Bachelor Project:

Action RPG with Procedural Story

by Tristan Weber (Matr.-Nr.: 730633)

Supervisors: Prof. Dr. Martin Leissler
             Prof. Katharina Kafka
1. Abstract

This game project aims to fill a procedurally generated world with procedurally generated stories. It is developed in conjunction with the related research project *Analysis of Story Archetypes and Structures to Parameterise Game Plots* by the same author.

The results of the research project are applied in this bachelor project. The focus lies in the creation of a social system of procedurally generated characters and their relationships. The modeled characters and their motivations build the basis for the generation of branching quest trees.

The characters inhabit a procedurally generated world. The generation of the three dimensional world map is the second focus of this project. It was designed to produce varied and visually pleasing landscapes.

In addition a simple prototype scene with a player controlled character presents the central controls of the action role playing game (RPG). It also showcases the developed pixelation shaders that are used to create retro aesthetics from 3D models.
2. Table of Contents

1. Abstract .................................................................................................................. 1

2. Table of Contents .................................................................................................. 2

3. Introduction .......................................................................................................... 4

3.1. Problem .................................................................................................................. 4

3.2. Motivation .............................................................................................................. 4

3.3. State of the Art ...................................................................................................... 5

3.4. Related Work ........................................................................................................ 6

4. Approach ................................................................................................................. 7

4.1. Identification of Tasks .......................................................................................... 7

4.2. Goal ......................................................................................................................... 7

4.3. Asset Creation ........................................................................................................ 10

5. Implementation ...................................................................................................... 12

5.1. Overview ................................................................................................................ 12

5.2. Selective Pixelated Rendering .............................................................................. 12

5.3. Per Object Pixelation via Standard (Padded Pixelate) Shader ....................... 13

5.3.1. Interface .............................................................................................................. 13

5.3.2. Shader Passes ..................................................................................................... 14

5.3.2.1. Shader GrabPass .............................................................................................. 14

5.3.2.2. Padding Pass ..................................................................................................... 14

5.3.2.3. Standard/FORWARD Pass ............................................................................. 14

5.3.2.5. Individual GrabPass ...................................................................................... 15

5.3.2.4. Pixelation Pass ............................................................................................... 15

5.4. Pixelation as Post Processing Effect via Pixelation Camera ......................... 16

5.4.1. Interface .............................................................................................................. 16

5.4.2. GrabBeforePixelation Shader ........................................................................ 17

5.4.3. PixelationPost Shader ...................................................................................... 17

5.5. Gameplay Prototype Scene .................................................................................. 18

5.6. World Map ............................................................................................................ 19

5.6.1. Designing a World Map for Procedural Narratives ..................................... 19

5.6.2. Generating the World Map ............................................................................. 20

5.6.3. World Map 3D Mesh ....................................................................................... 20

5.6.4. Procedural Landscape Shading ....................................................................... 22
3. Introduction

3.1. Problem

For every game project a number of assets have to be hand crafted by the developers. These assets are for example 3D models, textures or the game’s levels. But also the story of a game is traditionally written by an author.

Procedural content generation (PCG) is a method to shift some of the workload from the creation of hand crafted assets to the creation of systems that can automatically generate diverse assets based on a few parameters. This is often used to generate levels, but also the generation of creatures or items can be successfully realized through PCG. This project aims to use the method of PCG to generate stories.

3.2. Motivation

This bachelor project was about creating a playable action RPG with a procedurally generated world and story.

The main goal was the creation of systems that generate diverse narratives. The player should be able to continue playing the game for many hours without seeing too much repetition in the kind of narration he or she experiences.

This project realized two main parts for such a game. The first one is the procedural generation of the world map. It is a three dimensional landscape with mountains, valleys and the ocean. The world map is divided into several sub areas, which would be the game’s different levels.

The second part is the generation of a virtual social system with a variable amount of characters and the relationships between them. This social system is used to model conflicts and interests of the characters to generate their base motivations. These motivations in turn are used for generating quests from a predefined pool of quest structures. They are then further fleshed out by the generation of subquests and subactions, so that a player may never see the exact same quest twice.
3.3. State of the Art

Procedural content generation (PCG) is commonly used to generate levels. For example, the *Diablo* franchise (Blizzard Entertainment 1996-2012) uses a predefined pool of small level pieces and sticks them together at runtime to generate levels with a unique layout. The popular game *Minecraft* (Mojang AB 2009) goes even further and generates its whole world from 1m x 1m blocks.

Another common use for PCG is the generation of items. Items with randomized stats can be seen in the Diablo franchise but also in other games like the *Borderlands* franchise (Gearbox Software LLC 2009-2012).

Games that use PCG to generate some kind of narrative often do so by inserting randomly selected targets into predefined quest structures. For example, in *Xenoblade Chronicles X* (Monolith Soft 2015) in addition to the detailed main story there is a quest board where the game world's inhabitants post their quests in a “kill X amount of Y” or “gather X amount of Y” fashion. These seem repetitive after a short while, even if X and Y are always replaced with different targets.

![Quest board in Xenoblade Chronicles X](image)
Some strategy games, like Civilization V (Firaxis Games 2010), provide emergent narratives. There nations can interact with each other through diplomacy. If the player aggressively attacks other nations, the other nations will react by warning the player of potential consequences for his or her actions. The overall story arc, which nations are loyal allies or which ones should pay for the war crimes of the past, are mainly in the player’s head. The narrative emerges as the player writes history in the randomly generated world.

XCOM: Enemy Unknown (Firaxis Games 2012) makes the player even bond to the procedurally generated characters. The player controls a team of randomly generated soldiers. Because the soldiers’ death is permanent, it can have a huge emotional impact on the player after having him or her deployed in countless missions, even though the soldiers are actually void of much personality at all.

Dwarf Fortress (Tarn Adams 2006) is one of the most powerful procedural story generating games. A whole world is simulated with a rich history. While it is extremely powerful at generating stories that resemble a believable historic background for the gameplay, the worlds generated are generally too complicated to be fully comprehended by the player.

Using PCG for more complex narratives in RPGs seems to be not done outside of experimental projects.

3.4. Related Work

Ed Key and Nicolai Troshinsky of Twisted Tree Games are currently developing Forest of Sleep, a game that aims to procedurally generate stories that are like old russian folk tales (Phill Cameron 2015). Their focus is mainly on generating stories in the visual and narrative style found in historic eastern european tales. The way they go about this is by developing a story engine that connects individual parts of crafted templates and connecting them based on a predefined ruleset.

The author knows of no other projects currently under development that focus on the procedural generation of stories.
4. Approach

4.1. Identification of Tasks

The main work for this procedural action RPG was divided in two parts. The first one is the creation of all assets. This includes all 2D and 3D assets, but also the gameplay scripts.

The second one is the development of the system to create an abstract representation of the story.

These two parts work relatively independent of each other. The assets and gameplay would work with a completely handcrafted story and the story generator could be used for a completely different setting with different assets.

4.2. Goal

From the beginning the goal of the project was to create an action RPG. The gameplay therefore would incorporate combat and equipment management. As inspiration for the gameplay comes on the one hand Dark Souls. The combat should be very unforgiving but yield high satisfaction when successfully executed. On the other hand the gameplay is inspired by Hack and Slay games like Diablo (Blizzard Entertainment 1996-2012). The perspective would be from a birds eye view with the player character centered in the screen.

One of the big weaknesses of a procedurally generated story is that they are relatively formulaic by their nature. On the other hand they can offer emerging narratives, the “story that emerges in the player’s head over the course of the game” as described by Leigh Alexander on Gamasutra (2012) talking about XCOM: Enemy Unknown. He argues that the blank slate nature of the procedurally generated characters allow players to develop the emotional connection to them in their own pace by living through the adventures the game creates.

Losing one of the characters that grew over the course of many hours to death can have just as much of an emotional impact on the player as the iconic Aeris death scene from Final Fantasy VII (Square 1997).

To achieve the possibility of losing the player character for good which is a much more
significant event than just losing a life token the game should feature permanent death, as many roguelike games do. The duration of a single playthrough in most roguelike games is very short, usually ranging from a couple of minutes to an hour at most. This is of course much too short for the player to build an emotional connection to any character, so playthroughs of the game developed in this project should last longer than that. This means the nature of the narratives in this game also has to be varied instead of always featuring an archetype like the classic hero’s journey over and over.

In his book *Video Game Storytelling: What Every Developer Needs to Know about Narrative Techniques* (2014), Evan Skolnick argues that the foundation of interesting stories is a conflict (Chapter 1 - Conflict: The Fuel of Story). In that spirit the goal of this project was to model social dynamics between procedurally generated characters and have their relationships as drivers for the creation of stories. The motivations of the characters based on their relationships could then be used to create procedural quests using the system developed by Jonathon Doran and Ian Parberry in their paper *A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs* (2011). They model quests from very basic atomic actions based on a virtual character’s motivation. The motivations they categorize every basic quest structure into are knowledge, comfort, reputation, serenity, protection, conquest, wealth, ability and equipment.

The main drawback of their system is the lack of a comprehensible choice of motivation, they just pick one randomly.

The social system modeled in this project would therefore be perfect to generate characters with motivations that make sense in the world they inhabit and to the player fulfilling their requests. Only a character with friends would be motivated to protect, only an intelligent character would be motivated by knowledge. This social system does not aim to be an accurate depiction of real world social dynamics, it uses several simplifications and shortcuts. The main goal here is to provide just enough social friction between the characters to generate varied motivations for them while also providing some parameters to flesh them out further. For example there could be an intelligence attribute that determines if a character is motivated by knowledge while possibly also defining his role in the world as a scholar, teacher or wizard instead of a
dumb and aggressive orc (who consequently might be motivated by conquest).
The combination of procedurally generated characters with needs and motivations
and the quest generator of Jonathon Doran and Ian Parberry gives a sheer endless
number of possible narrative scenarios.

As generating a great number of characters with all the relationships between them
can be both time consuming and very slow to process for computing their motiva-
tions, the total amount of characters per generated world is limited. While even with
a limited amount of characters endless quests could be generated, the tradeoff coming
with this decision is that the player would eventually get to know every single cha-
racter in the world and he would stop getting to know new entities within a specific
generated world, very likely resulting in a loss of interest in the game.
The generated world poses another challenge. Either it is limitless, but then special
care has to be taken to place the generated characters close enough to each other, so
that the player can reach them and solve quests involving more than one character. Or
the world is limited, but then just like with a limited number of characters the player
would eventually have visited every place.
One possible solution to the problem of eventually knowing every entity in the world
would be to periodically remove some characters from the world (for example by de-
ath or emigration) and add new ones (by birth or immigration).
A different approach could be to have the player character be the one to leave a speci-
fic generated world when the number of new people to get to know and new areas to
explore gets exhausted. If the player character is a traveler of some kind, for example a
pirate or an interdimensional traveler, the switch to a new world with new characters
is easily explained.
This would allow players to build an emotional connection with their characters over
the course of the adventures in each world. Every player would have his own stories
to tell of the things they experienced, as the worlds and events they play through are
procedurally generated. And when the player has seen everything that is there to see
in one of the worlds, he or she can visit the next one.
4.3. Asset Creation

For the visual style it was decided to go for a mix of low resolution characters with high resolution effects and environment. One of the main inspirations for this was *Dungeon of the Endless* (AMPLITUDE Studios 2014), where pixelated characters explore pixelated dungeons. In that game both are illuminated in full HD and special effects like mist or particle effects are also in a modern high resolution aesthetic.

One of the main considerations for doing low resolution characters was the time constraints of a project that is under development by a single developer. Low resolution assets can be created much faster than modern 3D characters with high polygon count and HD textures.

Doing low resolution characters therefore can save a lot of time while still having a certain charm, as evidenced by the high number of modern games with retro graphics.

Even when deciding for a low resolution look of the characters there is no limitation on how to go about this.

One route very often seen in independently developed games is the use of pixelated sprite art resembling the look of Nintendo Entertainment System (1983) or Super Nintendo Entertainment System (1990) games. While this can certainly have a high
appeal, it forces the asset creator to create many individual sprites for each frame of animation.

An alternative would be to create low polygon characters, more in the style of Nintendo 64 games. This has the advantage of being easy to create compared to a high polygon character, while still being animated with a rigged 3D skeleton, allowing fluid motions without spending time on every pixel of every frame of animation. 3D models also have the advantage of being easy to visually embed in the game’s scenes, since they can be lit from individual light sources and cast shadows on themselves and the environment.

For this project a mix of both approaches was chosen. The characters would be created as low polygon 3D models, but rendered very pixelated in the engine. This combines the easy creation and animation of the characters using a 3D rig, while giving the visual appeal of pixelated graphics. Pixelating the 3D models also reduces the time that has to be spent on the textures as their resolution depends more on the pixelation factor than the resolution of the screen the game is played on.

The environment and other effects on the other hand were decided to be in full HD with a mix of 2D and 3D. With the decision to have a birds eye view on the player character it was also clear that the perspective on environment assets would not change much, allowing to create them in 2D or 3D on a case-by-case basis.

The goal of mixing pixelated graphics with high resolution graphics and 2D and 3D assets was to create an appealing and modern look for the game while still giving a nod to retro aesthetics and being able to develop the game in a reasonable amount of time.
5. Implementation

5.1. Overview

All assets in this project were created from scratch unless otherwise noted. The game is developed with the Unity 3D (Unity Technologies) engine. All scripts are written in C#. All shaders are in Unity's own ShaderLab (Unity Technologies 2015) language, though some shaders were created using the Shader Forge (Joachim Holmer 2015) plugin for Unity.

These tools were chosen because they are both modern and the author has worked with them before. The main focus of this work was on the creation of systems that procedurally generate the game world and story. To give an example of the targeted visuals a simple dummy scene with an example character was created. This scene mainly showcases the pixelation shader developed for this game.

5.2. Selective Pixelated Rendering

To combine the ease of animating 3D characters compared to frame by frame animation of 2D characters and the aesthetics of pixelated retro graphics, a shader based approach was used. As one of the goals was to have the characters pixelated but the environment and special effects in full resolution, rendering the whole screen at a lower resolution was not a viable solution.

For this project two methods of pixelating 3D assets with shaders were implemented. The first one is a material shader that allows per object pixelation. This allows every material with the shader to have a different pixelation factor. The drawback is that the image processing for pixelating has to be executed per object, which can be slow with a greater number of materials in a scene with that shader.

The second one is a post processing shader that is used on a camera. This pixelates the whole image that camera captures. To prevent everything from being pixelated, a dedicated pixelation camera has to be placed in the scene and the images of the normal resolution camera have to be combined with the pixelation camera. Rendering layers can be used to select which camera captures which objects. For this post processing shader only a single draw call is made instead of one for each object.

Both pixelation methods can be used in combination.
5.3. Per Object Pixelation via Standard (Padded Pixelate) Shader

With the fifth version of Unity a new physically based standard material shader was introduced. The per object pixelation material shader developed in this project is based on that standard material and called Standard (Padded Pixelate), because of the padding pass in the shader.

Since it is based on the standard shader, for example lighting and normal information are computed exactly the same and the pixelation affects the rendered object in a second step.

The shader consists of five passes. The fallback shader is Unity’s built in standard shader.

A custom editor PixelationShaderGUI based on the Standard shaders editor has been added to easily modify the pixelation settings in the Unity editor.

5.3.1. Interface

The interface is based on the Standard shader as well. At the bottom the settings for pixelation have been added. Cell size determines the actual pixel size as a fraction of the screen width. Padding size determines the extrusion for the Padding pass. This is useful so edge pixels don’t get cut off. If the padding size is too big, it will be pixelated, too, though. The padding size should be adjusted according to the distance of the object to the camera so it is not visible itself but all actual edge pixels of the object are visible.
5.3.2. Shader Passes

5.3.2.1. Shader GrabPass

The first pass is a GrabPass. This one is shared between all objects with this shader (see also Unity's documentation of GrabPass). It saves a screenshot before the actual rendering of the objects. This is necessary for the 5th pass.

5.3.2.2. Padding Pass

In the padding pass the object is rendered unlit with all vertices extruded along their normals. The padding is used to prevent edge pixels from getting cut off in the last pass.

5.3.2.3. Standard/FORWARD Pass

The FORWARD pass of the standard shader is then applied over the padding, leaving only the extruded edge of the previous pass visible.
5.3.2.5. Individual GrabPass
For each object with the pixelation shader one additional grab pass has to be done. This grabs the Standard shaded object with the padding into a buffer. So the total amount of grabs per frame $g$ with the number of objects with the Standard (Padded Pixelate) shader $n$ from this method is:
\[ g = n + 1 \]

5.3.2.4. Pixelation Pass

The final pass is where the actual pixelation happens. The effect is based on the Making a local pixelation image effect shader forum post.
To get the pixelation the UV coordinates are rounded according to the cell size variable.
To prevent the edges of the object from being cut off, leaving „half pixels“, only fully visible pixels are shaded. To determine which pixels are fully visible, for each pixel all four corner coordinate's color values are compared to the same coordinate's color values in the first grab pass. If one of the corner color values is the same as in the first grab pass it must be at the edge of the object, since there is the same color visible as before the rendering of the object (the pixel would be cut off, as it is only rendered where actual vertices are).
This is also why the padding was added, because even slightly cut off pixels would not be drawn anymore, even though they might be 90% visible. With the padding the corners of a pixel will still have a different color value than before rendering the object.
5.4. Pixelation as Post Processing Effect via Pixelation Camera

The pixelation camera is a second scene camera only rendering objects in the Pixelation layer that have the GrabBeforePixelation shader applied. It produces clean aliased objects with alpha values in edge pixels. All objects rendered with this camera share a common pixel size, unlike objects with the Standard (Padded Pixelate) shader. On the other hand it only uses a single screen grab and provides correct edges and z-sorting even for objects very close to each other.

5.4.1. Interface

The newly added Pixel Camera only renders objects on the Pixelation layer. It has a greater Depth value than the main camera, so it renders after the main camera. To not overwrite all screen pixels, the clear flag has to be set to Don't Clear. This also preserves the depth values from the main camera, giving correct z-ordering. All other values are copied from the main camera via GetMainCamValues component.
5.4.2. GrabBeforePixelation Shader

All objects that should be rendered by the Pixelation Camera need the GrabBeforePixelation shader. In addition to Unity’s standard shader computation this does one screen grab (shared by all objects with this shader, so only one grab per frame) before rendering the objects with the standard shader.

5.4.3. PixelationPost Shader

The PixelationPost shader is a full-screen post effect used by the pixelation camera.

It compares a pixelized version of the screen grab before rendering the objects on the Pixelation layer with a pixelized version of the screen with the pixelization objects and only draws the pixelized version where there is a difference between the two (the difference being the object drawn in the Pixelation layer). All other pixels are just redrawn from the grab from the GrabBeforePixelation shader.
5.5. Gameplay Prototype Scene

To showcase how gameplay would look and give an impression of the controls a simple dummy scene with a test character was created. Here the player controls the dummy hero on an endless grid area.

The controls are:

- **Controller X Button** or **Left Mouse Button** Normal Attack
- **Controller Y Button** or **Right Mouse button** Heavy Attack
- **Controller Left Joystick** or **Keyboard WASD** Movement
- **Controller Any Shoulder Button** or **Keyboard Q** Defend

The dummy scene
5.6. World Map

5.6.1. Designing a World Map for Procedural Narratives

The game world was designed to be highly integrated with the story generation. As many action RPGs do (for example Diablo), the world in this game is subdivided into several large areas between which the player would transition.

Much like the story, the game world was designed to be procedurally generated as well. A hand crafted pre designed world would eventually be completely explored, which would reduce the player’s interest in continuing to play the game. The generated narratives were intended to route the player through the world in a comprehensible but not necessarily predictable way. It should make sense to go from one place to another not only because the player is told so within the story, but also from a gameplay perspective.

When new areas are visited, they need to pose an appropriate gameplay challenge. This does not have to mean that all enemies are automatically increasing in strength as the player character gets stronger, commonly referenced to as autoleveling, though this technique is successfully implemented in games like The Elder Scrolls V: Skyrim (Bethesda Game Studios 2011). An alternative can be to have an “intended order” for traveling through the areas. There are multiple ways to implement this.

The easiest solution would be to have a linear order of areas. Transitioning from one area to the next would only occur when the player is done there. The difficulty for an area could be set up when a new area is entered by evaluating the current player character’s strength. Ordering the worlds areas in this way would be highly predictable by the player, though.

The other extreme would be to have a completely open world structure. There the player could travel in any direction he liked, experiencing absolutely unpredictable adventures. The tradeoffs coming with that solution from a developer perspective are the extreme difficulty of telling stories that follow a clear narrative thread and that it would be very hard to have any kind of appropriate difficulty as the player progresses without autoleveling enemies. If the player is allowed to travel in basically any direction, it is impossible to ensure that they experience the generated story’s events in the correct order without magically spawning them into the players path which again
would be very predictable and make exploring very boring, as the direction of travel
would not really matter.
The solution chosen for this game is one between those two extremes.

5.6.2. Generating the World Map
The foundation for the procedural world generation in this game was the ManyLandс
s Generator by Nuclear Horse Studios (2014) released under MIT license. It was modi-

fied and substantially extended for the specific needs of this project under the name
WorldGenerator.

The two algorithms used as basis for the generation of a 3D mesh for the world map
are Perlin noise implemented in LibNoise (Ricardo J. Méndez et al. 2014) to generate a
height map, though this implementation does not use Unity’s built in landscape tools,
and the generation of a Voronoi graph using FortuneVoronoi (Ben D. 2013) to di-

vide space into non uniform tiles. A Voronoi graph was chosen over a 3D plane that is di-

vided into quads because it produces much more natural results.

The generated mesh is then procedurally textured with a procedural custom shader.
Further effects are added to give a visually pleasing world map that is also usable for
the specific gameplay demands of this project.

5.6.3. World Map 3D Mesh
The 3D mesh for the world map is generated in several distinct steps. First of all a
number of random two dimensional points inside the unit circle is sampled. The count
of points can be parametrically controlled. The points then are relaxed to slightly re-
duce their randomness via Lloyd’s algorithm (Wikipedia: Lloyd’s algorithm 2016),
to reduce clustering of points in single locations. The number of relaxation steps can
also be controlled via parameters. From these points a Voronoi graph is created. The
number of points and number of relaxation steps highly influence how long the wor-
ld generation takes. Several steps of the world generation require iteration over all
points. It is consequently especially important to choose these numbers low enough
to ensure fast execution of the WorldGenerator and high enough to produce visually
pleasing results. For each point coordinates Perlin noise is sampled to get the height of
the world map at that point. This results in a mountain like landscape. To give a more
visually interesting result, multiple Perlin noise variants are combined. As the game world should not only feature valleys and mountains but also coasts, all points below a certain threshold are treated as water. The points above that water height thus form islands. The Perlin noise variants are chosen to result in mostly big connected islands, since land areas should be the norm and coast areas the exception. It can still happen that in addition to one large island several smaller islands are generated as well. To remove these unreachable islands, all but the largest islands are culled and forced to be water tiles as well. To generate the 3D mesh from these points, each point is connected to it’s corresponding Voronoi graph corner points, resulting in a triangulated mesh. The outmost circle of points is filtered from this step, as some of their corner points have infinite coordinate values.
5.6.4. Procedural Landscape Shading

To shade the procedurally generated 3D mesh of the world maps landscapes a custom procedural material was created. The base resource for creating the landscape material was again Perlin noise. Again multiple Perlin noises are combined to create the final material.

To provide controls for fine tuning the noise textures, each noise layer is created in a custom shader and rendered to a render texture (see also Unity’s *Render Texture* documentation). The different noise shaders are all based on *Turbulence Library* (Jérémie St-Amand 2014) for Unity. They are rendered to render textures with a custom script. The actual landscape material shader was created with *ShaderForge* for Unity. As Perlin noise is used to determine the height (y coordinate) for each vertex of the landscape mesh, the possible minimum and maximum height values are known. This knowledge was essential for the creation of the landscape shader.

The material produces several bands of submaterials according to the world position of the rendered pixel. At low height the landscape is shaded like sand as to give the impression of beaches near the edge to the water. The next higher band is a grass like layer. The topmost layer is rock like, so the highest points of the map look like mountains. To prevent the bands from being too visible and regular they are blended into each other with noisy edges and are all visually unified by overlaying them with a noise based dirt pattern. Also if the normals are at very steep angle the rock layer is used regardless of the current height, so steep cliffs are always rocky and never sandy or grassy.

One of the main visual problems were the very straight edges near the water, since the coast edge was also just based on the Voronoi graph. The naive solution to this would be to increase the resolution of the Voronoi graph by adding many more randomly generated points to it. This was not sensible as it would also increase the time to generate the mesh by orders of magnitude. The better solution here was to animate the alpha values of pixels near the water by multiplying two moving noise textures, much like with the cloud layer above the landscape (see section „5.6.5. Procedural Cloud Shading“ on page 23). This gives the impression of waves petering out on the sandy shores and successfully minimises the impression of very straight coast edges.
5.6.5. Procedural Cloud Shading

With just a landscape mesh the world looked very static. To give it a more believable and real world like aesthetic a cloud layer was added above the landscape. This consists of two simple plane meshes that are high enough above the landscape as to never intersect with it.

The cloud shader is based on Turbulence Library noise shaders, too. The first mesh is rendered with the *DoubleLayerCloudNoise* shader. In there two Perlin noise layers are animated over time and multiplied with each other. The animation consists of simply transforming the sampled coordinates of the Perlin noises. The two noise layers are moving in slightly different directions. By multiplying both layers the resulting texture gives a convincing cloud look. Since the two layers are constantly moving the result gives the impression of clouds slowly coming into existence and fading away constantly, while overall moving in a certain direction. The overall direction and movement speed of the clouds is randomized when a new world is generated, so it looks like the wind speed and direction are not always the same.
The second plane mesh is used for casting shadows onto the meshes below the clouds. The shadow casting of the first mesh is completely turned off. For the shadow plane only the cast shadows are rendered. This is because the clouds themselves look better when rendered with different alpha values, so some clouds are very opaque while others are very transparent. Unity’s lighting system does not allow transparency to cast shadows. For the lights a material is either fully opaque or fully transparent at a specific location. The shadow material on the second plane thus uses an alpha clip value. Below a predefined alpha value all pixels are treated as completely transparent, all values above the threshold are treated as opaque. The cloud material’s Perlin noise and transformation coordinates and the cloud shadow casting material’s noise and transformation are synchronised via a simple script, so they always have the same underlying noise values.
5.6.6. Shading the World’s Water

The water shader for the world map’s oceans is based on Sjoerd “Hourences” De Jong’s outdoor water tutorial (2007). This Unreal Engine 3 tutorial was recreated using ShaderForge, though some adjustments had to be made because of the switch from Unreal Engine 3 to Unity.

The resulting material uses several color, greyscale and normal textures to give the look of moving waves that are lit by the sun and environment. It is used on a simple plane mesh that is positioned to match the landscape's assumed water level.

Water added to the world

5.6.7. Finetuning of World Map Environment Look

To visually tie the generated landscape, the cloud layer and the water together, some post processing effects were added. From the built in Unity post processing effects Color Correction Curves and a subtle Vignette were utilized.

Additionally a custom atmospheric scattering post processing effect was created. It uses a simple scene depth sampling to fade pixels that are far away from the camera to a predefined atmosphere color. The fading distance values are manually finetuned to give a good result with the defined camera perspective. This makes the landscape look
like viewed from an airplane, as the farthest mountains and water are disappearing into the atmospheric scattering. It should be noted that the achieved look was artistically motivated and is not a physically correct representation of Rayleigh scattering.

5.6.8. Clustering of Land Tiles for Division of the World intoPlayable Areas

Apart from the visual representation of the world, the generated map had to work well for the targeted gameplay requirements. As discussed, the goal was to divide the world map into several distinct areas that could be traveled to from one to the other by the player.

To produce areas with irregular and unpredictable borders again a Voronoi graph was utilized. For this second Voronoi graph a much smaller number of random points inside the unit circle is generated and relaxed, much like the creation of the world tiles. For each generated area Voronoi tile the land points generated in the world map mesh creation step are selected.

All area voronoi tiles containing only water points are ignored, as gameplay wise water does not play any role.

If an area Voronoi tile contains less than 25% of the median number of contained
points of all tiles, the area is ignored, too, so areas consisting only of a handful of land tiles are ignored and all areas are sufficiently large.

Second Voronoi diagram to cluster the tiles from the first step

Tiles are clustered based on the big Voronoi tile they are in. One of the big Voronoi tiles is too small to be considered for the world areas. Note the gaps in the green cluster on the right, those tiles are water tiles.

For each generated area a mesh is created in much the same way as the base landscape mesh. In addition to the Voronoi tiles, the outer edge of the area is used to create a border mesh. The outer border edges are moved slightly inward for each area, so they can never intersect.

For each area the neighbouring areas are determined. For areas that do not border to any other area (because for example the Voronoi landscape tiles are all water tiles between them), the closest other area is defined as a neighbor. If multiple groups of areas are created this way, they are connected to each other as well, so all areas can reach
every other area by finding a path over their connected neighbors. This ensures that the playable world is fully connected and reachable by the player.

A random area is chosen as the first area. A search tree is created with this selected first area as the root. Recursively all neighbouring areas are added to the tree. The branches of the resulting tree are then sorted by their size, the number of areas in a given branch and it's sub branches.

In the above image if the area marked A is selected as first area, the tree structure might look like this:
This sorted area tree is used to determine a story index for each area. The first area is at index 0, the next indices belong to the shortest branches and the highest index is at the last area of the biggest branch. This linear representation of the area tree can be used by the story generating systems to produce narratives that lead from one area to the next.

In the example from before the areas would be indexed in the following order:

A → E → B → D → G → H → I → J → F → C

5.6.9. Shading the Land Areas

Shading the land areas is done with a custom material that was created with Shader-Forge. The UV coordinates of the generated mesh are set up so that the area vertices are placed in the V coordinate range 0 – 0.5 while the border mesh has the V coordinates 0.5 – 1. The applied material shades the lower half of the UV map in a solid color with a pulsing transparency while the top half, the border, is generated by multiplying animated greyscale stripes. This gives the border an ethereal look. The base color for both the border and the area is the same. All areas get a hue assigned so that the hues are evenly spread out over the visible color spectrum, so each area has a distinct base color.

Finished world map
5.6.10. World Map Demo

A simple area selection system has been set up so that a generated world can be inspected. To switch from area to area use the keyboard arrow keys, to select an area use the E key.

When an area is hovered over, the connections to its neighbors are visualized by curved and animated dashed lines. Pressing in the direction of a neighbor selects that respective area. The shader for the animated dashed lines was created with ShaderForge and is a simple panning effect with a repeated alpha texture. The color of each connection corresponds to the connected area.

5.6.11. Onomatology: Generating Names for World Areas

To reduce the chance of ever seeing the name of an area twice, they are procedurally generated as well. A prototype name generator was implemented for this project. The names are generated by combining predefined syllables with single consonants and vowels over and over. The system is designed so the chance of adding more elements to a name are reduced with each element added. The name generator follows simple rules so that a name can not only consist of consonants and there is a limit how many consonants can appear in a row, etc.

There is also a chance that a predefined prefix and / or suffix is added to the finished name.

All generated world areas are assigned a generated name by this system. Here are some generated example names:

- Ballathett
- Æti
- Blen’nide
- Yster
- Adude
- Enoos Rock
- Chrohearth
- Everie
- Fooham
5.7. Modeling Simplified Social Dynamics

As discussed before, the goal for the story generation system was to have conflicts between procedurally generated virtual characters as the starting point of every generated narrative thread. For this a simplified model of social dynamics has been created. This was designed to generate characters with varied motivations to use as initial parameter for the quest generator designed by Jonathon Doran and Ian Parberry. The proposed model of social dynamics does explicitly not claim to be an accurate model of real world social dynamics in any way.

5.7.1. Virtual Characters

The virtual characters in this project have the following properties:

- **Gender:** This can be either male or female
- **Name:** First name + last name
- **Moral alignment:** à la Dungeons & Dragons, can be either good, neutral or evil
- **Ethic alignment:** à la Dungeons & Dragons, can be either lawful, neutral or chaotic
- **Wealth:** A generic floating point value in the range 0.0 - 1.0, where 0 is very poor and 1 is very rich
- **Intelligence:** Normally distributed floating point value in the range 0.0 – 20.0, where 10 is the average value
- **Athleticism:** Normally distributed floating point value in the range 0.0 – 20.0, where 10 is the average value
- **Charisma:** Normally distributed floating point value in the range 0.0 – 20.0, where 10 is the average value
- **Virtual Social Position:** A random normalized three dimensional coordinate (range [0, 0, 0] - [1, 1, 1]). This is used for generating the relationships between characters
- **Absolute Social Power:** Floating point value generated from all the other properties. This is used for creating a social power gradient between characters
The algorithm to create the normally distributed intelligence, athleticism and charisma values is the Marsaglia polar method (Wikipedia: Marsaglia polar method 2016). The distribution for each character attributes forms a bell curve. There is no adjusting done between the three attributes, a character’s intelligence has no influence on his or her athleticism or charisma whatsoever. A character can have low or high values in all the attributes, as well as a mix of low and high values.

5.7.2. Relationships between Characters

Each virtual character object also stores a list of relationships to other entities. In principle entities could be either other characters or groups of characters. At the time of writing this document, the system only generates character entities. However the system could be expanded to cluster multiple characters into groups and have the groups as a whole have relationships to other groups and characters. Characters might for example be hostile towards a whole group of other characters, which would be modeled by a virtual relationship.

The formula to generate the absolute social power $ASP$ is the following:

$$ ASP = (SC_c + SR_c) $$

$SC_c$ is the intrinsic social value of the character computed as

$$ I_c + A_c + 2C_c $$

where $I_c$ is the character’s intelligence, $A_c$ the athleticism and the $C_c$ charisma. This means here charisma is socially more important than intelligence or athleticism. For character attributes in the range 0 - 20 the range of possible values for the absolute social power thus is 0 – 80.

The social value of all relationships of the character is computed as

$$ SR_c = \left( \frac{t}{(t + 5)} \right) * 50 $$

with the total social power of the relationships

$$ t = (\sum R_w) \ [+ SC_s] $$

where the intrinsic social value of the characters spouse is $SC_s$, if he or she has one, and the social weight of each relationship $R_w$ based on the social power of the character in that relationship as defined in this table (see also the next section about social
powers in relationships):

<table>
<thead>
<tr>
<th>Role</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>2</td>
</tr>
<tr>
<td>Dominant</td>
<td>1.5</td>
</tr>
<tr>
<td>Peer</td>
<td>1</td>
</tr>
<tr>
<td>Submissive</td>
<td>1</td>
</tr>
<tr>
<td>Slave</td>
<td>-2</td>
</tr>
</tbody>
</table>

So the more relationships a character has, the higher will be the his or her social value (unless their social power in the relationships is often that of a slave), though the first couple of relationships are worth more than later ones. The social value difference between having 3 or 6 relationships is very large compared to the difference between having 103 or 106 relationships. The range of possible values is 0 – 50.

This overall results in floating point value in the range 0 – 130, where 0 would be a very low absolute social power and 130 a very high absolute social power.

The relationships of a character store the following information:

- **Self:** A variable pointing back to the character himself
- **Other:** The entity with which the relationship is formed
- **Emotion:** The general emotion describing the relationship, can be hostile, neutral or friendly
- **Family role:** This can be spouse, related or unrelated
- **Social power:** The difference in social standing between the two entities. The social power is graduated in the following order:
  - Master > Dominant > Peer > Submissive > Slave
- **Knowledge:**
  - **Location:** Does the character know, where the other entity is?
  - **Status:** Does the character know the current status of the other entity, is it deceased or alive?
  - **Belongings:** Does the character know what items the other entity owns?
  - **Others knowledge:** Does the character know what the other entity knows?
- **Tension:** Floating point value computed from all the social, moral and wealth differences as well as the emotion of the relationship
The values for social power are not about actual slavery but instead about relative social standing based on their absolute social powers \( ASP \). These social powers are assigned based on the following rules:

- **Master**: Own \( ASP \) is greater than other’s \( ASP + 30 \)
- **Dominant**: Own \( ASP \) is greater than other’s \( ASP + 10 \)
- **Peer**: Own \( ASP \) is within 10 of other’s \( ASP \)
- **Submissive**: Own \( ASP \) is less than other’s \( ASP - 10 \)
- **Slave**: Own \( ASP \) is less than other’s \( ASP - 30 \)

To give an example the social difference between the village idiot and a king will probably be master / slave, even if they don’t even know each other. A common merchant might be dominant to the village idiot, but could also be slave to the king, unless he is a powerful merchant and is only submissive or even peer to the king.

The formula to generate the tension value \( T \) of the relationship is the following:

\[
T = 2.5 \times |SP_S - SP_O| + 2.5 \times |M_S - M_O| + |W_S - W_O| + 7.5 \times E \times EF
\]

\( SP \) describes the social power (not the aforementioned absolute social power \( ASP \)) of the character him- or herself \( (SP_S) \) and the other character \( (SP_O) \) with values according to this table:

<table>
<thead>
<tr>
<th>Role</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>5</td>
</tr>
<tr>
<td>Dominant</td>
<td>4</td>
</tr>
<tr>
<td>Peer</td>
<td>3</td>
</tr>
<tr>
<td>Submissive</td>
<td>2</td>
</tr>
<tr>
<td>Slave</td>
<td>1</td>
</tr>
</tbody>
</table>

\( M \) describes the characters’ morality based on this table:

<table>
<thead>
<tr>
<th>Morality</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>0</td>
</tr>
<tr>
<td>Neutral</td>
<td>1</td>
</tr>
<tr>
<td>Evil</td>
<td>2</td>
</tr>
</tbody>
</table>

\( W \) is the wealth value of the characters.

\( E \) is an emotion factor from this table:

<table>
<thead>
<tr>
<th>Emotion</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friendly</td>
<td>0</td>
</tr>
<tr>
<td>Neutral</td>
<td>1</td>
</tr>
<tr>
<td>Hostile</td>
<td>2</td>
</tr>
</tbody>
</table>
EF is an emotion family role factor from this table:

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spouse</td>
<td>0</td>
</tr>
<tr>
<td>Unrelated</td>
<td>1</td>
</tr>
<tr>
<td>Related</td>
<td>2</td>
</tr>
</tbody>
</table>

Relationships can be one directional, when one character knows about the other but the other one has no idea about him. Or they can be mutual, when both characters have some opinion about each other. The scripts defining the relationships have been set up in such a way that it is easy to generate the corresponding relationship that goes from the other character to the first one.

5.7.3. Character Motivations

Every character stores a list of his possible motivations. A character can be motivated by more than one base motivation at a time. The motivation for generating a specific quest is randomly selected from the list of possible motivations. A motivation is added to the list of possible motivations when the following conditions are fulfilled (continues on next page):

Knowledge: The characters intelligence is in the top quarter of possible values (for the range 0 – 20 that is any value greater or equal to 15)

Ability: The characters athleticism is in the top quarter of possible values (for the range 0 – 20 that is any value greater or equal to 15)

Reputation: The characters charisma is in the top quarter of possible values (for the range 0 – 20 that is any value greater or equal to 15)

Wealth: The characters wealth value is below the current social systems poverty threshold or above the richness threshold. Only poor and rich characters are motivated by wealth, the middle class does not really care for wealth

Serenity: The character is ethically lawful
Equipment: If the character is motivated by neither of the above motivations, equipment is the first motivation added to his or her list of motivations.

Protection: The number of the characters friendly relationships is bigger than the number of hostile relationships. Characters with a lot of friends are motivated to protect them.

Conquest: The number of the characters hostile relationships is bigger than the number of friendly relationships. Characters with a lot of enemies are motivated to conquer them.

Comfort: The number of neutral relationships is bigger than the number of friendly relationships and it is bigger than the number of hostile relationships. Characters who have neither a lot of friends nor a lot of enemies are motivated to just lead a comfortable life.

5.7.4. Generating the Social System

At the core of this project is the generation of the characters and their relationships into a social system. The goal was to have enough characters to generate varied motivations for the quest generation.

The social system runs in multiple steps to create a network of characters connected by their relationships. First of all the system is initialized so that it has a reference to the world area data tree (see section „5.6.8. Clustering of Land Tiles for Division of the World into Playable Areas“ on page 26) and that the average wealth values are randomized. Three average wealth values are stored for each world, one for poor characters, one for the middle class and one for rich characters. This is later used for setting up the characters themselves. Wealth can have any value between 0 and 1, the three values are set to random values from these ranges:

- Poor: 0.01 - 0.05
- Middleclass: 0.40 - 0.60
- Rich: 0.95 - 0.99

To get more variety in the generated worlds, a basic main conflict setting is randomly chosen. The settings can be either war, then two factions of characters are created with one being mainly good and neutral and the other being evil and neutral. This
produces all kinds of characters.

The second main conflict is a cataclysm. Here all characters are good and neutral. The main driver of the story would be to unite with the generated characters against some kind of cataclysmic menace. This could be either the fortification against a natural catastrophe without actual antagonistic characters, or it could be some kind of demonic invasion of the world. These settings would not need a hostile character faction, fighting against the effects of the cataclysm would drive the story. This kind of main conflict is inspired by games where the player would be on the ominous quest to restore balance to the world, but no clear antagonist would be present. Even if in the end a named antagonist would be the game's final enemy, unless he is a major player in the narrative apart from that final showdown, this kind of story could still be considered a cataclysm.

The third and last main conflict setting is an escape. There all generated characters are inherently evil or just neutral. The players main goal here would be to escape from all the evilness in that world. To give a modern example of this kind of story one could look at Far Cry 3 (Ubisoft Montreal 2012) where the player has to escape an island full of mad criminals and other threats. Even though there are some friendly characters in that game, the main drivers of the story are the player and hostile characters.

The next step is the actual generation of the characters. Based on the randomly chosen main conflict, all generated character’s morals would be set up. The wealth of the characters would depend on a randomly assigned faction wealth. This base faction wealth is randomly chosen from the three values set up in the first step. The reasoning for this is to give each generated faction overall a similar wealth instead of completely random values. If the main conflict is a war and one of the two faction is overall very poor while the other faction is very rich, this can be used as reason for the war.

While all characters in a single faction share a common average wealth, some random variance is introduced so even inside one faction some tension can arise because of wealth differences.

All characters in one faction share that factions morality (good or evil) or they are neutral.

The characters’ intelligence, athleticism and charisma are all randomized based on a
normal distribution with an average of 10 and a variance of 10 (values can range from 0 – 20, but most characters will have values close to 10).

Each character is then assigned a random virtual social position. This is a virtual normalized three dimensional coordinate. For each character the virtual social distance to the other characters is measured. Characters that have close virtual social positions automatically have a relationship. If the characters are very close (distance < 0.1) and of opposite genders, their relationship is marked as a spousal relationship. If a close character already has a spouse or is of the same gender, they are set as relatives.

If the characters are somewhat close, but not close enough to fall into the first category (0.1 < distance < 0.4), the characters also have a relationship, but they are marked as unrelated.

When characters are created further than 0.4 virtual units away from all other characters, they get an unrelated relationship to the closest other character, so that no character can have zero relationships.

A visualisation of the social system is on the next page.

The relationship data (see section „5.7.2. Relationships between Characters“ on page 32) is mostly randomized. The relationships of spouses are always assigned a friendly emotion. The relationship of related characters can be friendly, neutral or hostile (conflicts can very much happen in families), but they are biased to be friendly more often than neutral or hostile. Unrelated relationships can also be friendly, neutral or hostile, but they are biased to be hostile more often than not.

It should be noted that this is a very limited and simple representation of actual social dynamics. Of course in the real world even spousal relationships can be hostile, to give an easy example. But again, the goal here was to model the social system so that it is somewhat believable and gives interesting relationships between characters that can be used to generate narratives.
Each sphere gizmo represents a character. They are placed at their virtual social position. Their size is determined by their absolute social power (ASP). Green characters are good, white are neutral and red are evil. Slightly lighter colored characters are female, slightly darker colored male.
The lines between them are their relationships. Relationships with a red cross symbol are hostile, green clover symbol represents friendly and the grey equal sign neutral. Little hearts show spousal relationships.
In this example circa 50 characters are created by the social system. As more characters are added the web of relationships gets denser and the whole system more complex.
5.7.5. Quest Tree Generator

When the generation of the social system is done, the provided demo scene randomly chooses one of the characters to be the client for a quest.

One of the clients motivations is randomly chosen for the quest tree generator. The quest tree generator is an implementation in Unity of the quest generator developed by Jonathon Doran and Ian Parberry.

For now the quest generator simply puts out the generated sequence of actions with the generated characters as clients and targets of the quest, as well as randomly chosen item names if a quest demands an item. The indentation of the actions corresponds to their depth inside the quest tree. They would be executed in the order they appear in, though.

Example generated quest

```
do quest for Leor Dien who is in Ufar
   --acquire Sword
   -----move to Sword by
   -----learn location of Sword
   -------acquire Sword
   -------acquire Arrow
   --------------move to Arrow by
   ----------------------------------explore randomly to find Arrow in Northvouple
   -------move to Stephan Marjoke by
   -----------go to Stephan Marjoke in Fiou
   --------exchange Arrow for Sword with Alan Starlin
   --------give Sword to Paulett Kimihiko
   ---------listen to Paulett Kimihiko to find out location of Sword
   --------go to Sword in Ungykirk
   -------pick up Sword
   ---explore with Sword
   ---move to Leor Dien by
   ------go to Leor Dien in Ufar
   ------report to Leor Dien
```
The quest generator works by randomly selecting one of the predefined sequences of actions (see Jonathon Doran & Ian Parberry 2011, A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs, pages 4-5) that correspond to the given motivation. Some of the actions are recursively replaced by other actions until only atomic actions remain.

If the quest generator encounters a <subquest> node, a new client for the subquest is chosen from the characters the original client has a relation to. The subquest is generated by choosing one of the motivations of that subquest client.

Some of the action sequences and action replacement rules defined by Jonathon Doran and Ian Parberry have been slightly modified for this project.

The output of the quest generator could be easily used to create full quest dialogues.
6. Results

6.1. Recapitulation of the Story Generation Process

The story generation follows a set of distinct steps.

First the world map is generated. The world map is generated from Voronoi tiles from a cloud of random points. It is then subdivided into different areas, which would be the game's different levels. These areas are sorted into a data tree structure. This is used to assign each area an index, so that for generated narratives the areas can be visited in a sensible order.

When the world map is done, the social system is created. Here randomized characters are generated and their relationships are set up by measuring the distance of their virtual social positions. Characters that are close in the virtual social space share a relationship. The closest characters are related or even spouses, other characters that are further away might be enemies.

For each generated character a set of motivations is determined. They are set up based on that character's relationships and intrinsic attributes. These motivations are then used to generate quests. The quests are made up of sequences of actions that are randomized according to a predefined ruleset. These action sequences could be used to generate dialogue and lead the player to the different world areas.

6.2. Future Work

To develop this project into a full game a lot of future work has to be done. While the core systems are actually in place to generate countless narratives, the game is currently missing it's basic gameplay mechanics.

For the story generation the actual dialogue templates and conversation presets have to be authored. To ensure that the player does not see the same lines by different characters over and over again, for every possible quest action many alternative lines should be written. A possible tool to make this task easier could be Rant (Nicholas Fleck 2015), a language for procedurally generating text.

While the characters have a lot of attributes to generate their relationships, these attributes are currently not used by anything else. It would make sense to use the attributes
to further define their roles in the world. The variables making up their personalities could also influence the way they talk. A lawful and intelligent character can be expected to talk very different than a chaotic brute. This would also increase the number of alternative lines that have to be written.

Another big section that has yet to be crafted consists of all assets needed to populate a procedurally generated world. This includes both character models, animations, textures and sound but also everything related to the environment.

The world map generation could be further expanded by creating more varied worlds. Right now all generated maps are coastal areas with grassland and mountains. Certainly other biomes like snowy glaciers or densely vegetated rainforests could be generated if the generation process and the used shader is adjusted.

While the world map in principle works, the actual levels where most of the gameplay would take place have to be crafted as well.

And of course the gameplay itself needs a lot of work to provide the player with a fun experience. The combat system has to be developed as well as all kinds of interactions with the characters in the game world.

Of course to be playable the game also needs a graphical user interface.

Last but not least almost all sounds and music for a full game are missing.

6.3. Conclusion

While the original goal was to create a playable prototype, the focus in the end shifted more towards the procedural generation of the world map and the generation of the social system.

The original quest generator by Jonothon Doran and Ian Parberry was very thoroughly documented, giving a good base for the system in this project. Their motivation based approach was a good inspiration for the social system. It also defined requirements, as the characters' motivations had to be the result of the social system's generation.

The different sub systems implemented in this project, the world generator and story generator, as well as all gameplay related to the action RPG, were created in a modular object oriented approach. They all were designed to be easily adjustable and expandable.
For every sub system and all shaders special care was taken to drive them by simple parameters. This makes it easy to tweak all aspects of the procedural generation. The use of magic numbers in the code was reduced to a strict minimum. The scripts also all follow a consistent coding style. These measures should enable the future work to expand this project into a full game.
7. References

Blizzard North 1996, *Diablo*, Blizzard Entertainment, Irvine, CA

Blizzard North 2000, *Diablo II*, Blizzard Entertainment, Irvine, CA

Blizzard Entertainment 2012, *Diablo III*, Blizzard Entertainment, Irvine, CA

Gearbox Software LLC 2009, *Borderlands*, 2K Games, Novato, CA

Gearbox Software LLC 2012, *Borderlands 2*, 2K Games, Novato, CA


Monolith Soft 2015, *Xenoblade Chronicles X*, Nintendo, Kyoto


Firaxis Games 2010, *Civilization V*, 2K Games, Novato, CA

Firaxis Games 2012, *XCOM: Enemy Unknown*, 2K Games, Novato, CA

Tarn Adams 2006, *Dwarf Fortress*, Bay 12 Games, Unknown


Square 1997, *Final Fantasy VII*, Square, Tokyo


AMPLITUDE Studios 2014, *Dungeon of the Endless*, AMPLITUDE Studios, Paris


Bethesda Game Studios 2011, *The Elder Scrolls V: Skyrim*, Bethesda Softworks, Rockville, MD


Ubisoft Montreal 2012, *Far Cry 3*, Ubisoft, Rennes

### Appendix 8.1

#### Example Generated Characters Motivation Distribution

**Average character count = 20:**

<table>
<thead>
<tr>
<th>Knowledge</th>
<th>Comfort</th>
<th>Reputation</th>
<th>Serenity</th>
<th>Protection</th>
<th>Conquest</th>
<th>Wealth</th>
<th>Ability</th>
<th>Equipment</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>10</td>
<td>5</td>
<td>20</td>
<td>9</td>
<td>17</td>
<td>14</td>
<td>8</td>
<td>92</td>
</tr>
<tr>
<td><strong>5.43%</strong></td>
<td><strong>4.35%</strong></td>
<td><strong>10.87%</strong></td>
<td><strong>5.43%</strong></td>
<td><strong>21.74%</strong></td>
<td><strong>9.78%</strong></td>
<td><strong>18.48%</strong></td>
<td><strong>15.22%</strong></td>
<td><strong>8.70%</strong></td>
<td><strong>100.00%</strong></td>
</tr>
</tbody>
</table>

**Average character count = 50:**

<table>
<thead>
<tr>
<th>Knowledge</th>
<th>Comfort</th>
<th>Reputation</th>
<th>Serenity</th>
<th>Protection</th>
<th>Conquest</th>
<th>Wealth</th>
<th>Ability</th>
<th>Equipment</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>13</td>
<td>2</td>
<td>1</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>4</td>
<td>21</td>
<td>3</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>17</td>
<td>5</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>22</td>
<td>18</td>
<td>26</td>
<td>22</td>
<td>37</td>
<td>33</td>
<td>69</td>
<td>17</td>
<td>2</td>
<td>246</td>
</tr>
<tr>
<td><strong>8.94%</strong></td>
<td><strong>7.32%</strong></td>
<td><strong>10.57%</strong></td>
<td><strong>8.94%</strong></td>
<td><strong>15.04%</strong></td>
<td><strong>13.41%</strong></td>
<td><strong>28.05%</strong></td>
<td><strong>6.91%</strong></td>
<td><strong>0.81%</strong></td>
<td><strong>100.00%</strong></td>
</tr>
</tbody>
</table>

**Average character count = 100:**

<table>
<thead>
<tr>
<th>Knowledge</th>
<th>Comfort</th>
<th>Reputation</th>
<th>Serenity</th>
<th>Protection</th>
<th>Conquest</th>
<th>Wealth</th>
<th>Ability</th>
<th>Equipment</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>17</td>
<td>13</td>
<td>28</td>
<td>7</td>
<td>2</td>
<td>98</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>12</td>
<td>9</td>
<td>23</td>
<td>7</td>
<td>24</td>
<td>5</td>
<td>0</td>
<td>98</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>7</td>
<td>12</td>
<td>23</td>
<td>10</td>
<td>0</td>
<td>12</td>
<td>15</td>
<td>103</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>14</td>
<td>3</td>
<td>23</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>5</td>
<td>94</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>15</td>
<td>13</td>
<td>19</td>
<td>17</td>
<td>0</td>
<td>10</td>
<td>7</td>
<td>103</td>
</tr>
<tr>
<td>58</td>
<td>49</td>
<td>56</td>
<td>41</td>
<td>105</td>
<td>54</td>
<td>60</td>
<td>44</td>
<td>29</td>
<td>496</td>
</tr>
<tr>
<td><strong>11.69%</strong></td>
<td><strong>9.88%</strong></td>
<td><strong>11.29%</strong></td>
<td><strong>8.27%</strong></td>
<td><strong>21.17%</strong></td>
<td><strong>10.89%</strong></td>
<td><strong>12.10%</strong></td>
<td><strong>8.87%</strong></td>
<td><strong>5.85%</strong></td>
<td><strong>100.00%</strong></td>
</tr>
</tbody>
</table>