalities and mechanics, necessary for the scenario's components to interact with each other and react to input from the user. This includes scripts that will handle the state of each particle connection from the grid layer, as well as the visibility of buildings in the component layer.

These four layers together will form the interactive scenario. The detailed conceptualization for the development of each layer is described in the following. In it, a balance will be defined between the client's requirements on functionality, the desired visual quality and necessary measures of optimization, in regard to the technical capabilities of the target platform.

In a portrait orientation using an orthographic camera, the 3D environment will form a north to south contrast in the scene *(see Figure 4, perspective sketch)*. There will not be different LOD levels, since every object in the scene should be visible at the same size and quality.

The terrain will be created using the Gaia Terrain Generation Tool, more specifically its Terrain Stamp system. This allows importing custom heightmaps and stamping them onto the terrain dynamically. The great control over size orientation and height of the stamp makes this tool extremely valuable for rapid and precise terrain creation. Textures from the Gaia toolset as well as custom-made textures will be used to create a detailed, but simplified environment, which will support the scenario's main components with a visually pleasing background while not drawing attention away from them. Abstract visualizations of simple trees will frame the components nicely into their environment. These trees will have to be manually placed, in order to achieve the desired variation.

The Components will be modelled in a simple, abstract style, with minimal texture detail in white and grey colors. The Components should separate themselves from the detailed and colorful background, with their very different, clean and light style. All these Components will be placed into the terrain *(see Figure 4, grid layout sketch)*, with simple decal foundations that will let the buildings blend into the environment a lot better.

Using the Visual Effects Graph, the energy grid will be visualized by multiple connections between central points in the scene. The particle systems will connect generators and users to connection points and the grid components. In these systems the particles will follow a path, defined by several points for each grid connection. The entire particle grid will be able to change color and spawn frequency, to visualize power balance and volatility status, and react to changes of these variables at runtime.

When the state of the energy grid changes, the new state should be visible at the change's origin and travel through the grid over time.

In the graph, parameters will be predefined and used internally in the system. The graph will allow as much external input as possible, in order to optimize the development workflow and runtime flexibility.

The grid will react to changes at predefined junctions and will spread the new states through the system dynamically. In order to visualize multiple states traveling through a connection at the same time, animation curves will be used to update the color and frequency of the connections in any direction, while the particles still travel in their normal direction. When a new state is registered, it should be passed to the system as a keyframe on an animation curve. This keyframe is then moved every frame until the state has reached the end of the connection. At each junction there will be checkpoint objects that all connections will register as either start or end point. When a connections state has finished, the checkpoint on the finishing end of the connection should receive this state and pass is to all other connections registered to this checkpoint. This way a state will initially be registered on one end of the grid and then travel through the connections and junctions to the other end. The Checkpoints, state events and keyframes moving through an animation curve, together will form the StateShift system.

The states themselves will be visualized by color and frequency of the particle system. A good power balance will be displayed in green particle color, an unbalance in red color. Volatility will be visualized as gaps in the particle stream, where almost no particles will spawn. A high volatility will result in a higher frequency, a low one in a lower frequency with smaller gaps. The power and volatility state will be passed to the particle system in animation curves as already described. These curves can then be sampled by the particle's age over its lifetime. Depending on the power state, the particle then samples from a gradient that determines at which state the particle should have which color. The volatility is used as input for the gap frequency, which can be sampled using a sine wave. The gaps in the particle stream, where fewer particles spawn, should have a lower power level and therefore color than the normal parts of the stream. To achieve this, depending on the current volatility level, a small amount will be subtracted from the power sample and used to sample the gradient for the gap's particle color.

With this setup, a dynamic grid will be achieved, that feels more organic and reactive than a grid that reacts to changes in unison. The visual feedback will be more appealing and keep the viewers' attention, as well as encourage users to try and see what different effects certain changes to the grid will have.

To test the scenario's interactive functionalities, an Input System will be developed by the programmer, that will mimic the input capabilities of the actual target device, which is not available during development. This way the scenario can be tested, and feedback can be discussed with the client using a convenient desktop version of the application.

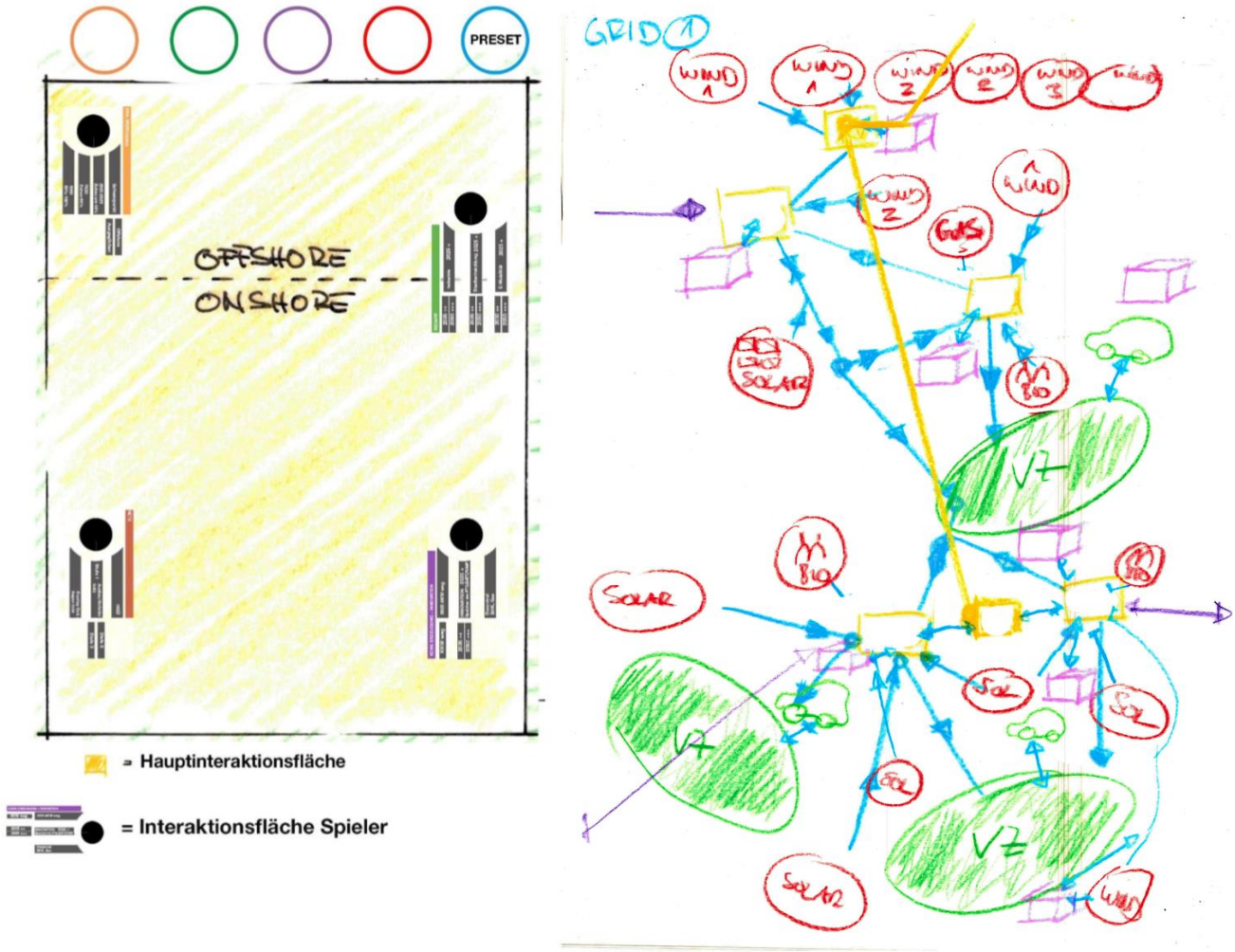


Figure 4: Perspective and grid layout sketches provided by the client

7. Development

The development consists of two main categories, 3D and technical. 3D development includes the creation of 3D assets, such as the buildings and grid components, the underlying terrain and environment, and the reaction of components to changes in the scenario, by displaying a specific status color or changing visibility. The technical development involves the designing of the particle system using the VFX graph and developing an interface for it, as well as creating all required functionality for the grid to react to changes in the scenario. Additional functionality like the user interface and the event system that defines the scenario's current state, depending on the user's input, will be developed by the programmer.

The process of developing the scenario is structured into several iterations, starting with a mockup and prototype. Here a basis for style perspective and overall functionality is developed, which will be extended in the Alpha iteration. Finally, in the Beta iteration all additional functionality is added, and visuals are adjusted to achieve the desired style.

After each iteration, the progress was discussed with the project lead and occasionally the client. Feedback and possible additional requirements are then described in the reflection of each Iteration. Few feedback moments were possible, due to the client's schedule, however constant updates were made, and the results discussed whenever possible. This was roughly at the end of the mockup and alpha iterations.

Iteration Mockup

In this first iteration, after a basis for pipeline and workflow had been established, a mockup could be worked out. This mockup should show a first take on the desired style and perspective of the scenario.

First, based on the simplistic, low poly style direction, a few prototype assets were created. The 3D models only had basic detail and the textures were flat colors mixed with ambient occlusion.

Figure 5: Mockup draft based on minimalistic style reference

A couple of buildings and trees were placed onto a basic Unity terrain with simple grass textures. After adding the Unity basic water to the scene, the minimalistic “miniature” environment style was defined *(see Figure 5)*. The perspective is done by using an orthographic camera, which makes objects, that are farther away, appear at the same size as objects closer to the camera. This way, the available screen space could be used to its maximum, without sacrificing detail in the distance *(see Figure 6)*.

The terrain in the mockup was done by simply lifting the surface slightly in the southern parts of the environment, since that is where the mountains should be visible. In the north a shore and simple water plane was added. The trees were done very simple, again in flat colors with ambient occlusion.

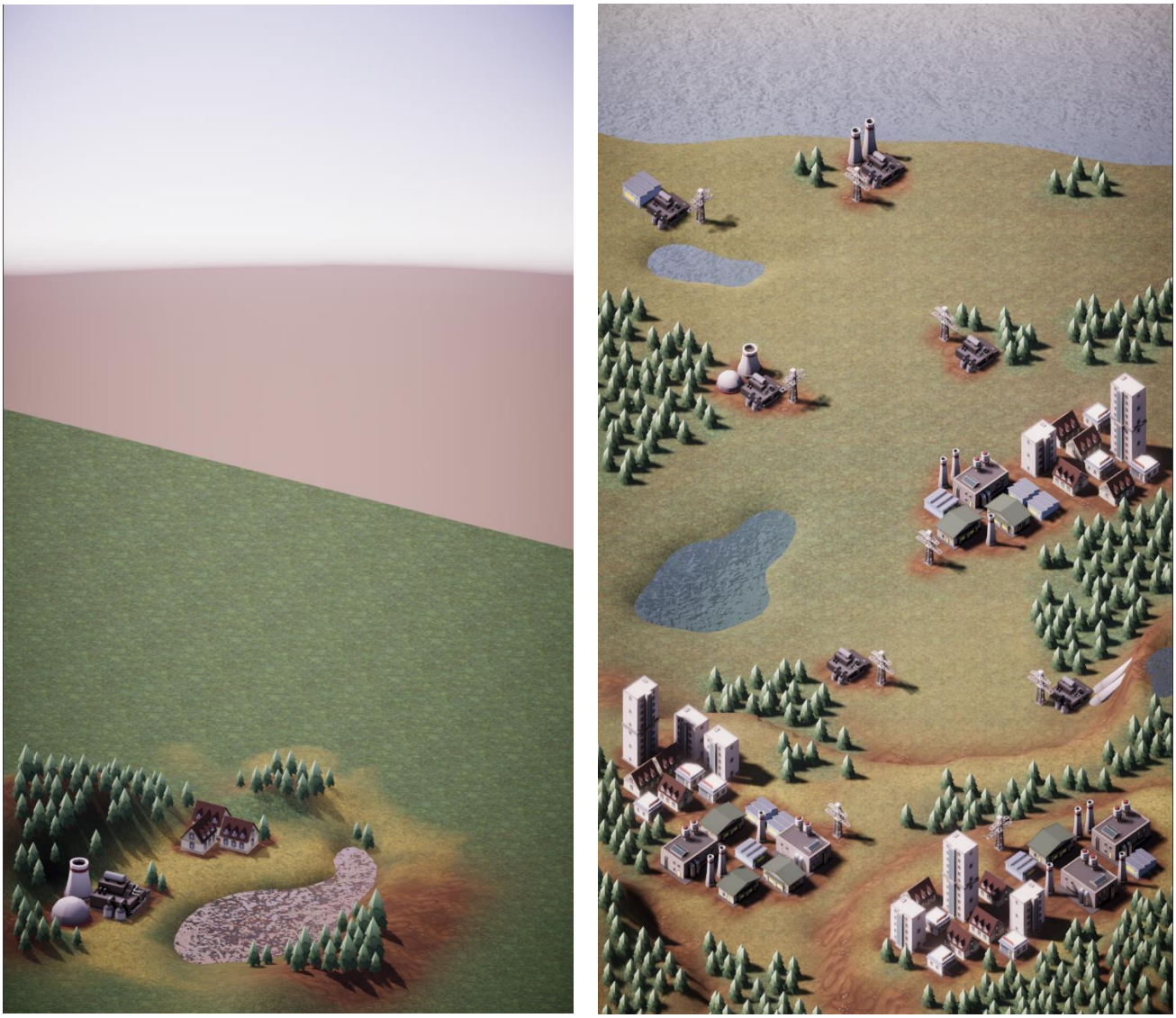


Figure 6: Early perspective test and improved full screen mockup

Mockup Reflection

The perspective proved to be very suitable right away. The framing was good and additional changes could later be done by decreasing the size of all other objects in the scene.

The terrain was lacking a lot of detail in structure and the textures should be more detailed in further iterations. Overall the environment was way too empty and blunt. The buildings needed to be smaller, so that more would fit into the scene.

The simplistic style was already going in the right direction but was getting slightly too childish and the buildings started to look like toy houses. To make the building style cleaner, the textures should be gray scaled, while keeping the ambient occlusion detail. All buildings would then follow the same color scheme and appear more separated from the underlying terrain. This way the components could be made more visible in contrast to the environment style.

Iteration Prototype

The next step was to create prototype versions of all the basic interaction and reaction mechanics, as well as the particle system visualizing energy that travels through the power grid. Additionally, the layout of the grid was thoroughly designed, so that the size and density of all the scenario's components would become more defined.

To start the prototype iteration, a first version of the particle system connection was developed, using the VFX graph. The system used a start and end point and let particles travel in between. This is done by using the particle's age, which ranges from 0 to 1. Each particle has a given lifetime, and after it spawns the age in seconds increases. When the age in seconds reaches the maximum lifetime, the particle is destroyed. The time since spawn divided by the total lifetime determines the particle's age. Using linear interpolation, any position in between the start and end point can be sampled over the particle's age. This way it travels at a constant speed from start to end throughout its lifetime. The particle movement presents the foundation of the energy grid system.

In the prototype version, the particle system was only reacting to a locally set power state, which was visualized by the system changing colors from red to blue. This way the reaction to changes could be tested. A system for globally distributing information about the current state of the scenario was still missing.

The buildings were gray scaled and reduced in size and three central areas were defined where the buildings should be denser, as to represent centers of energy consumption.

An early prototype version with a basic grid visualization was used as a basis to design a more finalized layout of the grid. In a brainstorm meeting the different layers of the grid were drawn onto transparent paper with the base in the background. Layer by layer the layout got more complete and after combining everything in Photoshop afterwards, this sketch could be used as reference for further building the scenario's power grid and placing all objects into the scene correctly.

Prototype Reflection

The central areas, where more buildings were close together to form a center of energy consumption, were already in the right direction but needed to be even denser and contain additional buildings. The buildings should be smaller and closer to each other so that the centers would seem more like a city, rather than just a collection of buildings.

The terrains texture and surface were still too simple. More detail needed to be added especially through additional textures. Surface structure easily got lost due to the lack of perspective in the orthographic camera setup, so major differences in height would hardly be visible, if not supported by textures and careful tree placement for example.

The trees themselves were also still too close to the rather childish mockup style and should be enhanced as well. Overall the terrain needed a lot more attention to detail, especially on the seashore and the mountain side. The environment should have a more realistic and detailed style, so that the components become more visible.

Iteration Alpha

The grid connections were reacting to a given power and volatility value. The power state was visualized by the particle's color, blending between blue and red, the volatility was visualized by gaps in the particle stream, which would change in size and frequency, depending on the volatility value.

In order to let the particle systems react to external input, a system was created, using scriptable objects, that would update any scripts that were registered whenever the scenario's state was changing. The scenario's state consists of a power and a volatility value, and in the net data object an event system was added, which calculated the current power and volatility states, depending on the user input. A script holding reference to this net object could then register to this event system and was notified on changes when the state was updated.

For editing parameters in the VFX graph, especially at runtime, a script was created that serves as an interface for the particle system's exposed values. This offered the possibility to create additional functionality in the VFX handler script, that would take external input, transform it into the required parameters and pass it to the particle system. As an example, when updating the power value in the script, it would translate that into a color value and pass it to the particle system. This reduced the amount of operations that had to be done in the particle system itself and ensured a higher flexibility in development. When having to adjust parameters or functionalities it was therefore not necessary to open and alter the VFX graph, which takes more time than simple adding functionality in script.

To manage and quickly adjust parameters of the particle effect, scriptable objects were introduced, which could hold different sets of parameter presets. A scriptable object is a class, which can exist as an asset instance in the project files. This asset can be used globally, meaning if it is changed anywhere in the project, it is changed for every script or other object that holds a reference to it *(Unity Technologies, 2019, Unity – Manual: ScriptableObjects)*. Multiple instances of the same scriptable object can be created, which can hold different values for each variable. This means that, if the VFX handler uses parameters from a scriptable object, if it is adjusted, the parameters of all other particle systems are adjusted as well. This way every particle system can be altered simultaneously, instead of having to adjust every system individually. Additionally, several different behavior presets can be created, using multiple instances of the behavior object script. When changing one behavior, only the particle systems using the same behavior will be updated.

The VFX reactor registers to the net data event and updates the VFX handler's power and volatility values. In order to have maximum control over the particle's taken path, the PathLink system was introduced. The VFX handler takes in references to several points that can be placed into the scene. It can then extract a 3D path from these points and pass it to the particle system.

The path extraction considers where each point is positioned in relation to the total distance of the path. This prevents the particles from speeding up or slowing down irregularly while moving along the path. The positions of the path points are then passed into three animation curves for each dimension. An animation curve takes in Keyframes, which have a time value (normally ranging from 0, the start of the curve, to 1, the end of the curve) as well as a certain

individual value that can be sampled by evaluating the curve at the keyframe's time. The keyframe specific value is one dimension of the path point's position (x, y, z), since one curve can only hold information about one axis in 3D space. That is why three curves are necessary to store a 3-dimensional path. The curves are then passed to the particle system by using the corresponding identifiers, defined in the VFX graph's parameters.

In the graph the three Curves are each sampled using the particle's age over lifetime. That means at the beginning of its lifetime the particle would sample the path at t = 0, at the end of its lifetime at t = 1. This way, each particle individually samples the path at a consistent speed. The lower the lifetime was, the faster the particle would travel along the path. After sampling each Curve at the particle's current age, the three dimensions are merged back into a position in 3D space and used as the particle's current output position.

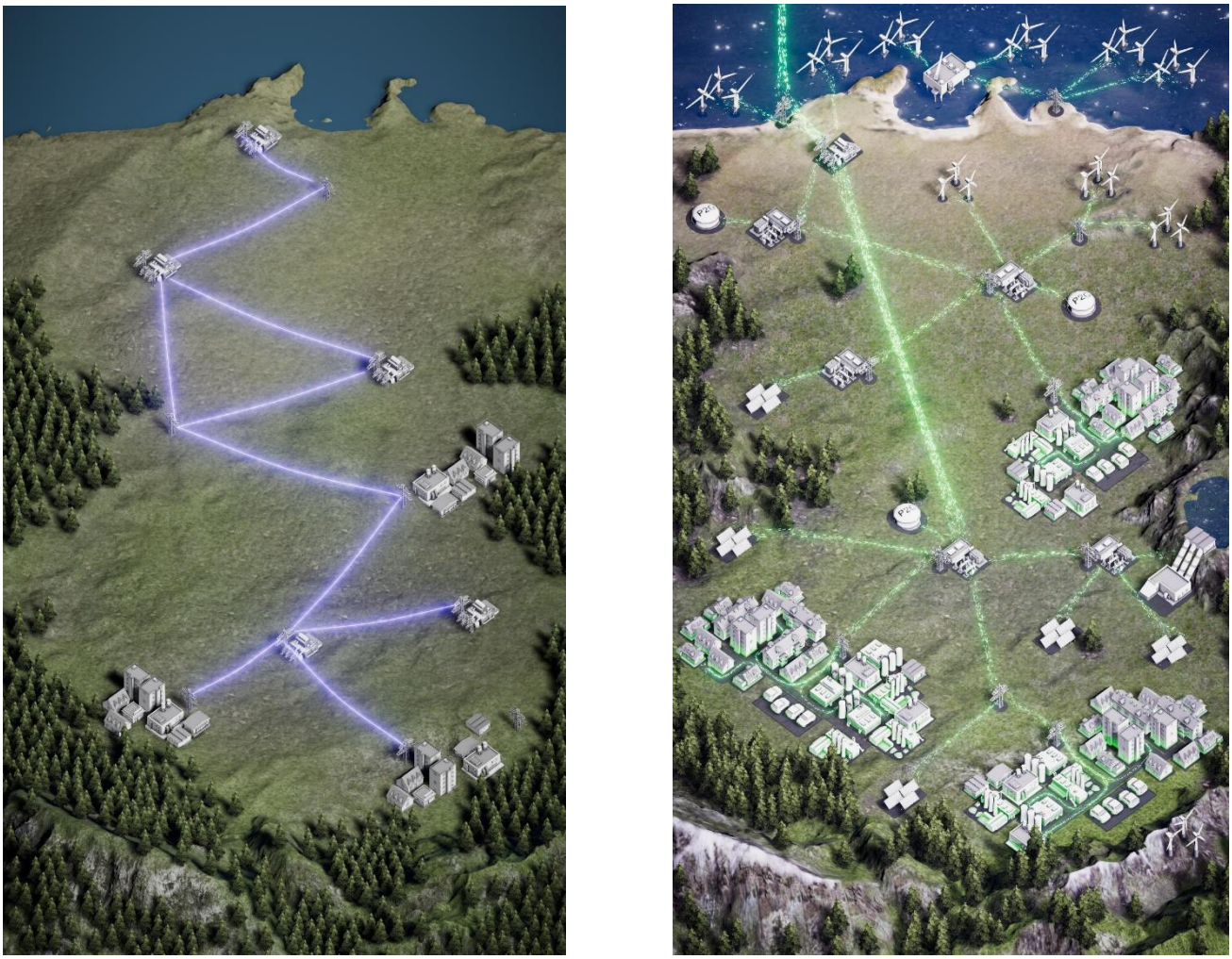


Figure 7: Terrain rework with more surface and texture detail

After the particle system was drastically improved, the terrain needed to be reworked. It was lacking a lot of surface and texture detail. In order to get a quick but high-quality result, the Gaia Terrain Tool was used. The terrain stamper enabled very fast structuring and sculpting of the terrain and it was easy to create a mountain range in the south, as well as an interesting seaside in the north of the environment. Using the standard textures provided by Gaia and several custom textures, made with Substance Designer, the terrain was textured with more detail *(see Figure 7)*.

Instead of the low-poly trees, detailed ones were added *(see Figure 8)*. A problem was that, because the project uses the HD render pipeline, many terrain creation options were not working anymore. The tree painting was also affected by this. For a terrain to recognize an object as a suitable tree, the object needs to have specific materials applied, which are incompatible with HDRP. Therefore, while using HDRP compatible materials, the terrain couldn't use any of the tree specific functionality and the trees could only be painted onto the terrain without the random rotation, height and color hue modifiers.

Figure 8: Comparison of old low-poly trees to the improved detailed version

Alpha Reflection

PathLink proved highly efficient during development. The otherwise tedious and time-consuming task of manually setting up the curves in the VFX graph itself became very convenient with this simple tool. It was possible to make minor adjustments and design the path a lot more accurately than it would have otherwise been possible.

In order to make the grid react in a more organic way, the changing of color and frequency for different power and volatility states, should gradually spread through the grid, instead of changing simultaneously throughout the entire network. Otherwise the grid would feel a little too stiff. The colors should be changeable to blend from red, through yellow to green, to make the power state easier to understand. The gaps representing different levels of volatility were disconnecting the particle stream too much. A clear, continuous stream should always be visible. The volatility gaps should therefore consist of portions with more and gaps with fewer particles, so that the stream is clearly visible at any state, but differences in frequency can still be observed.

Although the terrain was a lot closer to the originally intended style, it was still lacking some interesting detail. It should be more than just a monotone background and give the viewer something pleasant to look at. Due to the restricted use of the tree painter terrain tool, the trees all had the same size, rotation and color and therefore were extremely repetitive. This should be fixed by either manually placing, rotating and scaling the trees, or by creating a specific tool for randomized tree positioning.

The building models were overall too dark, since the textures were only gray scaled. To lighten the appearance of the components and make them more distinguishable from the environment background, the textures should have a white base color and light gray detail on some surfaces. Together with the ambient occlusion detail this would create a simplistic and light texture style. Additionally, the buildings were not connected to the environment enough. In order to make them blend in with the underlying terrain textures, decals should be used that would project a foundation texture on to the terrain.

The sizes of the buildings were now very suitable. Even with a larger amount of buildings, the environment would still not be oversaturated. However, only few buildings were currently used in the scene, so by using the layout reference, in the beta iteration the grid should now be built completely. This way it would become clearer how much space all the necessary components would cover on the screen.

Lastly, a water shader should be implemented, to replace the flat colored water planes, used as placeholders. The water shader should offer depth and transparency functionalities, as well as simple wave animations and foam. It should match the overall detailed style of the terrain environment.

Iteration Beta

To start the Beta Iteration, which should result in a fully functioning version of the scenario, the building textures were adjusted to the new light gray style. The centers of power consumption were built out of industry, city and automobile components. The rest of the terrain was filled with renewable and conventional energy generators, as well as junction points, that would serve as a base for the particle system grid *(see Figure 9)*.

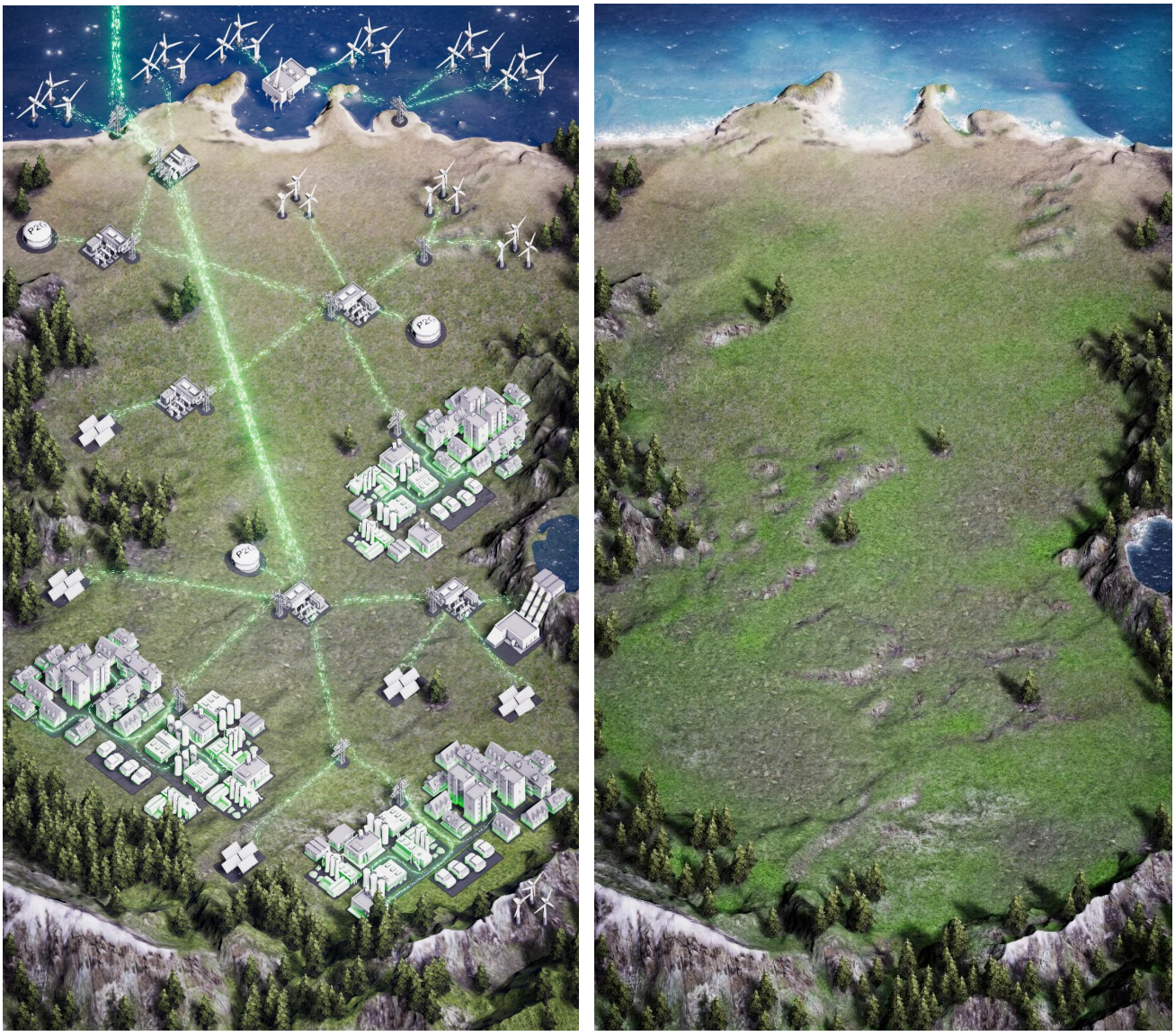


Figure 9: Improved grid layout and environment design

The appearing and disappearing of components was improved. Instead of animating the buildings to disappear into the ground, they were now dissolving from top to bottom of the object and would reappear the same way. This was done with a simple dissolve shader, using the objects height and then cutting off the objects mesh above a certain threshold. If the threshold was de- or increased over time, the object would gradually dissolve or reappear *(see Figure 10)*.

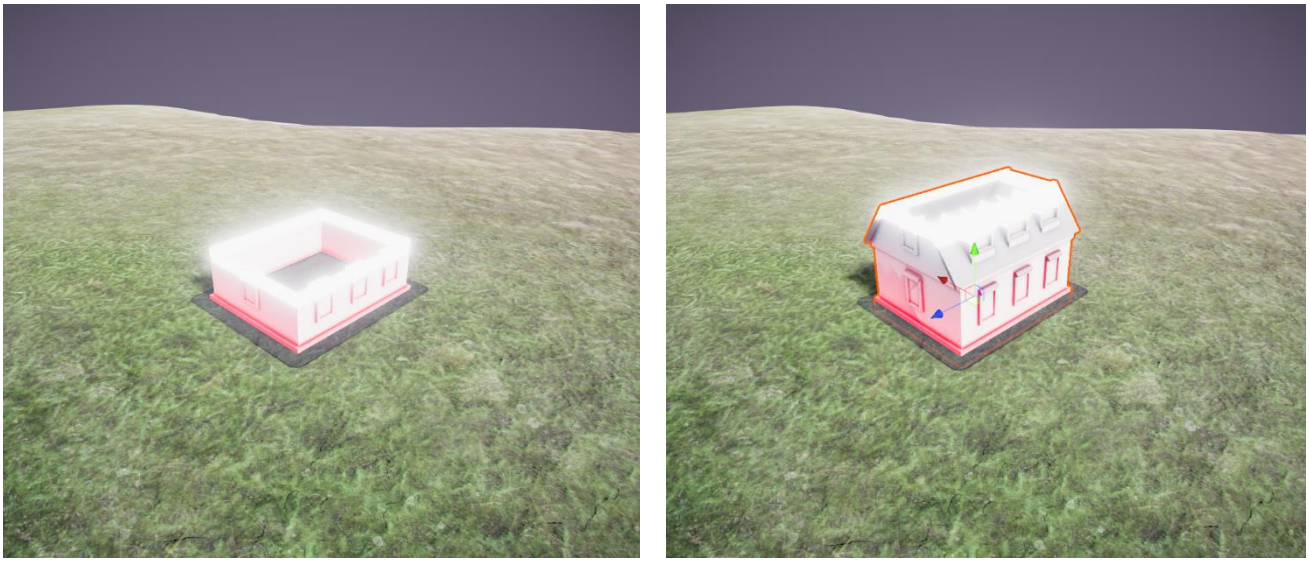


Figure 10: The material shader cuts off the geometry above a certain threshold and displays glowing edge

The structure of the terrain was already fitting, however the texture could still be improved. Sand textures were added for further defining the seaside, and cliff textures helped give the mountain range a more refined shape. Through blending multiple textures together, a nice gradient could be established from north to south, as to add variation to the supporting environment. In order to connect the buildings to the terrain a little better, the decal projector, that is available in the High Definition Pipeline, was used. After creating simple textures for the building foundations, these could be projected onto the terrain very easily. There is no convenient system to change the terrain's textures at runtime, so the projection of additional textures was very helpful in this situation. The building materials could be configured to ignore these decal projections, so that the foundation textures would only appear underneath the buildings on the terrain *(see Figure 11)*.

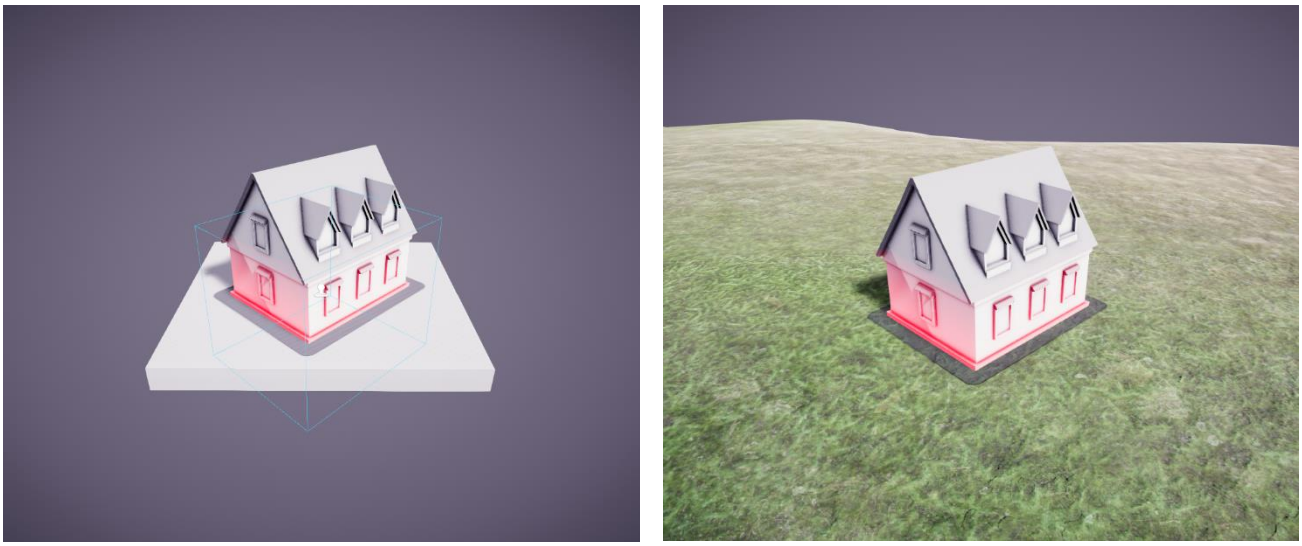


Figure 11: Decal Projector that created a texture overlay on any material that reacts to decals

The water systems in the standard unity environment resources did not fulfill the requirements, and most external water shaders were incompatible with HDRP. A custom water shader was therefore developed using the Shader Graph. This allowed a more flexible

design of the water surface and the possibility to add foam around the shore. For this, both a wave and a foam texture was created in Substance Designer. In the shader, the textures were sampled multiple times with different offsets over time. When blended together this created an animated effect for waves and foam detail. With additional parameters for intensity, size, speed and color for both waves and foam, the water could nicely be designed to fit the scenario's style *(see Figure 12)*.

Figure 12: Improved water shader with animated wave and foam textures

The particle system underwent a last major change. In order to make it react in a more natural way and feel more organic and responsive, a change in the scenario's state should start to take effect at specified locations in the grid and then travel through the network. The problem was that multiple states could move through a connection at the same time, even in opposing directions. The more recent state should always override the older one and these different states would somehow have to be passed to the particle system, so they could be displayed. When having to handle multiple states within the same system, and in order to keep the complexity of the VFX graph to a minimum, it was most suitable to pass the power and volatility states to the system using animation curves. Similar to the path curves, the state curves would represent a connections length from 0 to 1 on their timeline. Different states could also be handled as values between 0 and 1 on the altitude of the timeline. Each new state travelling through a connection was therefore represented by a keyframe starting on one side of the state curve and being moved frame by frame towards the other. The amplitude of the state keyframe could be sampled by the particle system and particles could then determine their individual power and volatility values at their given position on the path.

In the scene, checkpoints were added which had a checkpoint script attached. The checkpoint script has an event system that, when receiving a new state, pushes it through an event function. Any particle connection that is registered to the checkpoint receives the new state, except the one that originally pushed the state. Each connection has a queue of states that are currently running through it. Each state in this queue has a certain power and volatility value as well as a specific time that locates it on the connections path and most importantly an individual age. The age of a state increases every frame and this age is also passed on when the state reaches the end of a connection and is pushed to the checkpoint. This way two states can be compared and the younger state can be determined. When two states running in opposite directions pass each other, the older state is removed from the queue and only the younger state remains. While there is still a state in the queue, the updated state curves for power and volatility are passed to the particle system each frame. When the queue is empty, the state shifter returns to idle and is only reactivated when a new state is registered.

The VFX handler of each connection has the option to react to changes of the scenario's state. The direction a new state should travel can also be determined. This way, if the scenario's state changes, by changing the input of a specific source, the grid will start to react at the corresponding location. For example, when reducing the amount of power generated, the grid would turn red, starting at the generators and eventually moving towards the users.

Beta Reflection

With the StateShift functionality, the grid was reacting in a responsive and organic way as intended. Having the states travel through the grid made it easier to locate where the change is coming from and made the grid feel a lot more dynamic. The system was stable and reacted well, even when the scenario state was changed multiple times in short succession. The PathLink tool proved extremely useful when making quick adjustments to the grid layout and the state shifter ensured a satisfying functionality of the grid.

The terrain now looked a lot more detailed and had a nice shoreline, which worked very well with the custom-made water shader. The environment style was now closely resembling the original concept, where the terrain had a very realistic style and high surface and texture detail. The clean, neutral white buildings created a fitting contrast to the green and blue background of the terrain and seaside. The cities were framed nicely by a moderate number of trees which had some variation in height and rotation. All components were now blending a lot better into the background terrain, after adding decal projectors to every building *(see Figure 13)*.

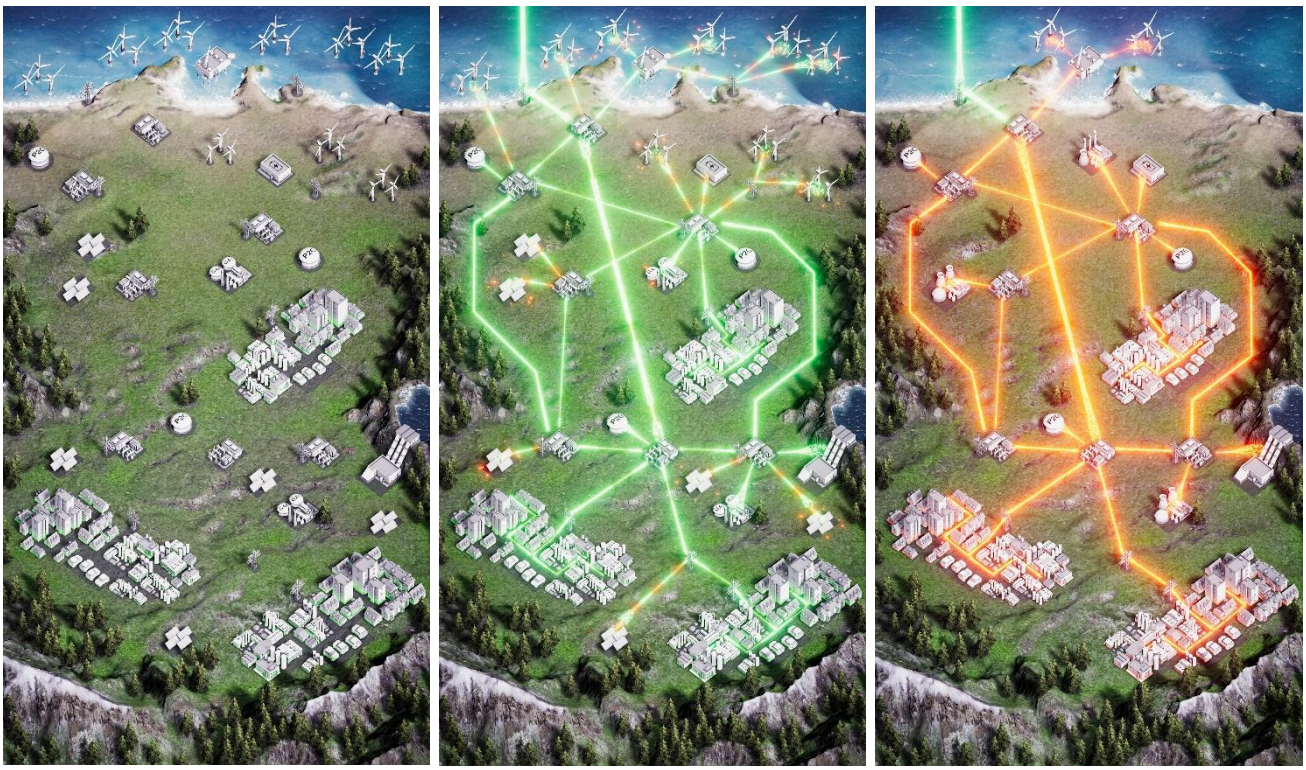


Figure 13: Final scenario content with all components and the power grid with surplus and shortage

7. Conclusion

Although the project was heavy on actual development in the later stages, its successful completion was ensured by a solid development basis, a thought through workflow and a clear understanding of the required pipeline and functionality for the scenario to work as desired.

Especially with the little time available for the project, it was vital to create an optimized workflow and to have a clear view on what was required in functionality and style and how the desired result could be achieved. Analyzing the different advantages and restrictions and then selecting the most suitable pipeline prevented obstacles during later development, because few unforeseen problems arose. It was important to define at least the basic requirements for all the scenario's components in order to design an efficient workflow. While 3D art and environment design were less of a problem regarding development efficiency, the creation of the particle grid certainly was. Without a proper understanding of how the particle system could be visualized and what functionality it would have to offer, it would have been a highly inefficient development process.

The terrain could be designed very well using the Gaia Terrain Tools. The terrain stamper and texture assets were useful for sculpting and texturing a nice and detailed terrain background. The price of Gaia was justified by its value during development and choosing the external software was helpful. The trees had to be manually placed into the environment, but since the tree count was kept quite low this proved to be a very accurate and convenient way to fill the terrain. However, a bit of time could have been saved by using the Unity Terrain tree painter, had it worked properly in HDRP.

The available rendering pipelines could easily be compared and, through analyzing the advantages and restrictions, the High Definition Pipeline could quickly be defined as the most suitable one. Even though the graphical load was higher than in the LWRP because the VFX graph was used, along with HDRP rendering and some post processing, the performance was very balanced, because most graphical aspects had been optimized and simplified, as to fit within the technical limitations of the target device. The Lightweight Pipeline would offer a higher graphical performance even on less powerful systems, however in this situation if would have prevented the use of the VFX graph and the Decal Projector, both of which were vital features for developing the scenario.

As mentioned earlier, the target device was unfortunately not available during development, due to the touch table being in possession of the client in Berlin. Because if this, the input simulator proved very useful, by recreating the intended user input as best as possible. Adapting to this situation early on enabled the development and testing to continue without significant delay and therefore little time was lost.

The resulting scenario fulfills the requirements of the client and stays within the technical limitations, regarding the graphical load. Environment and Components fit together well even though they were created in different styles. The terrain presents a fitting background that doesn't draw the user's attention away, but rather supports the scenario's content. The interactive layer handles well and makes the users interaction with the scenario feel responsive, due to the grid reacting in a dynamic way.

To conclude, within a short amount of time, a suitable pipeline could be defined and a corresponding workflow formulated. This ensured a highly time efficient development of the scenario by selecting suitable tools and optimizing the use of particle systems, using the VFX graph and the interaction with its exposed parameters.

8. Discussion

The established workflow and the fact that, with it, is was possible to complete a well-structured development in a short period of time, shows how valuable a thought through basis is for a time-restricted production process, such as in this project.

In projects with a lot of time pressure it becomes even more vital to establish a suitable workflow early on. With the correct focus, this can quickly be achieved. Realizing problems or obstacles throughout development might be a more common situation. However, having to change an entire pipeline in the middle of a project, due to predictable problems that would have been obvious, had pipeline been thoroughly analyzed, is an unnecessary waste of valuable development time.

Creating an efficient workflow does not mean that specific development tools need to be created or an entire custom pipeline needs to be developed. Simple things like having a sufficient understanding of an engine's or tool's capabilities or collecting knowledge about best practices or common existing workflows can already have a huge impact on the development process. These things will therefore not have to be determined by trial and error during development, which will save a lot of time and additional, unexpected workload. This means that even with little time it is more than possible to establish a workflow that will aid development and make the process more efficient.

It was very helpful to have a conceptual basis of what the scenario would have to include and how each component was to be developed. Already thinking about how the visual requirements could be fulfilled within the technical limitations and software restrictions proved very valuable during later development, since there was no need for discussion about which tool would best be suited for a particular feature. There were no unpleasant surprises, like incompatible tools or pipelines, since all the possible issues had been addressed ahead of the development.

Although it would have been nice to have a more refined custom pipeline for the development with particle systems, the time it would have taken to develop the necessary tools and interfaces for this would have probably been significantly higher than the time it could potentially have saved during development.

9. Recommendations

Based on the result of this research and development project, several recommendations can be given regarding the work with interactive particle systems and the value of a well-defined workflow.

It is highly recommended to spend time ahead of development to analyze potential tools that can be used to create different features and components of the project. The most suitable pipelines and tools can be selected, and an optimal workflow determined. Additionally, all the potential issues and problems, due to incompatibility of certain tools, for example, are already known and can be overcome more easily. This takes a lot of stress out of the development process, since the unexpected, unpleasant surprises will be held to a minimum.

Creating an interface for the VFX graph particle system proved extremely convenient for development. Even if little extra functionality is desired, a system to interact with the parameters through script makes the development process a lot easier, since the scripts' functionalities can quickly be altered or extended, whereas the graph's parameters can only be changed as they have been defined and exposed in the graph itself.

In script, different behaviors can be managed, by using scriptable objects, with different sets of parameter values. This proved very efficient when different versions of the particle effect needed to be created. Quickly exchanging behaviors was a lot easier than creating a new prefab version.

Without the option to test the scenario on the actual hardware it was developed for, an input simulator had to be implemented that recreated the interaction with the touch tables input tokens. This was a great way to test the scenario's functionalities without having to wait for an opportunity to test on the target device. User feedback from test sessions on the actual device would be extremely valuable in the future, regarding possible changes and polishing of the scenario's interactive component.

10. References

Unity Terrain Tools

Procedural Worlds. (2019). *Introducing Gaia*. Retrieved March 2019 from http://www.procedural-worlds.com/gaia/

Procedural Worlds. (2019). *Stamper Introduction*. Retrieved March 2019 from http://www.procedural-worlds.com/gaia/tutorials/stamper-introduction/

Procedural Worlds. (2019). *Spawners – Terrain Trees*. Retrieved March 2019 from http://www.procedural-worlds.com/gaia/tutorials/spawners-terrain-trees/

Unity Technologies. (2019). *Unity Terrain – Getting Started*. Retrieved February 2019 from https://blogs.unity3d.com/2018/10/10/2018-3-terrain-update-getting-started/

Unity Technologies. (2019). *Unity Terrain Trees*. Retrieved February 2019 from https://docs.unity3d.com/Manual/terrain-Trees.html

Unity Technologies. (2019). *Unity Terrain Engine*. Retrieved February 2019 from https://docs.unity3d.com/Manual/script-Terrain.html

Particle Systems in Unity2018

Anthony Uccello, Eric Van de Kerckhove. (2018). *Introduction to Unity Particle Systems*. Retrieved February 2019 from https://www.raywenderlich.com/138-introduction-to-unity-particle-systems

Unity Technologies. (2018). *Shuriken Particle System Documentation*. Retrieved February 2019 from https://docs.unity3d.com/ScriptReference/ParticleSystem.html

Unity Technologies. (2019). *Visual Effect Graph*. Retrieved February 2019 from https://unity.com/de/visual-effect-graph

Unity Forum Discussion [Reddit]. (2019). *Shuriken (Particle System) vs VFX Graph*. Retrieved February 2019 from https://www.reddit.com/r/Unity3D/comments/afvlog/shuriken_particle_system_vs_vfx_graph_when_to_use/

Brackeys [YouTube]. (June 2019). *How many particles can Unity handle?*. Retrieved from https://www.youtube.com/watch?v=0deXRHX9C08

Unity Editor Interface

Sirenix. (2018). *Odin Inspector*. Retrieved March 2019 from https://odininspector.com/documentation/sirenix.odininspector.assetlistattribute

Scriptable Render Pipelines in Unity2018

Unity Technologies. (2019). *Lightweight Render Pipeline*. Retrieved February 2019 from https://unity.com/de/lightweight-render-pipeline

Unity Technologies. (2019). *High Definition Render Pipeline*. Retrieved February 2019 from https://blogs.unity3d.com/2018/03/16/the-high-definition-render-pipeline-focused-on-visual-quality/

Unity Technologies. (2019). *Decal Projector.* Retrieved April 2019 from https://github.com/Unity-Technologies/ScriptableRenderPipeline/wiki/Decal-Projector

Unity General

Unity Technologies. (2019). *Unity – Manual: ScriptableObject*. Retrieved April 2019 https://docs.unity3d.com/Manual/class-ScriptableObject.html

Unity Technologies. (2019). *Unity – Manual: LOD Group*. Retrieved March 2019 https://docs.unity3d.com/Manual/class-LODGroup.html

Power Grid Structure and Development

Copenhagen Economics. (2018). *Netzentwicklungsplan für die Zukunft*. Retrieved March 2019 from https://www.tennet.eu/fileadmin/user_upload/Company/News/German/Hoerchens/2018/TenneT_study_clean_format_10JUNE2018_vrs10.pdf

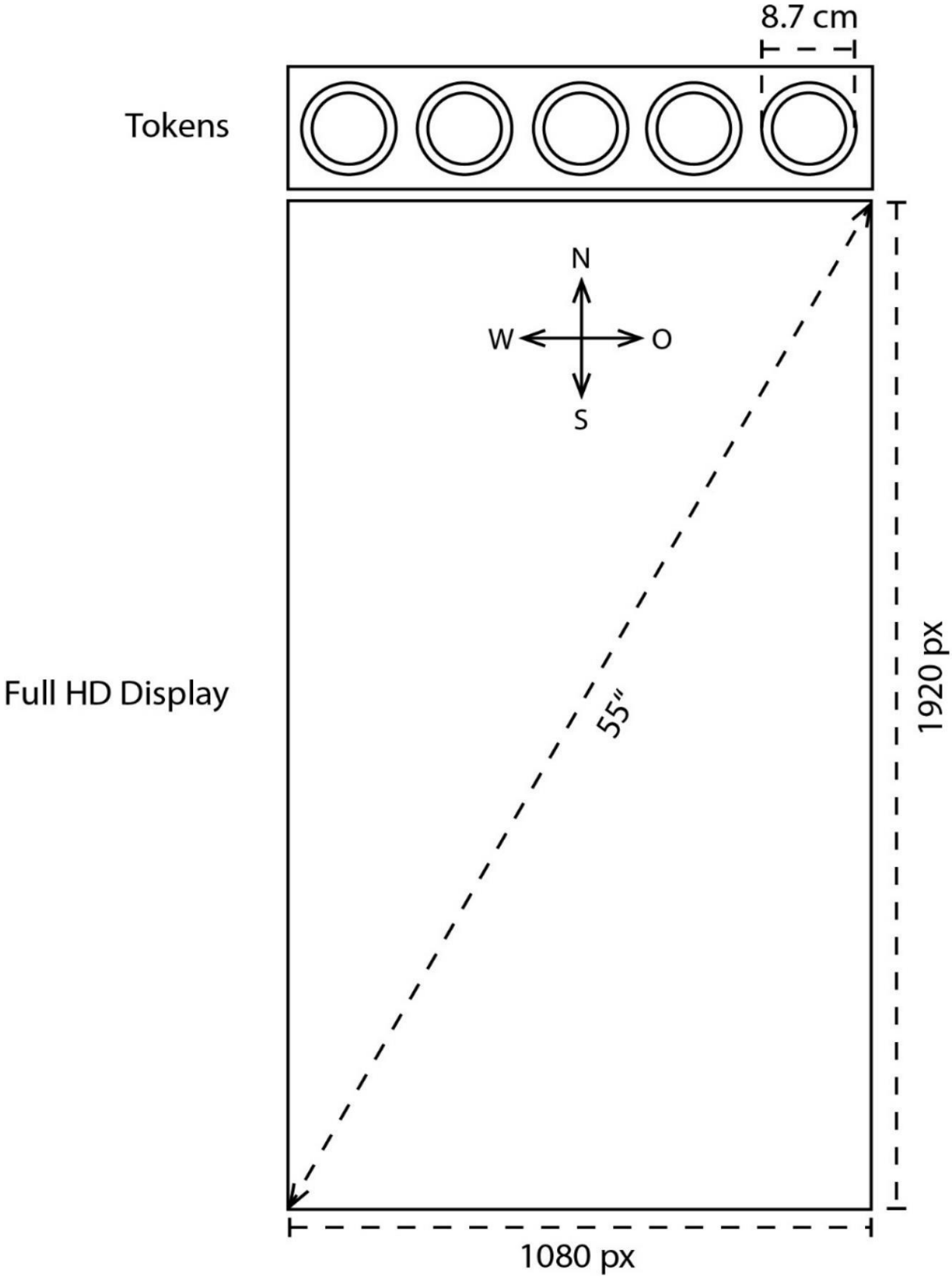
TenneT TSO GmbH. (2019). *Projekte NEP 2030*. Retrieved March 2019 from https://www.netzentwicklungsplan.de/de/projekte/projekte-nep-2030-2019

Technical Data

TenneT TSO GmbH. (2019). *Spezifikationen (PC, Screen, Table, Etc.)*. Appendix I

11. Appendix

Appendix I – Technical Information about the Touch Table



Blueprint sketch of the touch table's layout

SPEZIFIKATIONEN (PC, TOUCH, DISPLAY ETC.)

Bildschirm Daten	
Bildschirmgröße	55"/ 138cm Diagonale
Format	16:9
Auflösung	1920x1080 Full HD
Helligkeit	530 cd/m³
Toucherkennung	unendlich
Tracking Geschwindigkeit	Bis zu 200 fps
Sicherheitsglas	4mm Optiwhite Glass

Rechner Daten	
CPU	Intel Core i5 Series
RAM	8 GB
Grafik	Dedizierte Nvidia Geforce GTX 960
SSD Festplatte	Samsung Evo 240 GB
Konektivität	2x Gigabit LAN, 4x USB, WLAN, Bluetooth 4.0
OS	Windows 10 64Bit
Peripherie	Funktastatur

Data about the target device, made available by the client

Appendix II – List of created assets

3D Models:

- stadt_Haus.obj
- stadt_Hochhaus.obj
- wind_OffShore.obj
- wind_OnShore.obj
- speicher_pump.obj
- speicher_batterie.obj
- speicher_gas.obj
- powerplant.obj
- netz_masten.obj
- kv_platform.obj
- kuehlturm.obj
- akw_kuppel.obj
- industrie_lager.obj
- industrie_haus.obj
- industrie_halle.obj
- pinetree.obj

Textures:

- stadt_Haus_albedo.png
- stadt_Haus_splice.png
- stadt_Haus_big_albedo.png
- stadt_Haus_big_splice.png
- stadt_Haus_small_albed.png
- stadt_Haus_small_splice.png
- stadt_Hochhaus_albedo.png
- stadt_Hochhaus_ splice.png
- wind_OffShore_albedo.png
- wind_OffShore_ splice.png
- wind_OnShore_albedo.png
- wind_OnShore_ splice.png
- speicher_pump_albedo.png
- speicher_pump_ splice.png
- speicher_batterie_albedo.png
- speicher_batterie_ splice.png
- speicher_gas_albedo.png
- speicher_gas_kw_albedo.png
- speicher_gas_ splice.png

- powerplant_albedo.png
- powerplant_ splice.png
- netz_masten_albedo.png
- netz_masten_ splice.png
- umspannwerk_albedo.png
- umspannwerk_splice.png
- kv_platform_albedo.png
- kv_platform_ splice.png
- kuehlturm_albedo.png
- kuehlturm_ splice.png
- akw_kuppel_albedo.png
- akw_kuppel_ splice.png
- industrie_lager_albedo.png
- industrie_lager_ splice.png
- industrie_haus_albedo.png
- industrie_haus_ splice.png
- industrie_halle_albedo.png
- industrie_halle_ splice.png
- industry_factory_1_albedo.png
- industry_factory_1_splice.png
- industry_factory_2_albedo.png
- industry_factory_2_splice.png
- pinetree_albedo.png
- pinetree_ splice.png
- terrain_grass.sbs

Scripts:

- ComponentReactor.cs
- ComponentHandler.cs
- StatusHandler.cs
- VerlustReactor.cs
- VerlustHandler.cs
- SpeicherReactor.cs
- SpeicherHandler.cs
- NetzPathHandler.cs
- NetzLinkReactor.cs
- StateShift.cs
- StateShiftExtensions.cs
- StateFrame.cs
- StateCheckPoint.cs
- CheckpointReactor.cs
- VFXExtensions.cs
- PathLinkSO.cs

Appendix III – Work samples

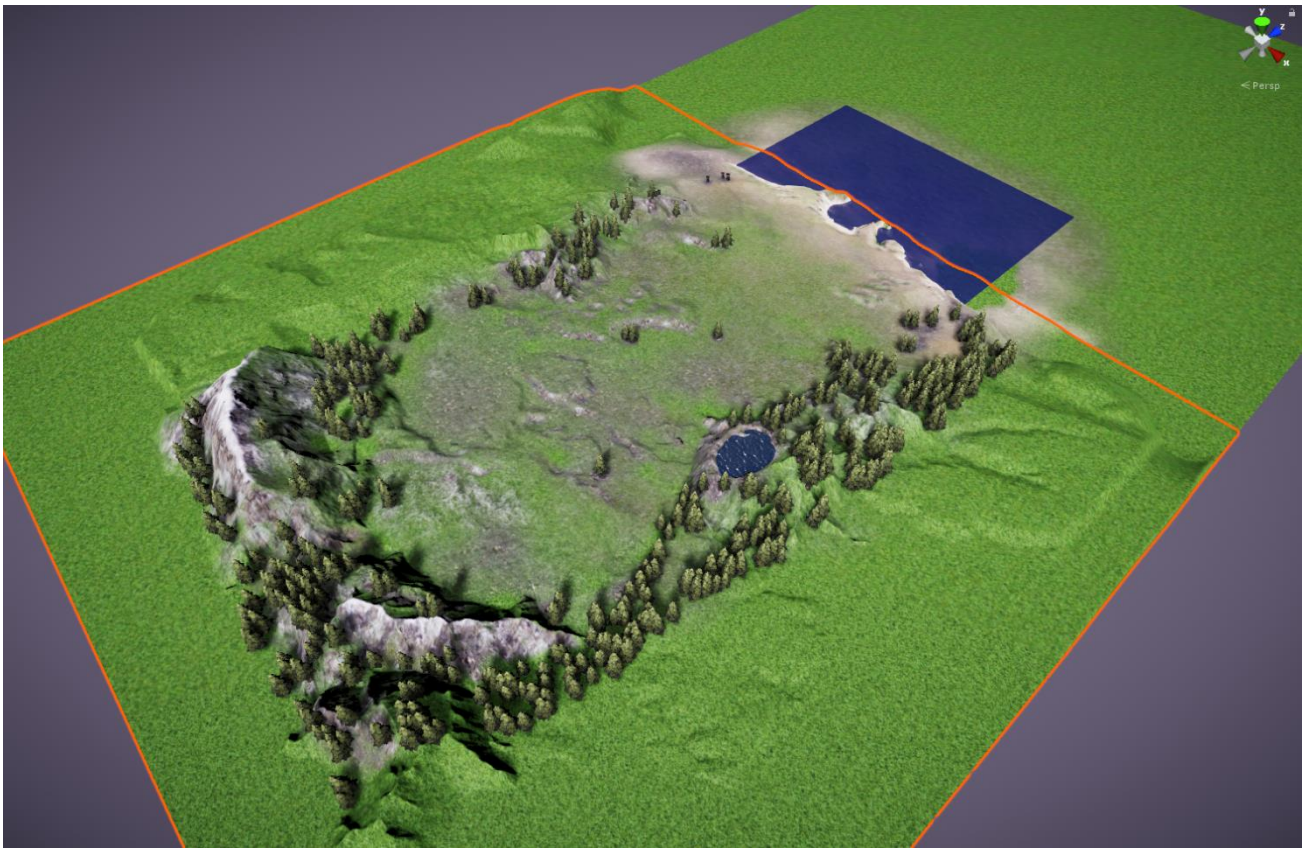
1) 3D development
2) Technical development
3) Technical Art
4) Script descriptions

In the following various samples of the work done in this graduation project are presented. They range from 3D modelling and texturing to the designing of particle effects and custom shaders using the VFX graph and Shader graph and technical tasks such as scripting necessary functionalities for the scenario to react to changes made by the user. These scripts will briefly be explained, along with samples of selected pieces of code.

1) 3D Development

- Terrain sculpting and texturing
- Component modelling

Terrain



Background environment created with a Unity terrain and sculpted and textured using Gaia

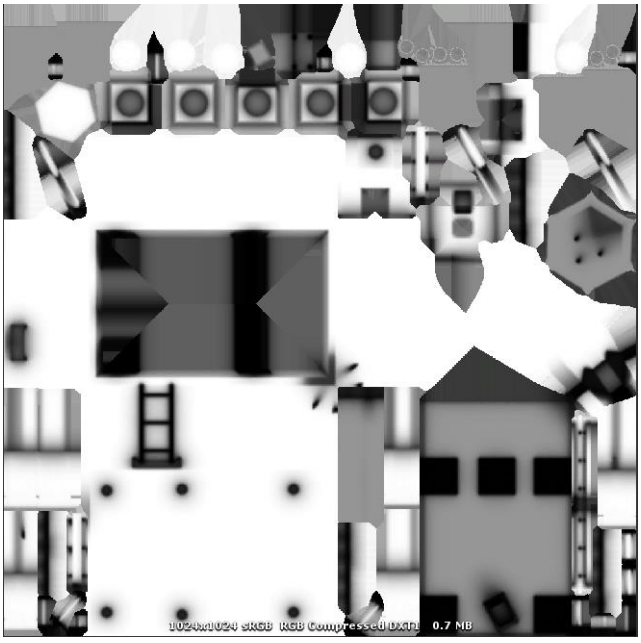
Component Modelling

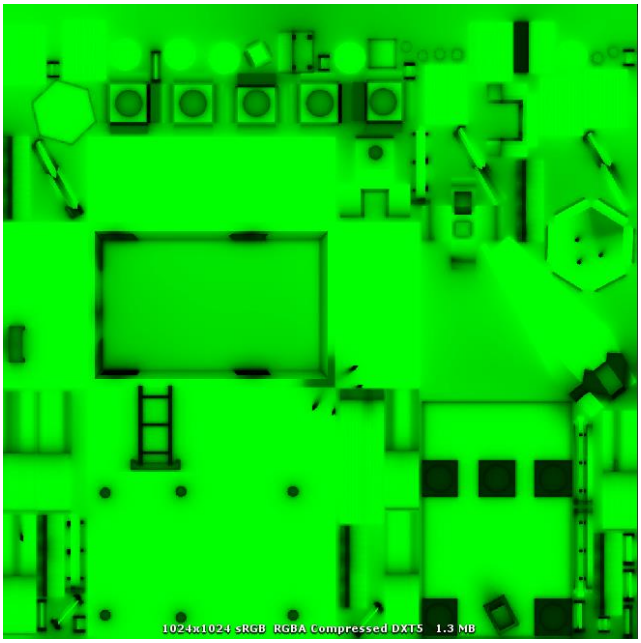
3D Models

A selection of 3D models used in the scenario

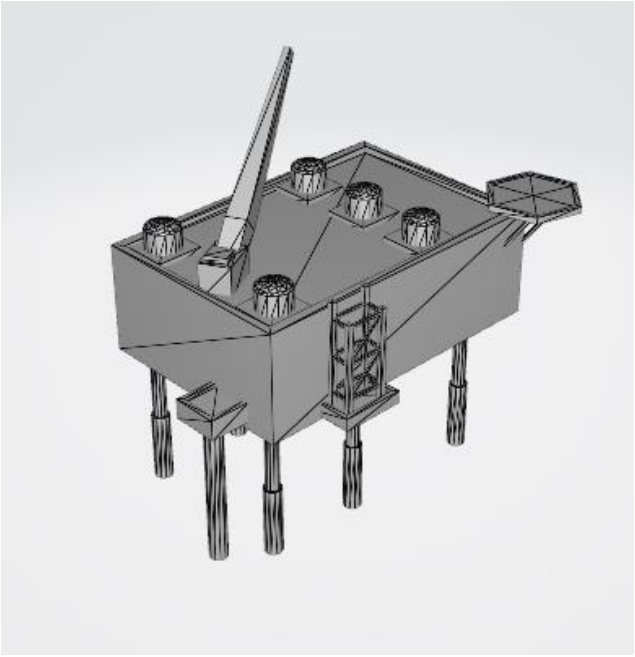
Texturing

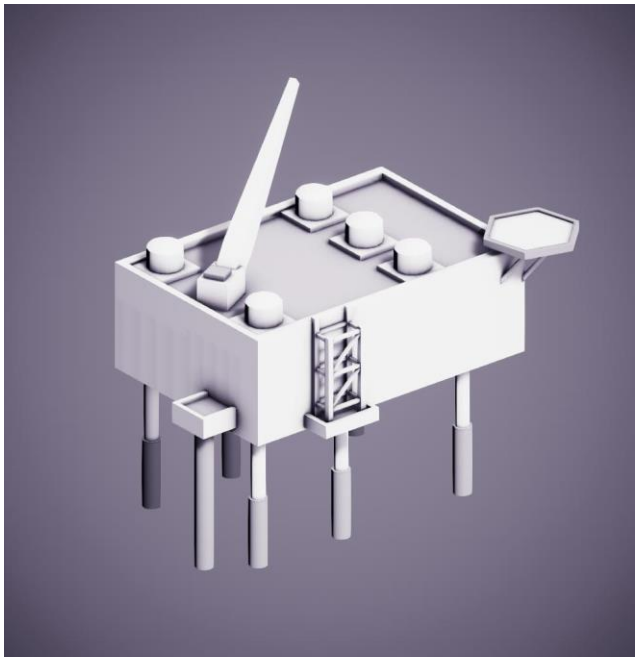
Texture maps and 3D model of kv_platform.obj





Albedo map (left) with white base and grey details mixed with ambient occlusion data; Splice map (right) consisting of Metallic (R), Ambient Occlusion (G), Blend Mask (B) and Smoothness (A), which are cast into the Red, Green, Blue and Alpha channels of a RGBA texture.





Wireframe model and final in-game rendering

2) Technical development

State Shift and Checkpoint Scripting

```
public void RegisterState(StateFrame state, object origin)
{
    OnStateRegistered(state, origin);
    // push new registered State to all connections through event
}

public void RegisterState(int power, int volatility)
{
    StateFrame newState = new StateFrame
    {
        power = (power + 12f) / 24f,
        volatility = volatility/12f,
        age = 0f
    };
    RegisterState(newState, null);
}
```

Checkpoint event triggered by registering a new scenario state

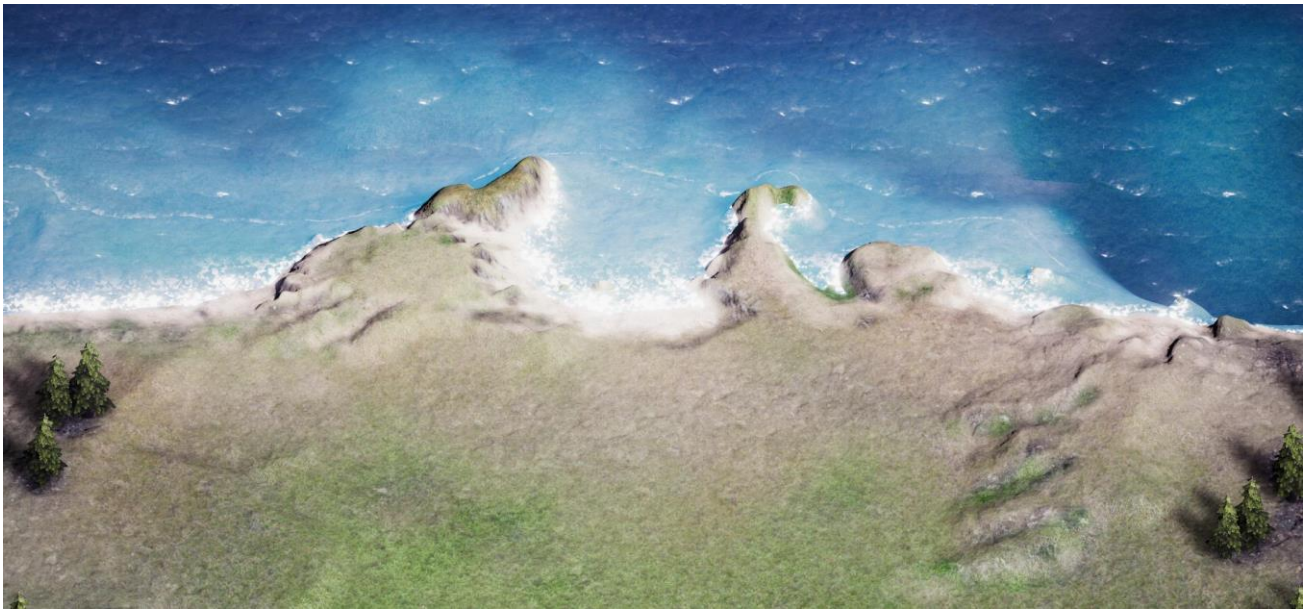
```
for (int i = 0; i < queue.Count; i++)
{
    StateFrame state = queue[i];

    int frameIndex = i * 2;
    int dragFrameIndex = frameIndex + 1;
    float drag = (state.forward ? 1 : -1) * dragDistance;

    powerKeys[frameIndex].time = state.time;
    powerKeys[frameIndex].value = state.o_power;
    powerKeys[dragFrameIndex].time = state.time - (state.forward ? 0.01f : -0.01f);
    powerKeys[dragFrameIndex].value = state.power;

    volatilityKeys[frameIndex].time = state.time;
    volatilityKeys[frameIndex].value = state.o_volatility;
    volatilityKeys[dragFrameIndex].time = state.time - (state.forward ? 0.01f : -0.01f);
    volatilityKeys[dragFrameIndex].value = state.volatility;
}
```

Extracting the states in the current queue into animation curves to be passed to the particle effect

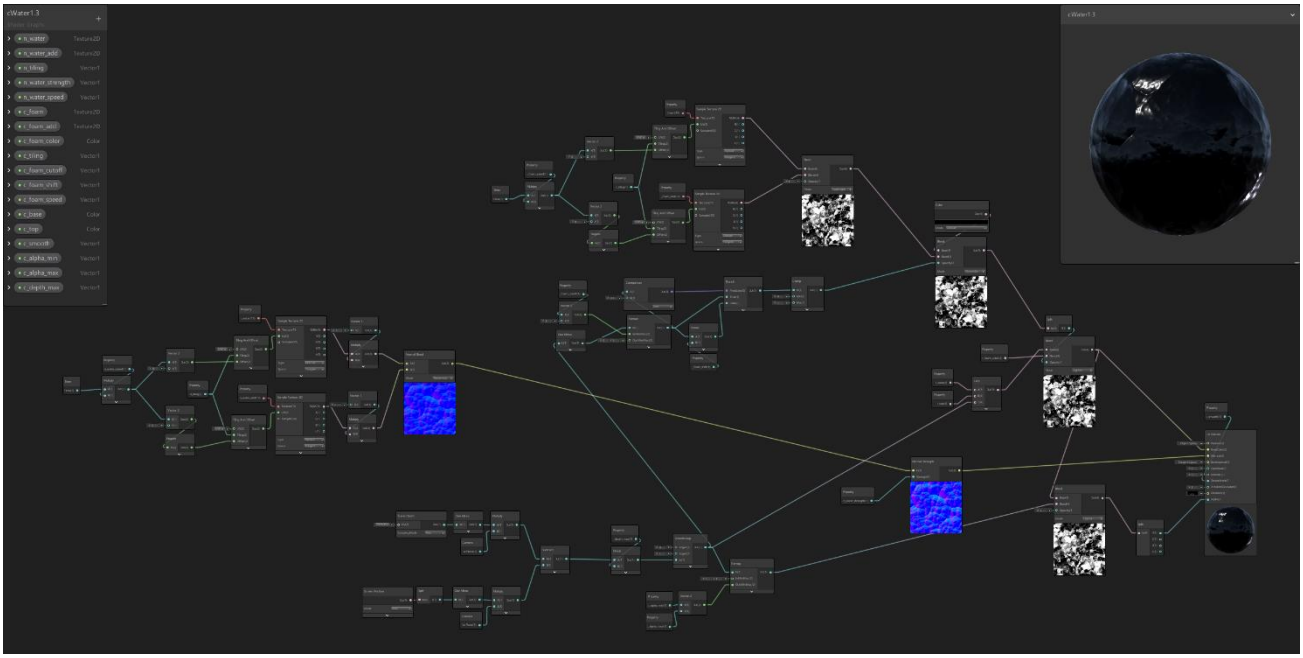
3) Technical Art

- Water shader
- Morph shader
- Particle effect
- Path Link

Water Shader

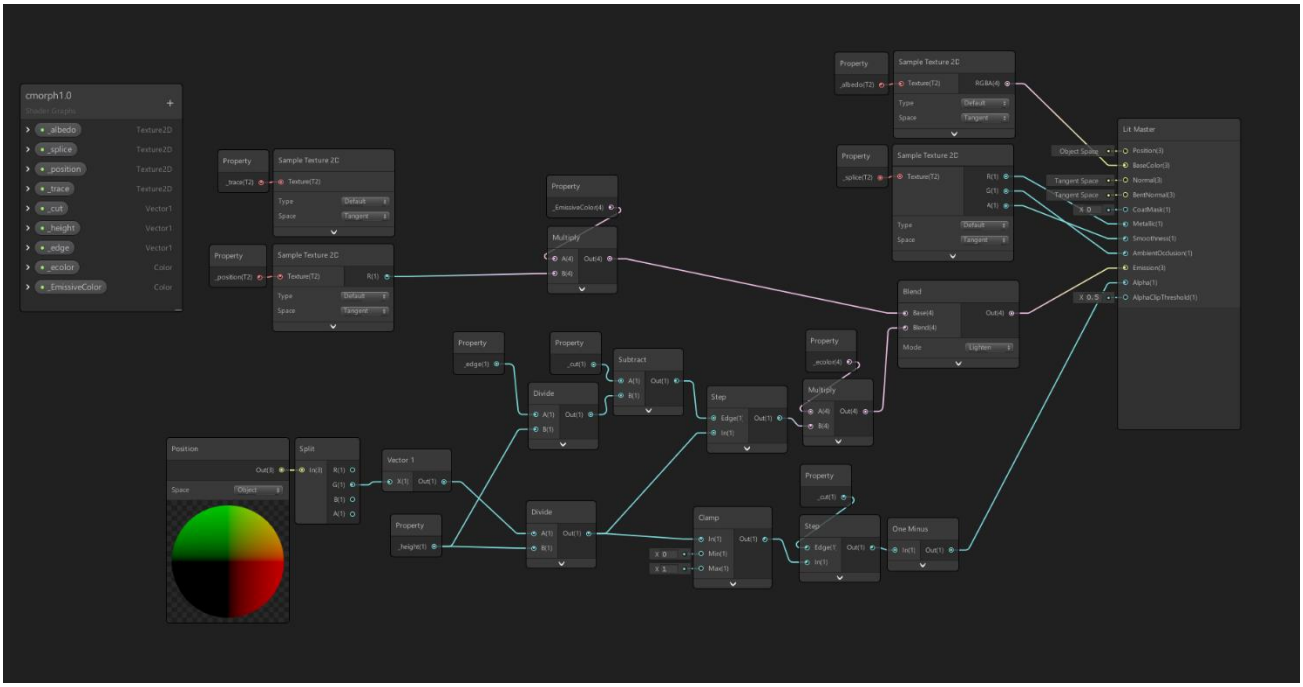


Water shader in the scenario

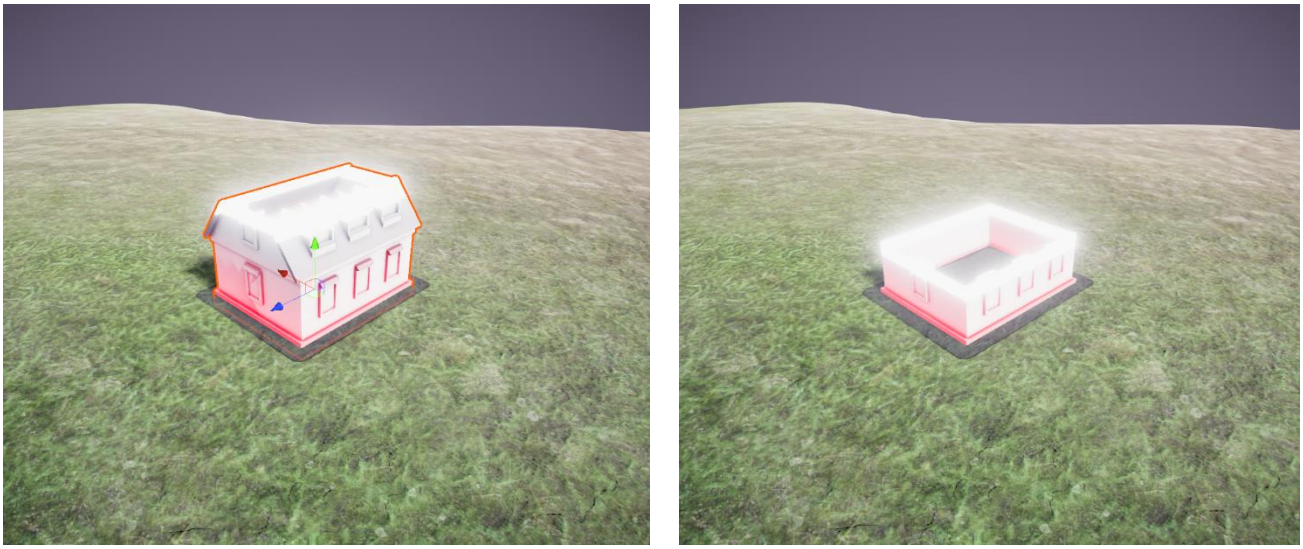


Animated texture sampling in shader graph

Morph shader

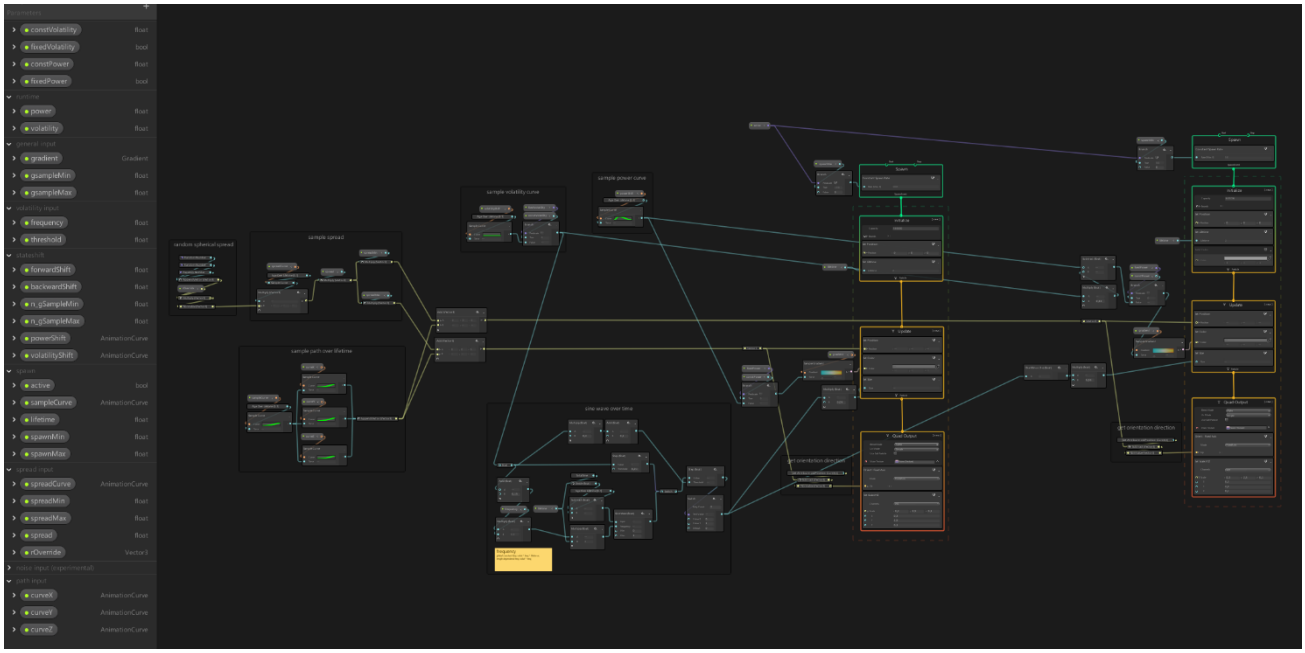


Sampling the objects height and cutting the geometry in shader graph

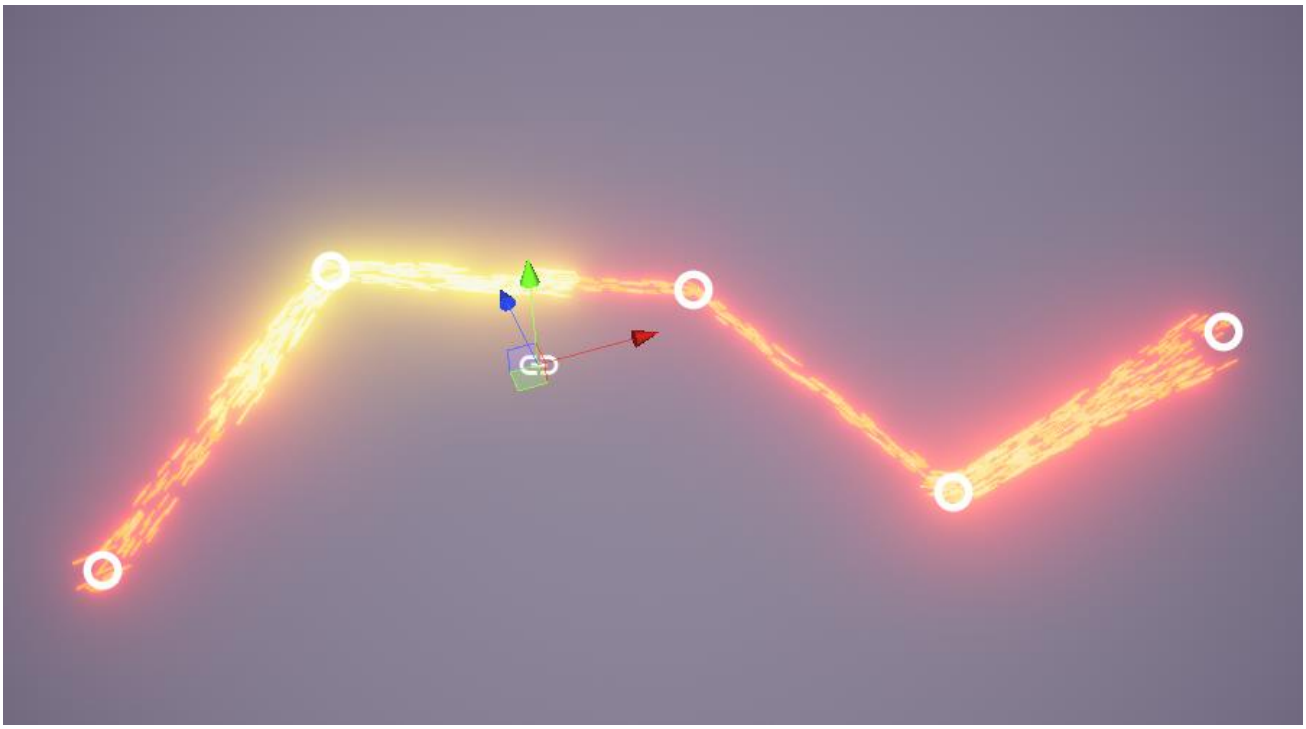


Steps of the components morph cutting in the scenario

Particle Effect



VFX graph for particle effect



Particles following the defined path of the particle effect in the editor

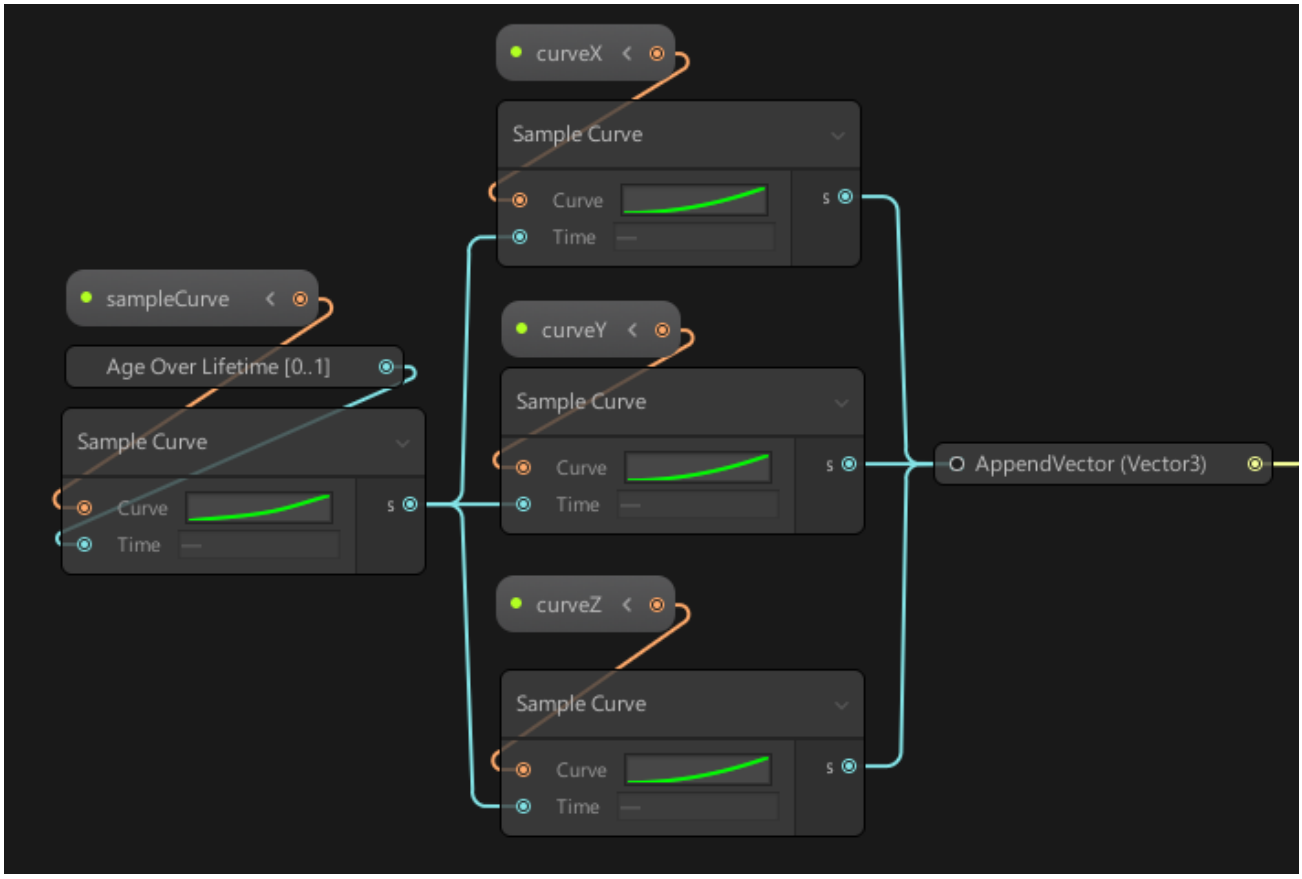
Path Link

```
foreach (pPoint point in points)
{
    if (lastPoint == null) lastPoint = point;

    float dist = Vector3.Distance(
        lastPoint.handle.transform.position,
        point.handle.transform.position);

    totalDistance += dist;
    distance[path.IndexOf(point)] = totalDistance;

    lastPoint = point;
}
```

Calculating the location of each point on the path

```
for (int i = 0; i < points.Count; i++)
{
    float time = distance[i] / totalDistance;

    xkeys[i].value = points[i].handle.transform.localPosition.x;
    xkeys[i].time = time;

    ykeys[i].value = points[i].handle.transform.localPosition.y;
    ykeys[i].time = time;

    zkeys[i].value = points[i].handle.transform.localPosition.z;
    zkeys[i].time = time;
}

curveX.SetKeys(xkeys);
curveY.SetKeys(ykeys);
curveZ.SetKeys(zkeys);

curveX.SetTangentMode(UnityEditor.AnimationUtility.TangentMode.Linear);
curveY.SetTangentMode(UnityEditor.AnimationUtility.TangentMode.Linear);
curveZ.SetTangentMode(UnityEditor.AnimationUtility.TangentMode.Linear);

SetCurves();
```

Extracting a 3-dimensional path from the points' positions using 3 animation curves



Sampling of the 3 curves in the graph in order to create a path

4) Script descriptions

- ComponentReactor.cs
- ComponentHandler.cs
- StatusHandler.cs
- VerlustReactor.cs
- VerlustHandler.cs
- SpeicherReactor.cs
- SpeicherHandler.cs
- NetzPathHandler.cs
- NetzLinkReactor.cs
- StateShift.cs
- StateShiftExtensions.cs
- StateFrame.cs
- StateCheckPoint.cs
- CheckpointReactor.cs
- VFXExtensions.cs
- PathLinkSO.cs

All reactor scripts inherit from *Reactor.cs*, created by the programmer. *Reactor.cs* registers to the events of the class it should react to. The different reactor scripts, when reacting to an event, trigger specific functions in their corresponding handler scripts.

The handler scripts contain all functionality that the component or particle system should execute, when reacting to an event, representing a change in the scenario's state. The handlers hold references to the according scripts, materials or particle systems they need to influence, for example the geometry-cutting materials on each component or the parameters of a particle system.

StateShift.cs is a script that takes in a new state, translates this state into keyframes on an animation curve and moves these frames over time along the curve's timeline. If a state reaches the end of the timeline, it is passed on to the next *StateCheckPoint.*

StateCheckPoint.cs reacts to states being registered by triggering an event containing data about the new state, which all *StateShift* instances that are registered to the checkpoint will be able to receive.

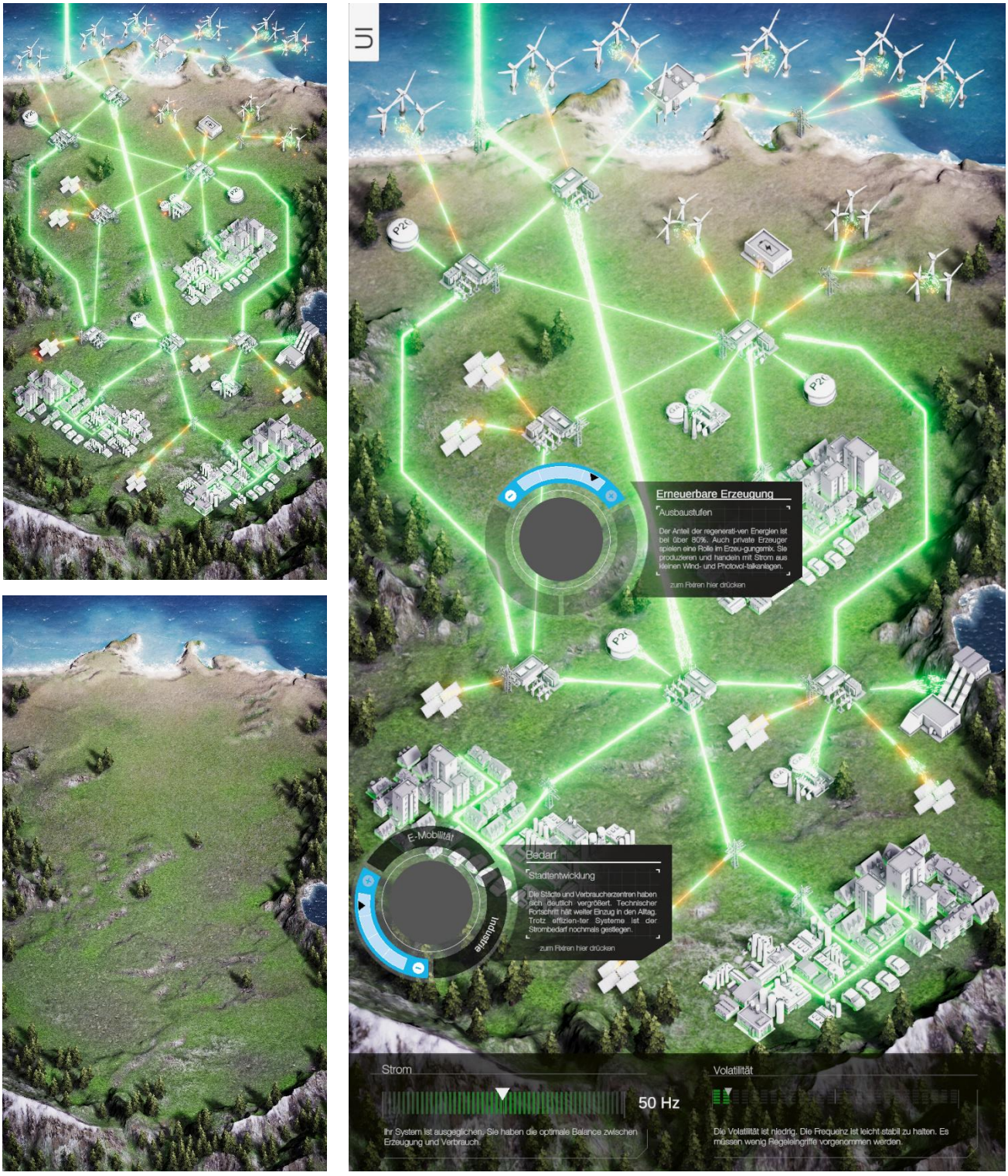
The initial triggering of a new event originates from the *CheckpointReactor* class. It inherits from *Reactor.cs* as well and triggers the first checkpoint event containing the new state, if it is specified that the checkpoint should react to a scenario change directly.

The *PathLinkSO* is a scriptable object class that contains variables a particle system can use. With it different behaviors can be defined and exchanged during development. If the behavior is updated all particle systems using the same behavior will change immediately. This optimizes the development with particles systems.

VFXExtensions and StateShiftExtensions hold helpful functionality that can easily be reused throughout the development, such as the updating of a particle system parameter e.g.

Appendix IV – Visual progress and Beta release representing the projects result

Beta Build after finalizing the last Iteration



Beta release build with full UI implementation

Visual development process

Mockup, Prototype and Alpha Iterations