**Ulm University** | 89069 Ulm | Germany

**Faculty of Engineering and Computer Science**
Institute of Media Informatics

# Exploration of Techniques for Procedural Content Generation of Levels in Serious Games

**Author:**
Michael Legner
800817
michael.legner@uni-ulm.de

**Reviewer:**
Prof. Dr.-Ing. Michael Weber
Prof. Dr. Helmuth A. Partsch

**Supervisor:**
Julian Frommel, Julia Greim

**Year:**
2014-2015

Print: PDF-LaTeX $2_\varepsilon$

Name: Michael Legner Matrikelnummer: 800817

**Erklärung**

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den ............................................................................

Michael Legner

# 1. Abstract

The aim of this thesis is to explore the current state of the art in procedural content generation for levels. Based on the gathered knowledge, a library is developed to be used in the development of serious games.
First, the thesis provides an overview over the usage of procedural generation in commercial games, research and in the so called demo scene as well as a short overview over the field of terrain generation, one of the main subjects of this thesis. It further introduces the term serious games and gives an overview of projects in this field. The first part is rounded up by adaptivity in games.
Next, a very short history of video games is given as well as a definition for the term "serious games". The topic of procedural generation is then presented with the types of assets than can be created as well as a brief overview of the techniques used. Afterwards an evaluation and selection of game engines, with the selected Unity engine being used for the development later on.
The following sections focuses on the development of the library and gives a detailed report on the techniques used and how they were implemented. The thesis concludes with a summary of the goals achieved, followed by the limitations of the current work and improvements for future projects.

# 2. Acknowledgments

I would like to express my deep gratitude to my master thesis advisor, Prof. Dr.-Ing. Michael Weber for sparking my interest in the field of procedural generation giving me the chance to write this thesis.
Further I want to thank my supervisors Julian Frommel and Julia Greim for their continuous support and feedback during the creating of this thesis.
I would also like to thank Prof. Dr. Timo Ropinski for his advice on terrain synthesis and terrain sketching.
Additional thanks go out to Bernd Linder for his feedback on the thesis.

# Contents

# 1 Motivation

Procedural content generation has been used in games since the days of home computers such as the Commodore C64, BBC Micro or Apple II. This thesis aims to explore the current state of the art of procedural content generation for levels and use the knowledge gathered to develop a library, which can be used in the development of serious games.

## 1.1. Serious Games

Games have been played since the ancient times, with the main purpose of providing entertainment. By the turn of the 19th century, special games were used for strategic simulations and training, especially the military had great interest in using games this way. The first mechanical simulations were developed, mainly for military uses such as a riding simulator during the first world war [1].

The book Serious Games [2] by Clark Abt was first published in 1968 and coined the term. It describes games, whose main intention is not to entertain, but to educate and train. At the time, video games did not exist, so instead card and boardgames are described.
Today, it usually refers to video games following that concept. Due to the increasing power and multimedia capabilities they are an excellent platform. But there have been critical voices and problems of acceptance of video games as a new medium. The wide spread use of smartphones and new interaction concepts like the motion controller on the Nintendo Wii console attracting different groups of players and helped reducing prejudices toward games [3].

## 1.2. Procedural Content Generation

Most games use pre-defined assets such as textures, models and levels that are designed by artists and level designers. They are becoming more and more complex and expensive to create as technology allows for more detailed creations. Also, they are mostly static and do not change during the game, which could be intended to provide an equal experience for all players, but also reduces the replayability of the game.
Contrary to pre-defined assets, which are stored in a specified file format and loaded by the game, *procedural content generation* describes an approach using algorithms to describe assets, which are generated at runtime of the game. By making use of random number generators and other techniques to generate random structures an asset can be different every time a game is played.
Procedural content generation can also be used for other purposes like saving disc space through algorithmic modeling of assets. This has been done in the early days of computing when resources were spare and is still very popular today in the demo scene, where old hardware is used or strict size limits are imposed. Another application is to create challenges scaled to the players progress and abilities, which is used in some role-playing games.

## 1.3. Goal

The main goal of this thesis is to explore different techniques for procedural content generation. Also, a library to use with a selected game engine should be developed that can be used in future projects.

# 2
# Related Work

This chapters gives an overview over previous work that is related to the subject matter.

## 2.1. Procedural Content Generation

Procedural generation has been used since the early days of computing in games for various reasons. For some of the early games and games developed for research, the techniques that are used for generation are known, but for most commercial games it can only be speculated as the developers rarely release any information on it.

### 2.1.1. Early Games

One of the earliest uses was to save disc space, which was scarce in the early days of computing. With this technique, David Brabam and Ian Bell managed to have 8 galaxies with 256 star systems each [4] on a BBC Micro home computer which was only 48 kilobytes of memory in their 1984 game *Elite* [5].
Other games used it to increase replayability, like the 1980 game *Rogue* [6] developed by Michael Toy and Glenn Wichman at the University of California, Santa Cruz and University of California, Berkeley [7]. The game procedurally generates dungeons for the players to explore, fight monsters and gather items. Since it was not possible to save the game, every time the player dies or the game starts, a new dungeon is generated.
One of the most well known games to use procedural generation is the second installment of *The Elder Scrolls* series, *Daggerfall* [8]. The game has a vast game world, spanning about 161.600 square kilometers with 15.000 towns, cities, villages and dungeons, which were generated beforehand and modified. The about 750.000 inhabitants are generated at runtime, as well as the items and a number of quests aside the main story line. [9]

### 2.1.2. Current Games

One of the best know games to currently use procedural generation is the action role playing game *Diablo* [10]. Levels are procedurally generated as well as enemies and items, depending on the progress and player level.
Keeping the tradition, newer Elder Scrolls games like the fourth installment *Oblivion* [11] also use procedurally generation, mostly for quests, items and enemies, which scale with the players' level [12].
The indie game *Minecraft* [13] uses 3D Perlin noise for the voxel-based game worlds [14] and spawned countless of other indie games using similar techniques.
Elite: Dangerous [15], uses procedural generation to generate solar systems and missions, similar to its predecessor Elite[5]. The upcoming title *No Man's Sky* [16] generates an entire universe, which consisting of up to 18,446,744,073,709,551,616 planets (18 quintillion), based on a 64bit seed [17].

### 2.1.3. Demo Scene

Procedural generation is very popular in the demo scene, where old hardware is pushed to its limits to show graphical fidelity that was not thought to be possible on it. Others aim to produce impressive graphical showcases in the so called demos, which are a non-interactive program showing off graphical fidelity. Demos are divided in categories, which often impose strict size limits for the executable file. For example, the 4k category only allows executables with a maximum size of 4 kilobytes [18], which makes it impossible to use pre-designed assets. Instead, the available processing power is used to generate assets algorithmically. A good example is the game *.kkrieger* [19], which uses procedural generation for all content in the game, making it possible to put an entire, although relatively simple, game into only 96 kilobytes of disc space[20].

### 2.1.4. Research

The 2009 game *Galactic Arms Race* [21] is an example for a game developed for research, which uses an approach based on evolutionary algorithms to procedurally generates weapons based on the players usage and play style [22].
The generation of game mechanics is the goal set in the approach proposed by Joris Dormans [23] based on "key and lock" mechanisms and described as a formal grammar. The mechanism can be taken literally, but also in a more general term, where the first item is used to accomplish a certain task. In a different example, the key can be a weapon and the lock an enemy, which can only be defeated with the weapon.
Fernandez-Vara et. al. [24] took on the task to generate puzzles for adventure games. Contrary to those found in puzzle games, which generally have no context and therefore only have to be logical, the puzzles in adventure games have to fit in the narrative of the game. Some commercial games use randomized puzzles, but mostly as a form of copy protection, which needed a code card or wheel [25] to solve them. Notable examples are *The Secret of Monkey Island* [26] and more recently *Ankh 2: Heart of Osiris* [27]. But Fernandez-Vara et. al. wanted to go a step further and generate all puzzles in an adventure game. To do so they analyzed existing games and found patterns in the puzzles, which mostly consists of the same actions, like giving an item to a certain

character, disassembling items and creating new ones. They developed a small game which uses this concept to generate all puzzles. They succeeded, although the game proved to be repetitive as the patterns did not allow for too much variation.

### 2.1.5. Terrain Generation

Synthesizing realistic appearing terrain dates back to work of Mandelbrot in 1983 [28], using fractals to generate heightmaps. Other popular approaches include the midpoint displacement algorithm introduced by Fournier et. al. [29]. Different types of gradient noises can also be used to generate heightmaps, the most prominent being Perlin noise developed by Ken Perlin [30] [31].
While the aforementioned approaches generate heightmaps that look authentic, they are not based on physical principles. Erosion simulation emulates the natural process how terrains originated. The basic process is that material from higher regions is disintegrated and transported to lower regions, resulting in steep parts becoming more steep, while relatively flat segments are evened out even more. Kelley et. al. [32] were the first to introduce an approach to approximate terrain through stream simulation. A combination of fractal modeling and erosion simulation was developed by Musgrave et. al. [33].

**Terrain Sketching** is a technique to give users a tool to intuitively create prominent terrain features such as mountain ridges or rivers. The approach introduced by Zhou et. al. [34] is based on patches that were extracted from real world elevation data and placed along a sketch provided by the user.

## 2.2. Serious Games

The term Serious Game was coined by Clark Abt in his 1968 book of the same name [2]. Although focusing on card and board games, his definition of a game with the primary focus to educate and train instead of entertain still holds up:

> Reduced to its formal essence, a game is an activity among two or more independent decision-makers seeking to achieve their objectives in some limiting context. A more conventional definition would say that a game is a context with rules among adversaries trying to win objectives. We are concerned with serious games in the sense that these games have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement. [2]

In digital games, one of the decision makers is replaced by an AI or programming that simulates a player.
Michael Zyda proposed an updated definition in his article from 2005 [35]:

> a mental contest, played with a computer in accordance with specific rules that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives [35].

While the primary focus is on education, this does not mean that the games cannot be fun. In fact, it has been shown that when the games are, people are more likely to play them [36], something that a lot of previous educational games neglected and thus failed[37].

### 2.2.1. Military

Especially the military has a high interest for realistic training simulations. They provide much funding and developed one of the earliest serious games. In 1996, a modified version of Doom was used for military training, called *Marine Doom* [38]. The major difference is that the player dies a lot faster than in the commercial games, which requires the player to be more careful [39]. Today, the US military still uses technology developed for commercial games, most recently the CRYENGINE3 [40] was licensed for this purpose [41]. Another project by the US military is *Americas Army* [42], a team-based first-person shooter which is intended for recruitment [43].
Another game developed for the military is *Full Spectrum Warrior* [44], which saw a commercial release for XBox in 2004 and PC and Sony Playstation in 2005. The player

takes on the role of a marine commanding a squad of four soldiers in a middle eastern scenario, which was developed for the commercial versions. It is a strategy game and focuses on tactical commanding, but contrary to most other games of the genre, it is not played from a birds eye view, but instead the limited perspective of the soldier to make the game more realistic. The game was developed for commercial consoles initially to save costs and because recruits showed a high affinity for video games.

### 2.2.2. Commercial

On of the earliest games to be considered a serious game is the 1982 game *Math Grand Prix* [45] for the Atari 2006 console. While being a racing game, the cars only move if a player answers a math question correctly.
More detailed and sophisticated is *SimEarth* [46], a game developed by Will Wright who also created *SimCity* among other games of the "Sim"-series, which all simulate a certain aspect.In this game, players take on a god-like role and can adjust various parameters on a simulated planet. The game does not have a set goal, the player decides how the planet ends up with his decisions: habitable for intelligent life, an infernal hellhole or anything in between, with the former being considered very difficult. The game teaches the player about the development of planets and evolutionary principles.
A popular series of games with the title *Crazy Machines* [47] is intended to teach players about physics by presenting the player with physics-based puzzles. The puzzles represent abstract machines, they are considered solved if the machine is running. To do so, the player has access to a set of mechanical, electrical and optical parts, with some already placed unalterable. The games have been released since 2004 on PC and various mobile Platforms.
Among other awards, the 2013 established *Deutscher Computerspielpreis* (German computer games award) has a category specific for serious games. The 2013 winner *Menschen auf der Flucht*(people on the run) [48] [49] focuses on the topic of people fleeing from a civil war in Africa, which is rarely addressed in games.

### 2.2.3. Research

The game modification *1378km* [50] for *Half-Life 2* [51] by Jens M. Stober, a student of Karlsruhe University of Arts and Design spawned controversy due to its theme. The name refers to the length of the border between West and East Germany and players take roles in one of two teams: refugees who try to escape to West Germany or guards who try to stop them. Guards who have shot refugees have to stand trial at the end of the game, as it was during the so called *Mauerschützenprozesse* (wallshooting trial) from 1994 to 2004. While some people called it "tasteless", others saw it as a new way to relive historical events [52].
Developed to support the healing process of young cancer patients, *Re-Mission* [53] has not only shown promise but positive results in a study [36]. Players take control of a so called nanobot, navigate through the body of a cancer patient to destroy cancer cells and even manage treatment side effects. The results show that players acquire knowledge about the illness faster and an increased rate of self-efficacy, which is presumed to have led to changes in the behavior of the patients regarding their illness. This effect already showed when players played the game an hour a week, with a further increase for players who finished the 20 level game. Further research was conducted to understand how this effect could be replicated in future games.
A different setup was used in the project *Weatherlings* [54]: it uses game mechanics from collectible card games and is played on mobile devices. Characters on the virtual cards are used to battle others and have traits which are based on weather conditions. Combined with real weather data, being able to forecast for a short period of time gives the player significant advantages, but requires knowledge about weather and how it changes.
The game *Atomic Orchid* is based on a mixed reality approach, similar to the popular game Ingress [55]. Most players are equipped with a mobile device tracking their position and enabling them to complete objectives in a fictional scenario after a nuclear explosion. Other players take a commanding role in form of the so called HQ which is played from a computer, where they see the whole map and more importantly radioactive clouds, whereas the field players only see the level of radiation at their current position. Players

can communicate via text messages.

What all those examples have in common is that they rely on proven gameplay mechanics. This seems like a good idea, as it lowers the bar to enter a game and serves as a motivator for the players. Early educational games have a reputation for not being fun and as such failed to motivate players, but instead were only seen as another tool to force education on them [37].

## 2.3. Adaptivity

This sections gives a brief overview how adaptive systems haven been used in commercial games, serious games and research. They usually rely on the same or similar techniques as procedural content generation, the major difference is that they have to adjust during the game and are not generated once before the game starts.

### 2.3.1. Commercial Entertainment Games

Epic Games' multiplayer first-person shooter *Unreal Tournament 2004* [56] has eight difficulty settings when playing against computer controlled opponents (bots). The difficulty can also be set to automatically adjusting, which means the difficulty is switched in the game according to the players performance. In the authors experience, the changes are very noticeable to the player, as the behavior and skill of the bots vary widely.
The massivly multiplayer online role-playing game (MMORPG) *World of WarCraft* [57] introduced a system called Flexible Raid in patch 5.4 [58]. Raids are portions of the game available to groups of players, providing the most challenge in the game and the best rewards. Previously, raids had fixed group sizes for 10, 20, 25 or 40 players. Flexible raids scale the challenge according to the size of the group, which can range from 11 to 25 players.
MMORPGs usually consist of a vast world, which is segmented into areas which are designed for players of a specific level. As a high level player, coming back to areas

that are meant for lower level players can be boring as the enemies are too weak to provide any reasonable challenge. To counter this problem, *Guild Wars 2* [59] introduced a system called "dynamic level adjustment" [60], which scales the players level and attributes to match with the current area.
*The Elder Scrolls IV: Oblivion* [11] uses a system to scale enemies according to the players level and attributes to create a constant challenge [12]. A similar system [61] is employed in *Fallout 3* [62], a role-playing game of the same developer, Bethesda Game Studios.
*Left 4 Dead 2* [63], a cooperative first-person shooter, introduced a system to dynamically change the game's difficulty and pacing, called the "AI Director". It monitors players progress, location and status and places enemies and items accordingly. The goal is to create a different experience every time, which the developers call "procedural narrative" [64].

### 2.3.2. Serious Games

Göbel et al. [65] used adaptation in serious games developed for rehabilitation. Two games were created and both use sensors to monitor the players physical condition. The goal is to have players training without overburdening them, and the sensor data is used to adjust length and intensity to prevent doing more harm.
By using a player model based on knowledge space theory [66], Göbel et at. [67] were able to use adaptation in narrative based games. They divided the mode according to the three defined contexts learning, narrative and gaming to store specific information and used them to adapt accordingly.

### 2.3.3. Research

Game developer veteran Scott Miller proposed that instead of having difficulty levels for the player to chose, the game should adapt to the players skill and progress with a concept he calls "auto-dynamic difficulty" [68]. He proposes that only a few variables measured in the game are enough to make a reasonable assumption about the players

skill level. Then, other variables in the game that are not directly visible to the player are set accordingly, this should also be done subtle so that the player does not notice something is changing.
Based on this idea, which was initially tailored toward the 2001 game Max Payne [69], Charles et al. [70] developed a generalized model to determine the players skill and preferences. This is intended to adapt the game to cater to the individual player, instead of groups as most commercial games are designed. Games can adapt to the players skill and his preference, but the system can also be used to prevent players from exploiting oversights in the game design. For example, if a strategy or game mechanic is unintentional extremely powerful, players are very likely to use them exclusively to progress through the game faster and easier, at the expense of their experience of having beaten a challenge. Instead of using simple variables, which are not generalized enough to be used in games, they use an approach based on neural networks to determine the player model.

# 3 Theoretical Background

This chapter gives a brief introduction to video games in general and serious games in particular. Additionally, a brief overview of procedural content generation, the core concepts and techniques and how they are used in games.

## 3.1. Video Games

The term video game usually refers to games that are played on some sort of electronic device. The 1947 *cathode ray tube amusement device* [71] [72] is recognized as the first video game. Known for its usage of radar equipment as graphical display is *tennis for two* [73] developed and shown in 1958. As all previous video games were conducted at universities and other research facilities, Nolan Bushnell and Ralph Baer are seen as the men who brought video games to the commercial market. Nolan Bushnell is best

known for arcade machines and mostly *Pong* [74], which is often misconceived as the first video game, although it was a very influential. Ralph Baer is referred to as the father of home consoles, with his invention of the *Magnavox Odyssey* [75].
Over the following decades, vast improvements in processing power and memory available led to more beautiful presentations, complex gameplay and the ability to tell film-like stories in games. For the longest time, video games haven been seen as something only for children or nerds, which sit in their basement all day. New input devices like touchscreens used in the iPhone or Nintendo DS and motion controls as used in the Nintendo Wii have opened new markets and brought video games to all parts of society [3].

## 3.2. Serious Games

They major difference of serious games is that contrary to their entertainment-focused counterparts, the intention is to train and teach. This does not mean that they cannot be fun, especially since fun is major motivator and a reason why previous educational games failed with their intentions [37].

## 3.3. Procedural Content Generation

This section gives an overview of procedural content generation, what can be generated, which techniques are used in games in the past and present and why they are used in the first place.

### 3.3.1. Generateable Assets

Procedural content generation has been used in video games since the early days, for multiple reasons. This sections give a brief overview on what assets are commonly generated in games.

### Levels

Procedural generation of levels has been used in games to increase replay-ability and is a major part of the so called rogue-like games, which borrow concepts from the game Rogue [6], which was developed around 1980 by Michael Toy and Glenn Wichman at the University of California, Santa Cruz and University of California, Berkeley [7]. The game procedurally generates dungeons for the player to explore, fight creatures and collect items. One key concept is the so called permadeath, which means as soon as the player dies, the game is over and all progress is lost, there is no way to save progress and load it in case of failure. The player has to start again and will most likely get a different dungeon. More recent games introduce some kind of progression system, for example items that become available in future playthroughs once unlocked or global experience increasing the starting attributes. An example is the 2015 game *Ironcast* [76].
Based on the same techniques procedural generation can be used to create vast worlds, which are then modified by level designers. This technique is used to speed up the level creation process [77].
Another usage of procedural generation is to save disc space. Early home computers and consoles had very limited memory, storing a huge game world was not possible. Algorithms to generate levels take a lot less space up than pre-designed assets and only parts could be generated depending on the state of the game. As it was mostly not intended to generate random levels, the same seed for the generation was used every time, recreating the same levels every time, as seen in the game *Elite* [5] released in 1984. Todays systems have vast amounts of memory and processing power, but these techniques are still heavily used in the demo-scene.
A major part of creating a realistic world is vegetation. Creating it manually can be tedious, but there are solutions for it by using a form of formal grammar called L-Systems. For details see section 3.3.2 in this chapter.

### Textures

Procedurally generating textures has several advantages: the disc space requirement is very low as they can be generated with few lines of code. One problem with textures

in general is that they have to be made specific for a piece of geometry or otherwise they do not look fitting. Procedurally generated textures can be generated depending on the geometry and fit perfectly. Another problem with premade textures is due to their limited resolution, they are usually repeated in tiles when applied to geometry, which is noticeable and result in a less authentic look. Technologies like MegaTexture [78] can counter this problem, but at the expense of requiring huge amounts of storage space and textures have to be streamed to the graphics memory, resulting in loading artifacts when quickly adjusting the camera. Procedural textures can be generated with any resolution needed and with a few adjustments, variations of textures can be generated, for example different types of wood.
The major downside of procedural textures is due to their random nature, they are mostly useful for natural-looking textures like wood, marble or grass, as minor differences are not a problem but mostly desirable. The more specific features a texture should have, the more the process has to be controlled and is therefore slowed down as complexity increases. At some point, creating the texture manually is faster and easier.

#### Other

In role playing games, items can be procedurally generated. It is prominently used in the action role playing game series Diablo [10]. While the categories and graphics are set, the attributes are randomly generated depending on the progress of the game. A similar technique is used to scale enemies to the players level and progress, creating a constant challenge for the player, which is heavily used in the Elder Scrolls games [8] [11]

### 3.3.2. Base Techniques

This sections provides an overview over some techniques used in procedural content generation, with a focus on those used in the library and proof of concept game described in chapter 5. Other techniques are only briefly discussed.

### Noise

Various noise functions have been used for procedural generation since the very beginning. One of the most popular approaches was developed by Ken Perlin in 1985 [30]. The goal is to generate random structures instead of white noise. Since the generated structures are reminiscent of terrain heightmaps, they are often used for that purpose, although they are in no way physically correct.
Perlin noise uses gradients, which represent the slope of the tangent of the graph of a function[79]. As a type of gradient noise, it is generated by first generating pseudo-random gradients for every cell of an n-dimensional grid. The gradients should not be completely random, Perlin uses a specialized hash function which returns a value from a short set of randomized values. Next, interpolation between the gradients is used to create the noise function, usually polynomials are used. To further increase the natural look of the noise, a number of derivative functions are created and added up. The derivatives are created by doubling the frequency, which results in halving the amplitude. Thus the frequencies have a ration of 1:2 to each other and form "octaves".
The original source code by Ken Perlin is available [80], but written in a very compact manner which makes it almost necessary to decipher it. Also, the original algorithm's complexity has a complexity of $O(n^3)$, a version called Simplex noise that has a complexity of $O(n^2)$ was developed by Ken Perlin in 2001 and patented [31] [81] for dimensions of three and up. For a more detailed explanation of the method see the technical report by Whilem Burger [82].
Another application for noise is the generation of textures. Having a parametric description of a texture makes it possible to generate variations of textures, for example different types of wood. They are mostly used to create textures reminiscent of natural materials like wood or marble. Another usages is creating clouds, as 2D or 3D textures.

### Formal Grammars

The concept of formal grammars dates back to the work of Noam Chomsky [83] [84]. They are used to describe and generate formal languages. They consist of four parts: a start symbol $S$, a set of productions rules $P$, a set of non-terminal symbols $N$ (which

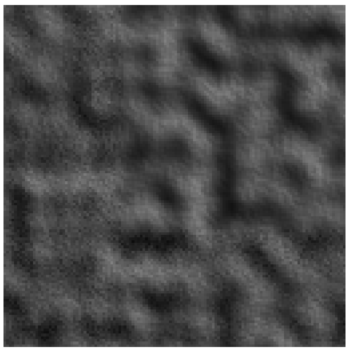


Figure 3.1.: Example of noise created with Perlin noise

includes the start symbol) and a set of terminal symbols $\Sigma$. They usually consist of text strings. The start symbol describes the initial state, while the production rules specify how the start symbol and non-terminal symbols can be replaced over the course of an iterative process called "production". Production ends when there are no non-terminal symbols left or can be limited by a set number of productions. An example for a simple grammar with start symbol $S$ and terminal symbols $a$ and $b$ is given in figure 3.2.

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

Figure 3.2.: Example of a simple formal grammar

Formal grammars have multiple uses, for example to create a parser for a formal language, commonly used in compiler construction. For procedural content generation they can be used to create missions as used by Joris Dormans [85] [86]. He uses a different form of grammars known as graph grammars. They are unique as for non-terminal and terminal symbols are not strings but nodes in a graph. To form the geometric structure of the level, shape grammars are used and consist of shapes.
Another popular type of formal grammars called Lindenmayer systems (or short L-Systems) are used to generate vegetation. They were developed by Aristid Lindenmayer in 1968 [87] as a way to describe the structure of plants. Similar to the usage of formal grammars with formal languages, L-Systems can be used to generate vegetation the same way. They are heavily used in the middleware *Speedtree* [88].

### Other Techniques

**Evolutionary Algorithms** are commonly used to find viable solutions to optimization problems in a reasonable amount of time. The perfect solution is often hard to find, but if a good one is already sufficient they are usually preferred because they are much faster to find. Based on principles found in evolution, where fitting solutions survive, evolutionary algorithms work on a set of different solutions and test them against a problem. Good ones are taken into the next iteration and for the others new ones based on the good solutions are generated. Other approaches include tournaments, where solutions are set against each others and the better one moves on to the next round.
In games, evolutionary algorithms are rarely used, most likely due to the amount of processing power required by them alone[89]. A notable exception is the action rpg *Galactic Arms Race* [21], which employs an evolutionary approach to generate weapons based on the players playstyle [22] [90].

**Celluar Automata** have been used to create cave structures in games. They consist of a grid of cells that have one of several defined states. Every cell has a defined neighborhood, usually the directly adjacent cells. By creating so called "generations" through iterating, the states of the neighborhood cells are changed over time according to a function defined, which can be displayed as an animation [91]. One of the most famous mathematical problems that can be described through a celluar automaton is Conway's Game of Life [92].
Celluar automata haven been used by Johnson et. al. [93] to generate caves for rouge-like games, with the states being floor, wall or rock. The grid is initialized with random values, over which is iterated a set number of times.
Aside from procedural content generation, cellular automata haven also been proposed to be used in cryptography to generate pseudo-random numbers [94] and error correction coding [95].

**Erosion Simulation** is an alternative method to generate terrain by simulating the natural process of erosion. The basic principle is that due to various natural effects,

material from higher levels is dissolved and transported to lower regions, eventually stopping when filling up irregularities. This means that steep parts become steeper and relatively flat sections are evened out even more.
These models are usually more accurate than the landscapes generated by noise functions that are trying to recreate the effect, but are also far more complex to implement. There is no built-in mechanism to ensure they are useful for games. Olsen et. al. [96] developed an approach which was implemented as cellular automaton and has been used in the commercial realtime strategy game Tribal Trouble [97], whose source code was released in 2014 [98]. Another example of implementing erosion simulation is using fluid simulation, as it was used by Neidhold et al. [99].

### 3.3.3. Discussion

While all the mentioned techniques provide good results in their specific field, a major challenge to use procedural content generation in a game is to ensure that the generated content is useful.
For custom generators, e.g. for items and enemies, the number of attributes is limited and can be kept within limits quite easily. But they still need lots of testing and tweaking, a negative example is the second installment of the The Elder Scrolls franchise, Daggerfall [8]. The game is infamous for generating pretty much everything: items, enemies, quests and large parts of the world. But this resulted in very mixed results as the random nature of many aspects of the game created illogical quests and therefore disgruntled many gamers [100].
With more complex content like a landscape, generating a suitable result is even harder. While the result may look good, all relevant places have to be reachable by the player. So generating a landscape without using further information can lead to results that has all the features of a landscape, but is useless for a game. The key is to generate something random, but which has a certain structure or satisfies constraints that keep the randomness at bay.

# 4 Technical Background

This chapter states the technical background of the thesis. It gives a brief introduction to game engines in general and documents the process used to select a game engine suitable for the thesis. Additionally, some important technical terms are detailed.

## 4.1. Game Engine

A game engine in general is a framework designed for developing video games. The main functionality include real time rendering of 2D and/or 3D graphics, input handling, audio output and physics simulation including collision detection. As the capabilities of hardware increased, other aspects such as scripting and networking became important and are a basic requirement for any game engine today.
With the limited resources available and the major differences on early hardware plat-

forms such as the Atari 2600/Atari VCS gaming console and home computers such as the Commodore C64, BBC Micro or Apple II, every game had to be written from scratch and adapted to the platform to get optimal performance. More performance available made it possible to abstract the technical aspects from the gameplay itself, helping development and portability of games. Another factor was the upcoming modding scene: players found ways to modify or hack a game, often in destructive ways. John Carmack of id software saw this as an opportunity and for their game Doom he developed a system to modify the game without overwriting existing parts [101]. In its successor Quake, the abstraction was complete as the gameplay code was written in the in-house developed language simplified version of C called "QuakeC" which is compiled into bytecode and interpreted at runtime [102]. Both engines created for Doom and Quake haven been licensed to other game companies [103].
The games industry today is a large market, even surpassing the film industry in terms of its net worth [104], it is also a very diverse one. The so called "Triple A" titles with budgets of 7 figures or higher developed by teams of hundreds of people and advertised with an equal amount of money. These games are usually based on in-house developments or big engines always trying to be at the technical edge like the Unreal Engine by Epic Games [105] and CRYENGINE by Cryteck [40]. Developments costs have skyrocketed over the years and big advancements in graphical fidelity are usually tied to the release of a new generation of gaming consoles[106].
Contrary to big budget productions and game developers, in the last years more and more small, independent developers have started to develop games on a much smaller budget, usually resulting in less graphical fidelity with a focus on gameplay and stylized graphics. They usually use different development toolkits created with the smaller scale in mind, such as Haxe [107] or highly specialized frameworks such as RPGMaker [108]. A new market has opened in the form of mobile games, with vastly different controls and new challenges such as a limited battery life. While quite powerful, they have significantly less performance than consoles and gaming PCs, which were the dominant gaming platforms for decades. Developing for mobile platforms is generally cheaper, comparable to independent games, but still portability and the ability to share assets and code is advantageous in terms of development costs. With his in mind, new game

engines focus on portability and asset sharing, probably the most popular right now being Unity3D [109].

## 4.2. Engine Selection

One of the first goals of this thesis is to find a game engine suitable for the needs of serious games. Finding a suitable engine is critical as it will influence the course of the thesis and is not easily revertible, therefore the decision making process has to be done carefully.
The engine must have all the functionality needed to start developing a game without having to make changes in its core. The learning curve should not be too steep, as it will be used by others without major experience in software development. It would be beneficial if the gameplay code can be written in a language different than the core, which would make development easier.
For an early evaluation, a cost-utility analysis was done and resulted in three potential candidates, which then were examined in detail. With those candidates, a small game was developed to test its capabilities, ease of use and time needed to familiarize with the engine.
For procedural content generation, a suitable approach has to be chosen. Criteria are complexity of implementation, use in serious and commercial games, and performance. Also, the randomness of the results has to be controlled, not all will be suited as they have to be playable.
A small game was implemented using the game engine with procedural content generation as a proof of concept. It should be designed to have multiple levels of difficulty which affects the generation process. Manual adjustment by the player is sufficient, changing the difficulty according to the players performance and physical condition is planned for the future. The differences in the generated content should be as subtle as possible, because if the players notices that the difficulty is raising or dropping, it could have an impact on his behavior.

| Score | Weight |
|---|---|
| 0 | - |
| 1-2 | nice to have |
| 3-5 | medium |
| 6-8 | important |
| 9 | very important |

(a) Weights used in cost-utility analysis

| Score | Degree of Fulfillment |
|---|---|
| 0 | not available |
| 1 | poor |
| 2 | fair |
| 3 | satisfactory |
| 4 | good |
| 5 | very good |

(b) degrees of fulfillment used in cost-utility analysis

### 4.2.1. Cost-utility Analysis

Game development is not only a big industry, but also a widespread hobby among software developers. As a result, an enormous number of commercial and free, open and closed source game engines have been developed. The Wikipedia List of Game Engines [110] lists 253 engines and libraries in total, while the DevDB [111] on the developer resource website `www.devmaster.net` [112] lists 368 engines. It is not possible to try every one of them therefore they have to be filtered to a bearable number. Beforehand, engines that are not flexible enough are eliminated, mostly 2D and 2.5D engines and engines that specialize in a certain genre.
Cost-utility analysis is a form of financial analysis. In this work the so called "Nutzwert-analyse" is used, which differs from the Cost-utility analysis in English speaking contexts, were is mostly used in health economics. The basic idea is to define criteria, categorize and weight them. The process itself is by far not as elaborate as other processes in decision making, but here the main use is to filter a vast amount of candidates in a short time period. The weights and degree of fulfillment are set subjective and were updated multiple times over the course of the thesis.
The weights are set from 0 (not applicable) to 9 (very important), for a detailed list see table 4.1a. Degrees of fulfillment are set on a scale from 0 (not available) to 5 (very good), detailed in table 4.1b.

As reference, *DevDB* is used as it provides more information about the activity of the engine, with additional candidates added by recommendations of the research staff at Ulm University. First, only active engines were considered and all without or with a very small number of user reviews were eliminated. The list of remaining engines can be found in table A.1. Next, the cost-utility analysis is done, the criteria are listed in table A.2. The following section describes the criteria in detail and the one afterwards discusses the results.

**Selected Engines** The selection consists of some well known and lesser know engines. Generally, open source solutions are preferred, as they provide the ability to change the core if needed (which is not intended) and independence from companies. Most of the proprietary engines have multiple licensing levels, including access to source code but only for a large sum, which is not suited for an academic budget. Some engines are only available through a licensing fee, but may be available for academic purposes at a reduced fee or even without cost. This is the case for the Unreal Development Kit, which is a free version based on the Unreal Engine 3 by Epic Games. It was initially not a candidate, since the focus of the engine is on big budget titles. But with the increasing importance of small, independent game developer for the industry, having a cheaper version with less functionality can attract small developer teams and even students. Therefore, Epic Games (and other companies) have started a program targeted to get more academic users into Unreal Engine 4 [113].

### Criteria

In this section, the criteria used in the cost-value analysis are detailed. An overview is shown in table A.2.

**License** While game engines with open source licenses are generally preferred, closed source solutions are not ruled out if other criteria have good scores. Therefore, open source licenses get a four times higher weight than closed source, with the other scored with 0.

**Technical Aspects** Technical aspects cover criteria programming language, supported functionality, architecture and supported platforms.

Most game engines are written in C/C++, mostly for performance reasons, but some can use code written in a different language through bindings or interpreters such as Python or .NET languages such as C#. This is beneficial as C/C++ is not in the curriculum of the major lectures at Ulm University, which mainly focuses on Java as initial programming language. Since most students are unfamiliar with it, they tend to have an avoidance of lower layered languages and often only heard of the downsides (pointer arithmetic in particular), at least having an option to write code in a different language would benefit adoption rate. Changes at the core of the engine are not intended.
Functionality is not a key point, as most engines support all the basic functions in terms of graphics, sound and input. Advanced features, especially in graphics, are not a hard requirement.
Supported platforms focus mainly on computers with Windows, Linux or MacOS. Having the ability to deploy a game in a browser environment would benefit testing and conducting studies as setup time would basically be non-existent. Mobile platforms, in particular Apple iOS and Android would also be beneficial. Availability of either of them or both will be rewarded with additional points. Additional platforms such as gaming consoles are of no additional value and are not rewarded with points.

**Activity of Developers** Since the engine decided on will be used in future projects, it should be actively developed and maintained by the developers to avoid running into a dead end or having to do more work on our own which not is not tied in with developing a game, but fixing bugs and developing additional functionality. Updates should be released in regular cycles. Having developers actively interacting with the community would benefit the learning curve or time needed to fix problems, as quick help would be available.

**Tool Support** Having the ability to use the engine and develop games in familiar Integrated Development Environment (IDE) would be beneficial in terms of familiarization with the engine. In our case, eclipse is the standard IDE, although mostly students can choose what they want to use as long as all needed functionality is available, but are on their own then. The ability to use assets from artistic tools to create models, animations and textures would also be of benefit. If the engine itself provides tools for tasks like programming, modeling or level design it would reduce development time and the need to find compatible solutions.

**Community** An engine can be good on paper, but if it is not widely used it is for benefit us. A widespread engine with experienced users can help easing the learning curve and solving problems quickly. Generally the number of active projects and their general progress is a good indicator, although not all problems are rooted in the used technology.

**Ease of Learning** The most important aspect is the ease of learning of the engine. As the projects will mostly be developed by students, it cannot be expected from them to have much experience in developing projects of this scope, they are actually meant to gather experience first hand. Therefore, having a too complex engine is of no benefit. A steep learning curve is probably the most significant aspect, which can be eased by having sufficient literature (ideally both analog and digital) and tutorials. Seeing the scope of a game engine, good documentation is a must and would be beneficial to developers. Having support directly from the developers can also be a great help, as their knowledge of the inner workings of the engine usually surpasses that of an average user which would benefit in solving problems related more to the internals. This criteria is closely tied in with the general activity of the developers.

### 4.2.2. Preliminary Results

In this section, the results of the cost-utility analysis are presented and discussed. For detailed results, the tables for each engine are found in the appendix A.2.

#### Open Source Game Engines

The **Blender Game Engine** is an extension to the modeling tool Blender [114]. Overall it is a decent engine, but it lacks developer support and in the future it will be developed to be a tool for game prototypes, architectural walkthroughs and scientific simulators [115], which is not the direction we want and is most likely going to be a dead end.

**Cafu** is also a decent engine with an outdated feature set for visuals. It seems like development has not stopped, but slowed down a great deal to the point where the commits to the source code version control system only appear weeks apart, are small and the last official release dates back to 2012 - not a good outlook and a potential dead end. **Crystal Space** is a very similar case, both have not been used in a lot of projects and activity by developers and users is decent.

**Irrlicht** is probably one of the oldest still developed engines and is used in quite a few open source and commercial projects, but the last official release was in 2013, commits to their version control system have slowed down. The biggest downside is that it is mainly a graphics engine, everything else has to be provided by add-ons which might lead into a dead end if development of the plugins stopped or the interfaces in the core engine changes. Overall a solid engine, but it lacks additional tools and steady development.

**jMonkey** is one of the two Java-based engine in this comparison and has a good set of features with everything needed from a programmers standpoint and good platform support. Learning curve seems good and quite a few projects are active, but it seems that the development of the engine has slowed down more and more over the last months. A major downside is the lack of support for common file formats of modeling tools, only Blender models are currently supported. It is possible to run jMonkey projects in a browser, but only as a Java Applet, which got a bad reputation for being a security risk due to flaws in the Java Runtime Environment.

Its competitor **libgdx** is heavily developed with new releases every couple of weeks and an comparable set of features. It also only supports a few model data types, mostly from open source software like Blender and brings some editors with it, but mostly for minor task like creating particles. On the plus side it supports all needed platforms and due its highly active development, **libgdx** makes it into the final evaluation.

**OGRE** is a very old engine but still actively developed, with a big community and many tutorials and literature available. Despite having a competitive score, it will not be evaluated in the final round. The reason is that it is mainly a graphics engine and all additional functionality is provided via add ons. This could be fatal if they are not

developed by the core-developers and interfaces change, which is a risk not worth taking and the other features are not enough to justify it.
The last of the open source engines is **Panda3D**, formerly developed by Disney as closed source and now in the hands of the Carnegie Mellon University. It has a solid feature set and has been used in a good number of free and commercial projects. It is a solid engine overall, but will not be considered further as it has almost no active developers. It seems like it is only developed if a student of CMU uses the engine, but otherwise not much happens. And it does not stand out in any other criteria to justify a place in the final evaluation.

### Closed Source Free of Charge Engines

**Esenthel** is very interesting engine with a feature set comparable to commercial engines and passable licencing fees if desired, but they are bound to the number of developers. The community and number of projects seems small, but is highly active as is the development of the engine. It supports all desired platforms and has a good number of tools with it. The biggest downside is that it can only be used with C/C++ code, which might make it hard to convince student to use the engine, but due to its otherwise high score it is included in the final round of evaluation.
The **NeoAxis** engine is based on OGRE, but greatly enhanced by visual updates, more features and editors. It is actively developed, with major releases every six months. It is overall a solid engine, but lacks features that stand out and lacks documentation and tutorials.
**Shiva 3D** seem overall like a solid engine, supported platforms and a features set competitive to commercial engines stand out. The downside is that development seems rather troublesome, the next big version is overdue for 2 years with a beta version only released recently, the last major version being more than four years old with a small bug fix version released in December 2013. Despite otherwise seeming like a good engine, this could prove troublesome which is reflected in the scores and therefore it does not make it into the final round.
**Unity** is probably one of the most used engines at the moment, from small two man

teams up to big budge productions. Its feature set is unparalleled by most other engines, especially in provided tools which make almost every other software (including an IDE and modeling tools) unnecessary for small projects. It is very actively developed, well documented and the large community provides much help and tutorials, in addition to the already huge number provided by the developer. It supports all desired platforms and gameplay code can be written in convenient languages (C#, Boo (Python inspired language) and UnityScript (JavaScript dialect)). In combination, Unity achieves the highest score of all engines and is therefore included in the final round of evaluation.

A bit of special place takes the **Unreal Development Kit** and **Unreal Engine 4** in this comparison. Both are mostly used by big budget products with professionals of all needed crafts involved. This makes for a features set even outperforming Unity, especially in graphical fidelity, where the engine is one of the most impressive on the market. To bring smaller teams to use the engine, developer Epic Games has released the UDK for free, which is based on the Unreal Engine 3. It has all the major features and a huge amount of tools provided with it. The downside is the complexity, which might be too much for small projects and students not familiar with a software this size and coded in C++. Additionally, the current Unreal Engine 4 is available for academic use for free (but still 5% of gross revenue has to be paid for sold products [116], which is not the case here), but with even more functionality and complexity, it is left out of the final round of evaluation despite having the second highest score over all. But it might be a viable option in the future, especially if students already worked with it.

#### Results

In the final round of evaluation, three engines will be looked at in detail: Unity (Score: 486), Esenthel (437) and libgdx (429). Every engine will be used to develop a small game to see how fast a usable result is possible and the amount of familiarization needed.

### 4.2.3. Detailed Evaluation

After the cost-utility analysis, three candidates have been chose to be further evaluated in detail: **libgdx**, **Esenthel** and **Unity** which achieved the highest scores overall. Although having a very high score also, the Unreal Engine 4 was left out as its primary focus on big budget production will likely result in a steep learning curve, which is not suitable for our needs.
For the remaining candidates, test installations will be done and a small game will be developed with each to find out how quickly and easily first results can be achieved. To learn more about the engine, documentation and tutorials will be studied in greater detail than before.

#### libgdx

libgdx is a game engine in the most classical way and the only open source engine in this detailed evaluation. It does not bring much in terms of additional tools and everything is done in code. It operates rather near the core of the engine, as shown in the examples, updates are done in a render() callback method. It can be decoupled from it by using an event-based system, otherwise update frequency depends on the frame rate which can lead to unexpected side effects. Although the engine is written entirely in Java (with bindings to OpenGL), performance is very good.
It comes with an installer tool which can generate projects for eclipse and IdeaJ, but this option is well hidden. Otherwise a generic Gradle [117] project is generated which can be imported in various IDEs given they are compatible or a plugin is installed. The installer downloads the latest version of libgdx and all required libraries, in particular the Java build system Gradle [117]. It supports all major platforms and only very little platform-specific code has to be written.
libgdx is relatively new and is actively developed with new builds almost every week. They usually do not break older programs, although it has happened a few times in the past. There is a good amount of tutorials, although one has to be careful if its based on an older and outdated version. The documentation by the developers is sufficient and the provided examples provide a good start, but advanced techniques are only used in

the examples without further explanation and use sophisticated programing techniques common in game engines, especially matrix transformations, but are hard to understand without proper documentation and prior knowledge.
From a procedurally generated content (PCG) standpoint it should not be too difficult. libgdx does support only a few types of geometry data, among them the Blender format which also allows to import complete scenes and a json-based format which saves the vertex coordinates.

### Esenthel

Esenthel is a closed source game engine written in C++ with a lot of advanced features. There is no free version, only a demo version which misses critical features like deployment to release. Licenses are rather cheap, with a monthly subscription model for about 10 US-dollars a month, a license priced 19 US-dollars a month gives access to the source code. Modern rendering techniques are supported such as Bump Mapping, Tesselation and Screen Space Ambient Occlusion [118]. Although code is written in C++ only, there haven been made changes to circumvent some of the quirks, for example the order of function declaration does not matter and prototypes are not needed, which makes the code somewhat reminiscent of Java or C#, but at certain points relies on C++ techniques like pointers. The engine does not bring a C++ compiler and debugger but instead needs external tools. On Windows, it relies on the infrastructure provided by Microsofts Visual Studio. It uses a client-server-structure at all times, even if only one developer wants to work he first has to start the server and then open the project, which essentially starts the client environment. It features a code editor with auto completion (although sometimes overzealous), a world editor and tools to edit models and textures to a certain degree. They are fine for beginners, advanced users might want to use other, more specialized tools. It supports a good number of formats for assets, as long as they are not patent protected like music in mp3-format. Import can be done via drag & drop and every asset is given an unique identifier (UID). The included editors displays a defined name instead of the UID, but as soon as the project is exported, for example for debugging, only the UIDs are visible which can be confusing. The code editor itself is

good, but to get its full potential it requires a certain workflow, otherwise development can be painful. Other tools include a complete GUI System supporting most standard graphical user interface (GUI) elements like labels, buttons and drop down menus.
Esenthel is a rather unknown engine with a small community, therefore the number of tutorials and additional material is very limited. The official forum is very active with the main developer himself answering a lot of questions directly. Most of his answers though are very technical, often consisting of only code using advanced C++ concepts. The official tutorials consist only of code, which is well documented but still only code. They can be found on the server as its own tutorial project and cover all major aspects of the game engine and its features. Additionally, there are video tutorials covering other topics like the usage of the graphical editors.
Having an integrated project structure can be a good thing, but Esenthel takes the idea a bit too far. The projects can be structured with sub folders, but they appear only in the editor. On the hard disk, in the projects directory are folders with seemingly random names, which contain some named files and a lot of files with random names and no structure. These are all the assets and code which makes it hard to identify them and can be painful to use with a version control system. The server software can be installed on a server for collaborative work, but has no version control integrated which is a must in modern software development.
Since all code is written in C++, for deploying an app for Android the native development kit is required as well as the very outdated Android SDKs Version 1.7 and 2.3.3 as well as Java Development Kit version 7, version 8 is not supported. Once they are all installed and correctly configured, deployment runs smooth. Deployment to a web-based environment could not be tested as only the demo version was used for testing, which does not support this feature.

#### Unity

Unity is a closed source engine with a huge following, no doubt due to the licensing model which gives the basic version away for free which also allows commercial releases. The pro-version has advanced features especially in the rendering and performance

department and is useful for big budget productions. Unity claims to have a market share of 45% with 3.3 million registered developers and over 600 million players [119]. It was also one of the first engines to lay its focus on easy multiplatform development. It is mainly used for small games developed my small and medium sized, independent game developers, but is also used for big budget productions like Wasteland 2 [120] with an approximate budget of over 3 million us dollars from crowd funding [121].
Unity deviates from the other engines in this evaluation due to its focus on scripting the game behavior and does not require knowledge about game engines and computer graphics in general, but they can be useful to increase performance. The basic element is the `GameObject`, which can have multiple components like scripts, sounds, geometry and `colliders` used for hit detection. Every `GameObject` handles its own behavior in the attached scripts, which can be written in C#, Boo (Python inspired language on Microsofts' dotNet platform) or UnityScript (JavaScript derivate), communication with other objects is done via an internal messaging system or the event system of dotNet. The game loop is not directly visible to the developer, instead every `GameObject` with a script component attached that derives from the class `MonoBehaviour` has an `Update()` callback which gets called every time a new frame is rendered. For programming, the MonoDevelop IDE is brought along but can be replaced with any other external editor.
Unity supports a lot of formats for assets like textures, sounds and models and converts them to a suitable format on import. All assets are organized in a folder of the same name which can have user created sub-folders for structuring. Most of the work can be done in a graphical interface, from creating objects to adjusting parameters, even while the game is running and the changes are immediately shown. Additional assets can be downloaded and bought from an integrated asset store. A physics library for both 3D and 2D environments is integrated and can be used though scripts or component properties. A great deal of changes can be done through the graphical interface, but it takes some getting used to, especially in finding all the relevant options. Optionally, a server software for collaborative work and version control is available for purchase [122]. Of all the engines in this evaluation, Unity certainly has the biggest community by far and therefore an enormous amount of tutorials and additional material are available: books,

videos and magazine articles. The developers provide a manual with guides in text and video form to all relevant information and always up to date with the current version of the engine.

### 4.2.4. Result

**libgdx** is a good engine overall with everything that is required but lacks features to help developers and tool support. While it would be a good engine to use for this thesis, I believe it should get more development time to stabilize its development and for the development of more tools.
**Esenthel** feels a bit like golden cage: very good feature wise, but almost no control over it. Combined with the fact that C++ code is required which is not in the standard curriculum of Ulm University and the well known dislike of many students for this language, Esenthel was not chosen for the final development.
**Unity** fulfills the prediction of being the best option after the cost-utility analysis: it requires the least amount of time to become acquainted with the engine, a big part plays the high abstraction from technical layers. With the main develop language being C#, which is close to Java and lots of additional functionality, Unity was chosen as engine for the next parts.

## 4.3. Data Structures

### 4.3.1. Heightmap

Heightmaps are used to describe a landscape in the form of a two dimensional raster image. They are used to describe a so called 2.5D terrain. The term is based on the fact that the geometric structure is essentially two dimensional with an added height component. This means for every cell with coordinates X/Y in a grid, there is exactly one Z-value (sometimes the coordinate axes are switched, with for every cell in a X/Z grid there is exactly one Y-value). This has the advantage of being a simple and efficient approach, but has the limitation of not being able to create any type of terrain. Overhangs

are not possible as they require multiple Z-coordinates. A similar technique is used by early 3D games such as Doom to cope with the limited resources available [123]. Although being three dimensionally rendered, every level is based on a two dimensional floor plan, which makes rooms stacked on top of each other or angular floors and ceilings not possible without tricks to emulate them, as used by the Build enigne developed by Ken Silverman [124]. A detailed explanation of the inner workings of the Doom engine (renamed to id tech 1 in 2007) was written by Fabien Sanglard [125].
In the following sections, the term heightmap is used when the finished product of the generation process is referred to.

### 4.3.2. Noisemap

In the following sections, the term noisemap is used to describe a a two dimension raster image, which is used during generation to store the noise generated by the Perlin noise [30] algorithm. In contrast to the heightmap, it describes the raw, unprocessed noise, while the heightmap is the finished structure.

# 5 Application

This chapter describes the concepts used in the library and proof of concept game.

## 5.1. General

In this thesis, three kinds of levels can be generated: **Landscape**, **cave** and **dungeon**. On every level, missions are generated, which take a role in shaping the level to ensure that all points of interest are reachable.
The **landscape** serves as a hub-level and consists of a basic height map and connects various other points of interest. The landscape will be generated once and then stored to not confuse the player when he searches for a certain location. A key aspect is the transition to a cave and dungeon, which are placed randomly and lead to newly generated levels every time, but can only be accessed once per play session.

A **cave** is a type of sub level and is generate from Perlin noise, which was used because it is suitable to achieve the natural look required.
**Dungeons** on the other hand should look like they have been built by humans, so the more natural and organic look resulting from using fractals is out of the question. A approach based on a graph structure is used instead.

### 5.1.1. Missions

The missions are the tasks a player has to do in order to complete the game and are the basis for generating terrain sketches, which will be explained in detail in section 6.2.4.
Missions are organized in an acyclic graph structure and consists of objectives, which are represented as the nodes. Some objectives have to be completed in a specific order, as they can have prerequisites, for example a has to be collected before chest can be opened. Others can be completed in any order, which is also represented by edges in the graph.
Every graph consists of a start and end node, which are introduced to make it possible to have multiple missions in one graph, although it is currently not used. The objectives vary depending on the scene they are generated for. The process of generating a mission is loosely based on Lindenmayer systems [87], although a lot simpler. A simple grammar is defined and the number of productions is based on the selected difficulty.
The nodes are placed on a `Texture2D` of the same size as the noise texture, the nodes are placed in segments, which are calculated through the size of the texture and the number of objective types in the graph, with start and end nodes also being a type. This results in a random distribution, but makes sure to use most of the available space, although they can be clustered. This can be solved by using a more advanced distribution, as projected in chapter 8.
There are different sets of possible nodes, depending on the level they are generated for. Landscape missions are meant to lead the player into caves or dungeons, which then have their own missions. They are only generated once upon the start of a game, whereas the missions in caves and dungeons are generated every time the player enters it. A player can chose to abandon the mission by returning to the starting point or use the

normal exit, but only after all objectives are completed. The mission on the landscape will be marked as fulfilled.

### 5.1.2. Landscape

Generating landscapes has been a very well researched topic and for this task, plasma fractals generated via Perlin noise are used. The approach used here is based heavily on one explained by a user called "scrawk" in his blog [126]. It uses the concept of 2.5D terrain, which means that for every point in a field of X and Z-coordinates, there is exactly one Y-value defining the height of the point. While this does not allow for certain landscape features like overhangs, it is a simple and effective method.
One heightmap is generated from two components: the plain- and mountain noise. Both have the same dimensions, the main difference is in the amplitude. Plain noise only has a very narrow amplitude, resulting in gentle terrain features. To generate distinct terrain features, the mountain noise uses an amplitude several times that of the plain noise, resulting in high spikes of values. As they are both converted into a `Texture2D` object, the negative values are all cut off at zero, resulting in completely even terrain at those spots. Both noises are added together, resulting in high amplitude spikes for mountains where the plain noise does not have a significant impact, while on the completely even places it generates a gently surface, resulting in a more realistic look.
The landscape makes use of the Terrain data structure provided by Unity, which has built-in features for tiles of geometry, splatmaps for texturing and more. Unfortunately, it seems that it is intended to be mainly used through the UI, as the documentation is rather lackluster.
Terrain can be split in tiles, defined by the neighbors property, which automatically smooths adjacent edges. This features was used in the early stages of development, but was deactivated with the introduction of terrain sketches. While not impossible, implementation of terrain sketches spread over multiple tiles was postponed due to time constraints.
As the terrain is generated at runtime, texturing is a problem. To solve this, texture splatting [127] can be used to dynamically apply texture onto the geometry, depending

on parameters. Texture splatting uses splatmaps, which are essentially alphamaps the size of the heightmap, defining the weight of a texture in the combined overall texture. In Unity, the two obvious parameters are height and steepness of a point. Depending on these two parameters, the mixture of textures can be calculated. In this case, textures for lower regions are grass texture, higher textures become more dirt and rock heavy, while at the top a snow texture is used. The steepness parameter is currently not used. The terrain class has a number of other features, including placement of trees and other small objects, which are currently not used.
A major problem when generating landscapes for games is the random nature of the process, which makes it possible to have points of interest in unreachable spots. To solve this problem, the concept of terrain sketches was adapted to the purpose.
In the approach by Zhou et al. [34], the concept was used to generated notable features of the landscape, in this case mountain ridges. First, an existing heightmap was analyzed on the characteristic features, which were extracted as small tiles. Those tiles were then applied to a simple graphic, containing a sketch drawn by a human, while the rest of the heightmap was filled with other features found in the existing heightmap and smoothed out.
This process was adapted to use the generated plain- and mountains noise described earlier. The sketch was generated based on the mission graph described in section 6.2.4, with the points of interest placed randomly in segments of the map based on the number of missions and to make use of the complete map. They are then connected as defined in the mission graph, which results in the sketch.
Instead of just adding the two noisemaps together, the concept is to weight the plains noise higher the closer it is to the sketch, and vice versa for the mountains noise. The result ensures that points of interest are reachable for the player, the look depends on the other noise, especially the mountain noise. It is possible to cut through a mountain region, which should be smooth.

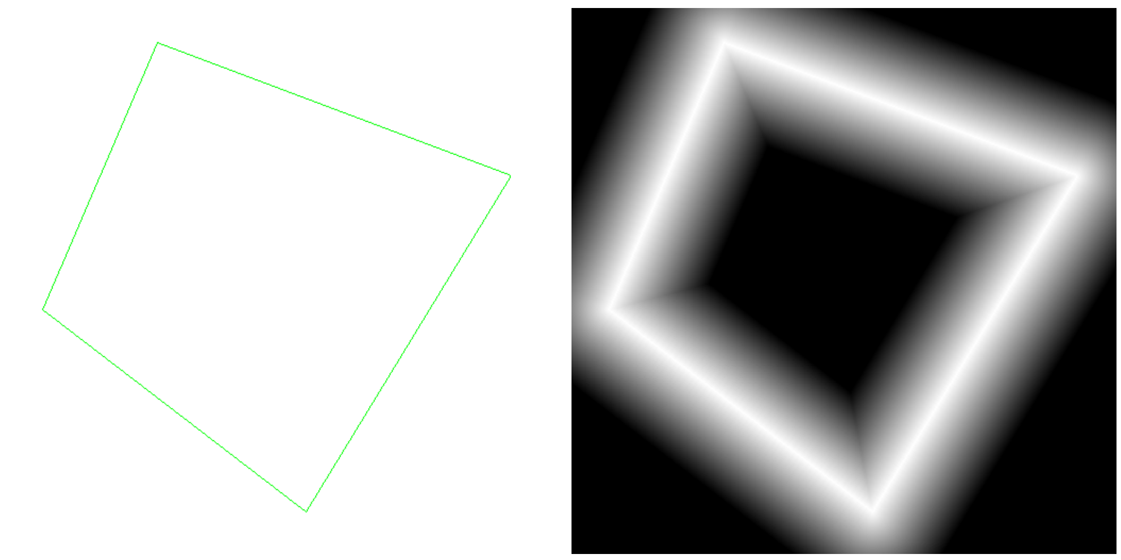

(a) landscape sketch (b) terrain weights map

Figure 5.1.: sketches for terrain

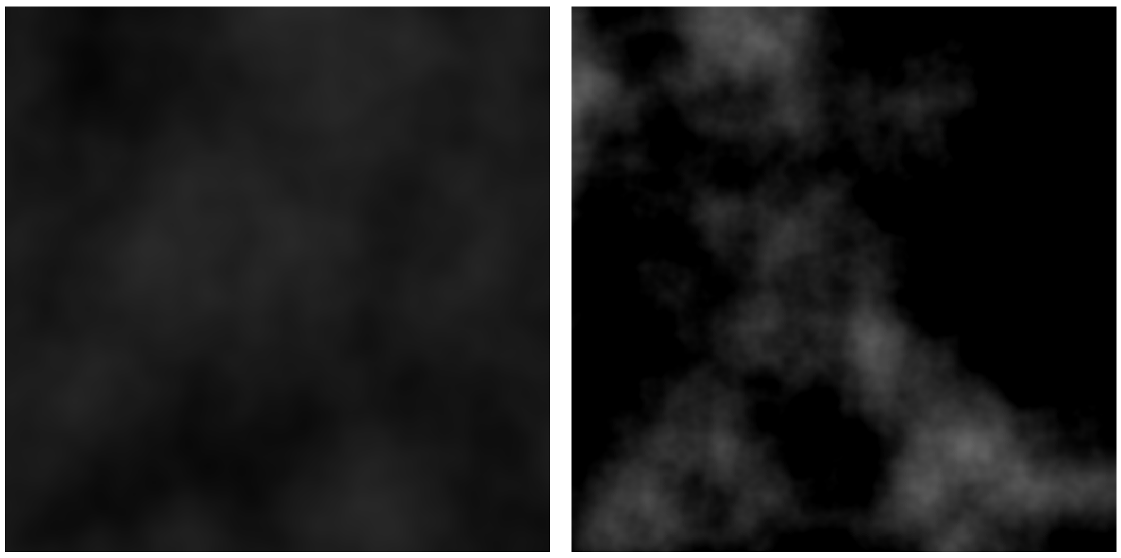

(a) landscape plains noise (b) landscape mountain noise

Figure 5.2.: landscape terrain noise

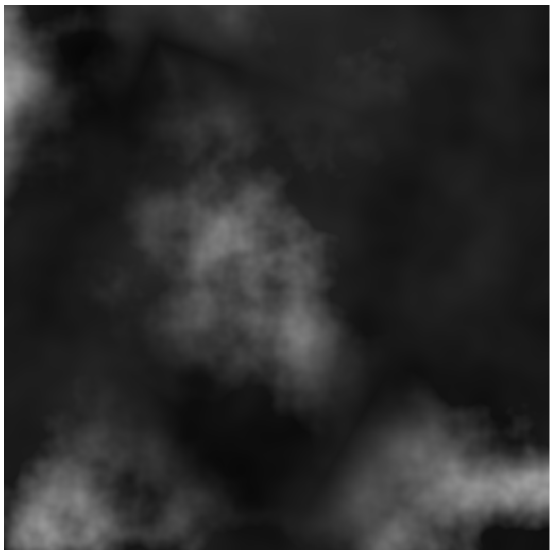


Figure 5.3.: complete terrain heightmap

### 5.1.3. Cave

Caves also use Perlin noise, but only one layer and the further processing is different: the noise is evaluated and if the values exceed a set or calculated threshold, they are set to the maximum value and vice versa. This results in a black and white noisemap with natural looking structures, the details depend on the settings used when generating the noise. A minor flaw is that all walls seem angled, as the noisemap specifies a 2.5D terrain heightmap and as a result, multiple Z-values for every x/y-coordinate are not possible.
The noisemap is then used to generate a mesh, where the color values are used to determine the z-coordinates of a vertex. Dynamic texturing similar to splatmaps in the landscape was not implemented due to time constraints and because graphical fidelity was not a high priority. The mesh generator used was taken from the Procedural Examples package [128] published in the Unity Asset Store.
Terrain sketches are used similar as in terrain generation, although no weighting is used as it is only one texture. They are simply drawn onto the existing texture with a set width to create accessible paths.

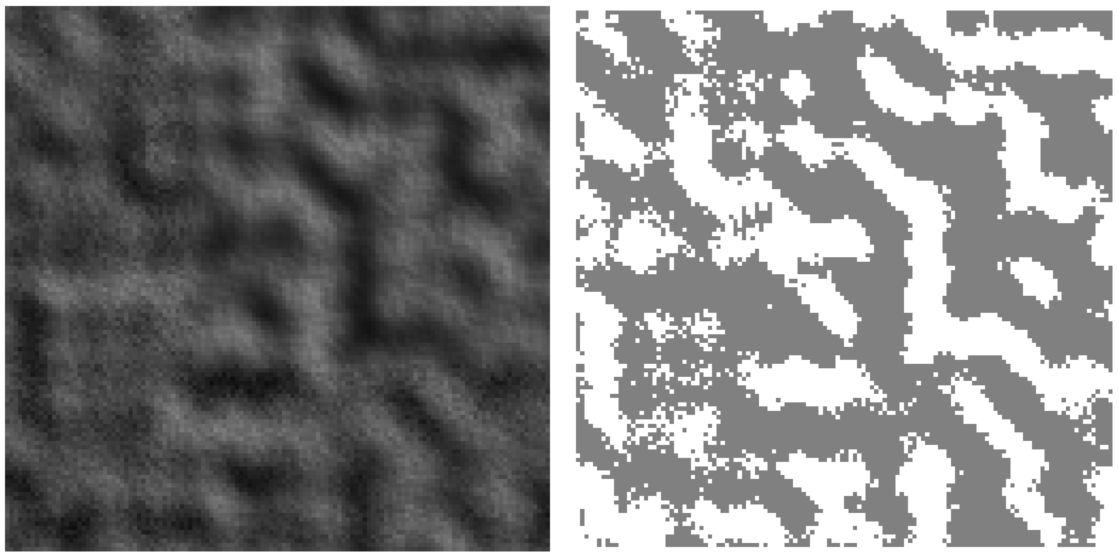

(a) raw noise for caves (b) noise after cave

Figure 5.4.: cave generation from noise

### 5.1.4. Dungeon

The dungeon generation uses a completely different concept than the previous two methods. The commercial game TinyKeep [129] uses an approach based on nodes and triangulation, as explained by the developer Phi Dinh [130]. Based on that, a similar but simpler approach was developed.

The basic premise is still the same: rooms are generated and used as nodes in a graph, which represents how the rooms are connected. Every room is a rectangle, described through its lower left and upper right point. Additionally, the center point is calculated and stored as it is used for graph calculation. With this approach, the usage of a terrain sketch as used in the other level types is not necessary, as the graph structure already ensures that all points of interest are reachable.

For the graph, numerous types are looked at: a minimum spanning tree is a well understood graph, but it generates a lot of dead ends, which is not desired in this case. The favorites were a Gabriel graph [131] and relative neighborhood graph [132]. They both generate structures that resemble human-made structures. The relative neighborhood graph was ultimately chosen as it is a lot less complex to implement,

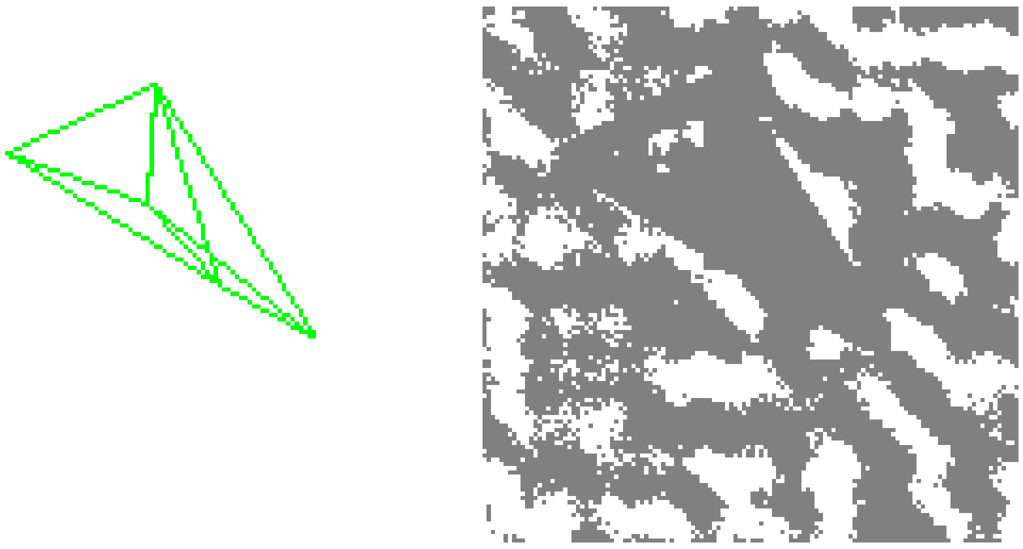

(a) cave sketch (b) cave with sketch

Figure 5.5.: cave with sketch

although the naive implementation is of $O(n^3)$ complexity. But since the current maximum number of rooms is only set to 10, it is not of concern, but should be noted in the future if larger structures are to be generated. An algorithm with $O(n^2)$ complexity using a Voronoi diagram and Delaunay triangulation is described in the paper.
The relative neighborhood graph was developed to extract a perceptually meaningful structure from a set of points, similar to how a human would group points. In this type of graph, two nodes are connected if there are no other nodes that are closer to either of the two than they are to each other.
At the end of the generation process, the rooms are drawn onto a texture to use the same mesh generator used in cave generation to create the actual geometry. It suffers from the same downside of using a 2.5D terrain and seemingly angular walls.

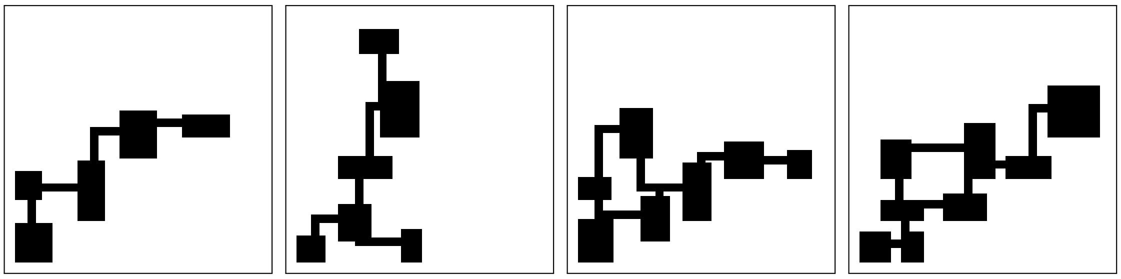


Figure 5.6.: Examples for generated dungeons

## 5.2. Adaptivity

Difficulty settings are implemented and affect the number of missions and objectives generated. They have to be set manually, as they are currently only implemented with future developments in mind. They can be changed during the game, but only affect it before a level is generated. Missions in the landscape are only generated at the start of the game and are not affected by changes during the game.

## 5.3. Game Concept

The game follows a rather simple concept, as the focus of this thesis is the procedural generation in general, the game is only used as a proof of concept. Initially, a larger concept involving a small role-playing game was planned, but not implemented due to time constraints.
The starting point of the game is the landscape, were markers are placed to transition to caves and/or dungeons, which are randomly selected and placed. Inside a dungeon or cave, a submission is generated and in its current state, points of interest are generated according to the missions and placed on the map randomly on the map and are shown by markers. To have a minimal amount of actual gameplay, players are required to run into a marker to fulfill the objective.

# 6 Development of a PCG Module in Unity

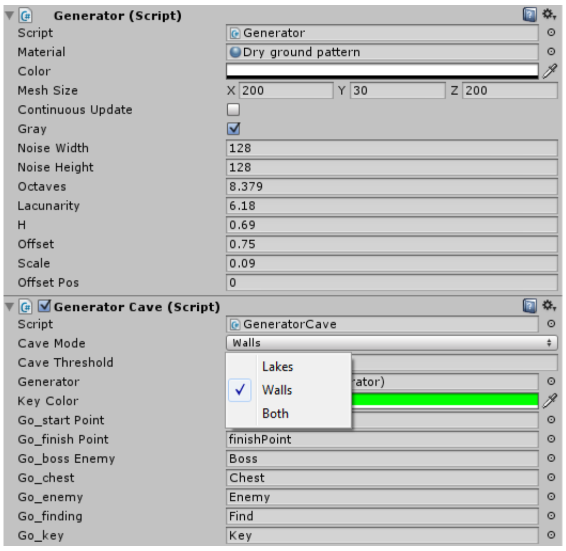
This chapter gives a brief overview of the Unity Engine and details of the implementation and architecture used in the library.
Note that all development was done with version 4.5 and 4.6 of Unity. A switch to version 5.0, which was released while developing, was neglected as the new features were not considered to be useful for this thesis. Unity has separate 2D and 3D development modes, which are not further used as only the 3D mode had the capabilities required.

## 6.1. Unity Engine

Unity comes with a fully graphical development environment. For developing code, the MonoDevelop IDE is bundled with the Unity SDK, an optional extension to use Visual Studio is available through a third party developer [133].

Every game consists of scenes (which can be seen as a equivalent of levels in general; in this thesis, both terms are used synonymously), which contain all objects used. The main building block of every scene is the `GameObject`, which is a container to hold so called *components*, which provide functionality. Some examples are cameras, controllers used by the player or any type of geometry.
Everything else used in scenes is called an asset, whether it be code, textures, sounds, models or anything else. They are stored in a folder of the same name. It is recommended to use a structure of sub-folders to organize all assets. Settings set in the development UI are stored in a separate folder. These folders are the only ones that have to be saved, as everything else is generated by Unity when loading a project.
Every piece of code is called a script and can be written in UnityScript (a JavaScript dialect), C# or Boo (a language heavily inspired by Python). All code is compiled into byte code to be run in the Common Language Runtime of the Mono platform [134], an open source implementation of Microsofts dotNet platform. Development is not limited to one language, all three can be mixed, with some limitations. Communication amongst scripts can be done by calling publicly declared member functions or a simple messaging system. The latter is only recommended with a limited amount of `GameObjects`, as performance can be very slow. Alternatively, the dotNET event-system can be used, but requires more administrative overhead, which is only worth it when there are a large number of `GameObjects` in the given scene. For this thesis, scripts were only written in C#, which was a personal choice of the author.
One particular feature of Unity is that member variables, which are declared public are shown in the graphical UI and can be monitored and changed at runtime, which severely reduces time to debug. This includes `enum`-values, which are displayed as drop-down menus. An example is shown in figure 6.1.

Unity provides a huge number of classes and interfaces available to the developer, which cover every basic aspect needed in game development, such as frequently used mathematical functions as well as interfaces to all components used in Unity, which are a great help with procedural content generation. Documentation is good, but clearly lacking in some places, for example the description of `Terrain` and `TerrainData`



**Generator (Script)**

| | |
|---|---|
| Script | Generator |
| Material | Dry ground pattern |
| Color | ⬜ |
| Mesh Size | X 200 Y 30 Z 200 |
| Continuous Update | ☐ |
| Gray | ✓ |
| Noise Width | 128 |
| Noise Height | 128 |
| Octaves | 8.379 |
| Lacunarity | 6.18 |
| H | 0.69 |
| Offset | 0.75 |
| Scale | 0.09 |
| Offset Pos | 0 |

**Generator Cave (Script)**

| | |
|---|---|
| Script | GeneratorCave |
| Cave Mode | Walls |
| Cave Threshold | |
| Generator | ator) |
| Key Color | 🟩 |
| Go_start Point | |
| Go_finish Point | finishPoint |
| Go_boss Enemy | Boss |
| Go_chest | Chest |
| Go_enemy | Enemy |
| Go_finding | Find |
| Go_key | Key |

Figure 6.1.: Display of public declared variables in Unity UI

functions is merely more than "use the function name in a sentence". But with the widespread use, other resources provide more information.
A major concept of developing games with Unity are *prefabs*. Every `GameObject` can be used as a prefab by simply dragging and dropping it into the assets view, all settings are saved into the prefab. They can be used in other scenes and instantiated at runtime, which makes them especially useful for functionality that is used regularly.

## 6.2. Procedural Content Generation Package

This section describes the Unity package developed in this thesis which can be used by future projects and contains everything needed to procedurally generate levels.

### 6.2.1. General

Unity provides functionality to export assets from a project into a package, combining the selected item in a container, which can be imported into Unity, preserving the defined structure. In general, Unity does not force a directory structure onto the developer, although it is advised to use one to organize the files. For script files this means that

they are accessible (hence the access modifiers are set accordingly), no matter in which (sub)directory they are located.
Most member variables in the created scripts are declared private and are accessible through properties, which provide getter- and setter functionality. Functions are declared private unless they are used from other classes, in that case they are also declared static if they do not need information from an instantiated object.

### 6.2.2. Architecture

The package uses an object-oriented architecture. The general generation class `Generator` provides basic functionality used for every level. Separate classes provide functionality specific for the set level, but use the `Generator` class through an instantiated object. When useful, separate classes are created for additional functionality.

### 6.2.3. Integration in Unity Architecture

Newly created scripts through the Unity development UI have a standard structure: they derive from the `MonoBehaviour` class and have callback function stubs for `Start()` and `Update()` functions. `Start()` is called upon loading of the scene, while `Update()` is called when a new frame is rendered. Additional useful callback functions are `Awake()` (similar to `Start()` but called earlier in the loading process, useful for initialization) and `OnGUI()` (renders the GUI elements).
Unity does not require any kind of architecture for the code, but it is recommended to have a central `GameObject` handling all functionality. In this case, the `GameManager` class handles functionality and initializes the basic functionality.
For every script that is attached to a `GameObject`, the public variables are shown in the development UI. For basic data types such as numbers or strings, default values can be set in the code, which are inherited in the UI. More specific data types such as textures or other objects can be set through the UI, otherwise they are set to `null`.

### 6.2.4. Content Generation Algorithms

This section briefly describes implementation details on the algorithms and data structures used in this thesis. For a general description, see section 5.1.
The previously described markers are saved as prefabs and are instantiated at runtime. To do so they have to be specified in the code, according member variables are available in all classes and can be identified by the `go_` prefix, which stands for `GameObject`, the data type of the variable.

#### Missions

Missions are organized in a graph-structure, with each graph object having a number of `MissionNode` objects organized in a list, which makes it easier to traverse all nodes. Each nodes also has a list with references to the following nodes, which makes for a directed graph, which is sufficient in this case. Each node also has a defined task, stored as a `String` and coordinates defining the point as two `Integer` values representing the point on the noise texture and a boolean variable storing if an objective is completed or not.
The definition of the grammar and production is handled by two classes, `LSystem` and `LSystemProductionRule`. Constants and variables are stored in dictionaries of type `<string, string>`. Production is handled by the `ApplyRule` function, which internally uses the `string.Replace` function of C#.
The constructor of the according class, which defines the constants, variables and rules for the grammar is shown for reference in tables A.16 and A.17.

#### Landscape

The main class to generate a landscape is the `TerrainGenerator` class. It uses an instantiated Perlin noise generator and has a huge number of member variables, which are mostly used to control the generation.
Key functions are `FillHeights()`, which generates the heightmap, and `AssignSplatMap()`,

which generates the splatmaps for the current heightmap. Heightmaps are stored as 2D arrays of `floats` for every tile in a `TerrainData` object. Converting them between `float` arrays and `Texture2D` objects should be kept to a minimum, as the loss of precision can lead to rough landscapes with little stair-like terrain. The heightmaps are also declared as static variables, preventing them from being deleted upon transitioning to a different scene. When the player returns from a the sub level, the landscape is built again with the previously generated heightmaps, because generating those is by far the most complex and calculation intensive task.
Splatmaps currently support four different textures which is a hardcoded number, but could be extended to support a dynamical number of textures . Weights for the texture depends on the height of a specific point compared to the maximum height: the top 10% and 35% respectively use a fixed texture without any mixture, lower levels use a mixture of two textures while the lowest 20% use only one texture again. These values are also hardcoded as determining them provided a big challenge, calculating them dynamically proved to be out of the scope of this thesis.
A new mission is generated the first time the landscape is generated and is stored in the `GameManager` to prevent it from being deleted upon transition to a different level.
The entrances to caves and dungeons are placed in the map through instantiation. The last know position is also saved, so that the player is placed near the entrance after leaving the sub level. The terrain sketch is applied using a key color to distinguish it from noise. For every point on the heightmap, the euclidean distance to the nearest point of the sketch is calculated, determining the weight for applying the sketch. The closer the point is, the higher the weight is for the plains noise.

### Cave

Caves are generated by the `GeneratorCave` class, which derives from `MonoBehaviour`. It uses a `Texture2D` object to store the noise, which has less precision than using a `float` array, but is easier to handle, especially as it can be drawn on without conversion. The `caveThreshold` variable determines when a point in the noise is considered belonging to the floor or the ceiling, which can be set fixed or calculated from the average

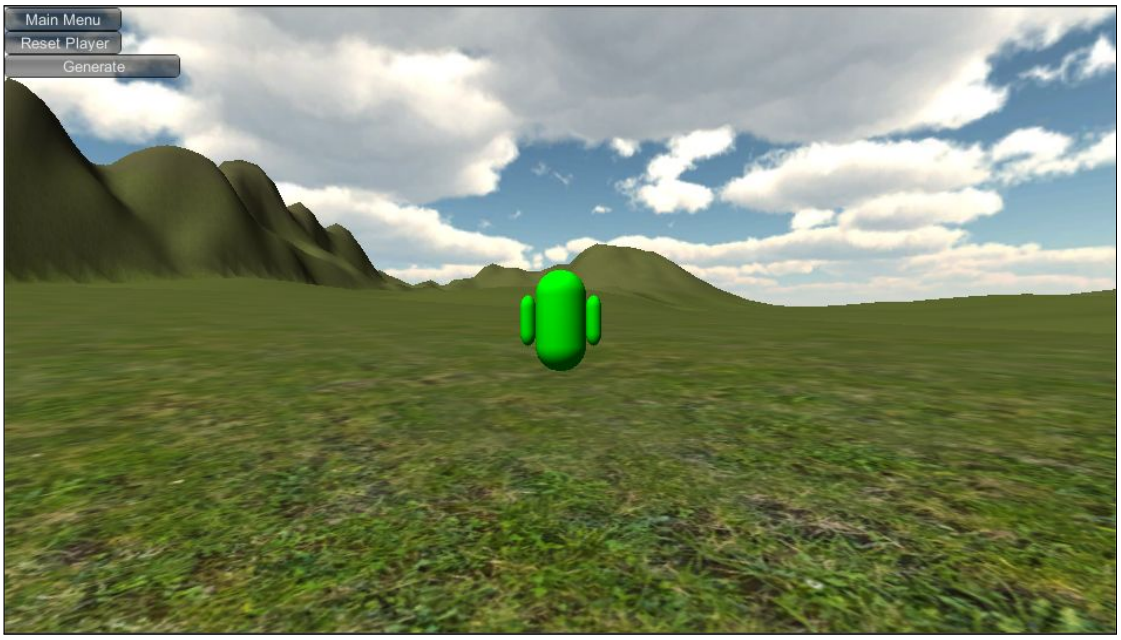


Figure 6.2.: landscape ingame

value.
The terrain sketch is applied to the noise texture by drawing a square of set size onto the noise where the sketch is, determined by the key color in the sketch texture. It uses the mesh generator provided by the `Generator` class to generate geometry. Points of interests of the generated mission are instantiated from prefabs and placed on the geometry.

### Dungeon

Dungeon generation is provided by the `GeneratorDungeon` class. It does not use noise, but draws a number of rectangles onto a `Texture2D` object. But first the rooms are calculated and stored in an more abstract class, making calculations easier as pixel-operations on the `Texture2D` objects are expensive. Rooms are organized in a list containing elements of the type `Room`. During the generation process it is ensured that they stay inside the boundaries of the `Texture2D` object they are to be drawn on and do not overlap. The center point of every room is used as a node in a relative neighborhood graph, which is calculated and determines which rooms are to be connected. This graph

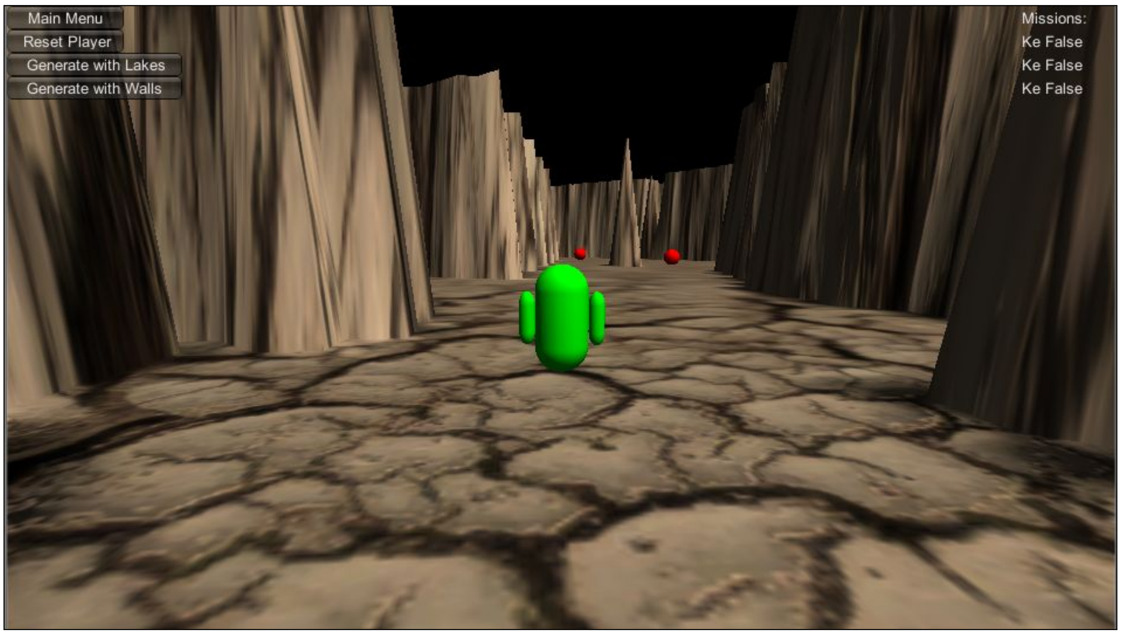


Figure 6.3.: cave ingame

ensures that the corridors are laid planar on the map and can then be simply drawn by calculating a right triangle between the two points and using the line between them as hypotenuse.

### 6.2.5. Game Mechanics

To provide functionality for the proof of concept game, additional classes handle player interaction and controls. Most of the controls for the player character and the camera are adapted from a tutorial by Christian Geiger and Patrick Pogscheba [135]. Additional code handles collisions between the player character and markers, which count as fulfilling an objective.

### 6.2.6. Additional Code

Additional functionality is provided in specific classes, such as Gaussian and median texture filter which can be used to smooth textures. Other classes provide functionality for the main menu and an options menu, the latter is mainly used for experiments with

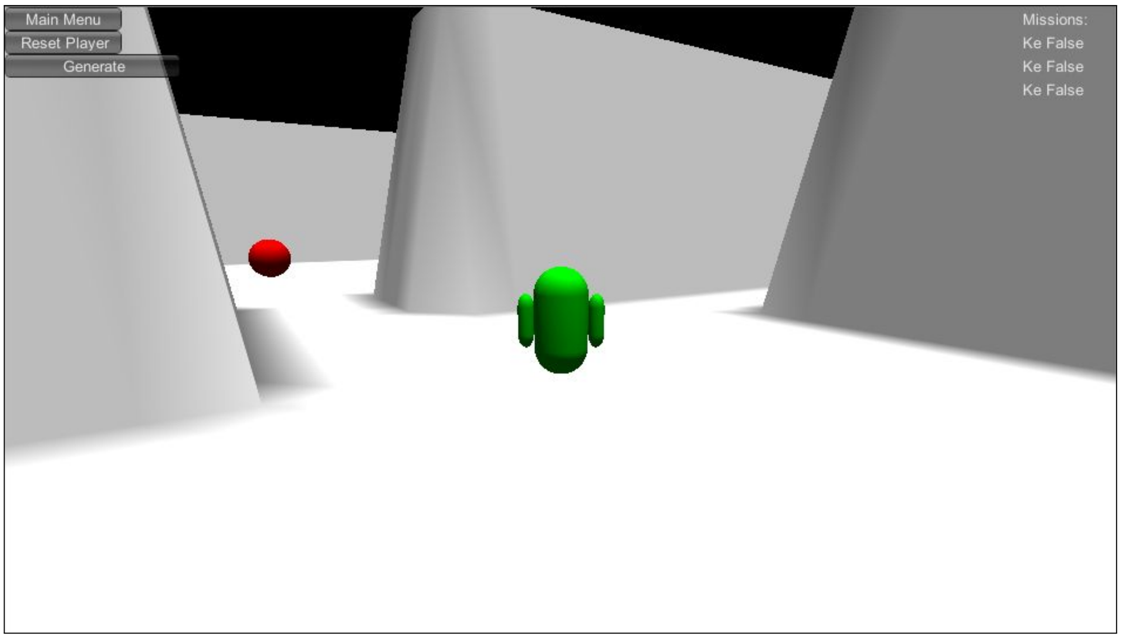


Figure 6.4.: dungeon ingame

the GUI system of Unity introduced in version 4.6.
The GUI seen on the figures 6.2, 6.3 and 6.4 is done in code, but can also be created in the UI, which has the advantage that the interface is visible in the UI and can be easily adjusted, while the interface in code is only visible when the game runs. An example used here is a reset button that puts the player in a safe place high above the level geometry in case it gets stuck (since there is no falling damage calculated it is perfectly safe).

# 7 Conclusion

To conclude this thesis a short summary on the results of the particular reviews will be stated and a final conclusion will be made.
First, a suitable game engine was selected, with a detailed view on **libgdx**, **Esenthel** and **Unity**. Unity was selected, its biggest advantages being the wide variety of different platforms supported without major changes needed, the possibility to use different languages to write code and the wide acceptance by professional and hobby developers alike, which directly leads to a huge amount of teaching resources available.
A small game was developed to test a library for procedurally generating content, with various techniques being looked at. A terrain generator based on Perlin noise has been used to generate a landscape, which is used as hub level. Two types of sub levels were developed, with caves also based on Perlin noise, while dungeons use a relative neighborhood graph to create a structure that looks like it has been made by humans, while noise functions are more suitable for natural-like environments. To ensure that the

points of interest in the level are reachable, the approach of terrain sketching has been adapted to this use case. The basis for the sketches are generated missions, which are generated by a formal grammar and translated into a graph structure.
The resulting library is suitable for role-playing and action-adventure games without significant changes.

# 8 Limitations and Future Work

This chapter describes the limitations of the current work and possible improvements and extensions for future projects.

## 8.1. Levels

The generated levels were designed with a role-playing aspect in mind and should work without modification in that genre. Due to the perspective used, action-adventure games would also be possible, no matter if played from a first- or third-person perspective. Games played from an isometric or birds eye view are also possible, but might require some modifications as space outside of the players perspective was ignored due to not being visible in the proof of concept game. This includes all geometry that is behind walls generated by the mesh generator used in section 5.1.3 and 5.1.4.

For strategy games the developed levels should also be usable, for multiplayer games they might require modification to ensure an even playing field for all players.
The levels generated are not useful for platforming games, as they require a vastly different level construction due to the changed perspective and lack of vertical elements. The basic concept of using noise and graphs might be useful, but the process to get finished levels has to be very different.
As the generated structures lack objects to separate parts of the levels from each other, the whole level is accessible from he start. The mission structures are quite simple, but the data structures used for missions are prepared to be used with complex, branching and prerequisite tasks to form a progression model. An example can be seen in the work of Joris Dormans [85] [86]. But the level has to be separated into logical segments accordingly. Multiple missions in a level are also imaginable, with the graph structures already prepared to support them. The objectives are currently randomly placed in a segment defined by the number of objective types, which is a very basic distribution and can lead to clumped up objectives. A different distribution would help with this and also ensure the space is used to properly utilized.
The current level generation does not use all terrain features available from Unity. It supports splitting the landscape into multiple tiles, which is currently not used due to the terrain sketch not supporting it. This could be used to create a much bigger world, which could be expanded at runtime and could take changing difficulty settings into consideration.

## 8.2. Presentation

Generally, the graphical fidelity of the levels is not great, they are mostly empty geometry with basic texturing or shading, aside from random elements in caves produced by the noise. Unity has built-in features to place trees and details objects such as bushes on terrains, but they are currently not used. Also, the calculation of the splatmaps is currently very basic and static. A more sophisticated technique would increase the graphical fidelity.
Caves and dungeons are currently only rudimentary textured if at all. Since the geometry

is not known beforehand, texture alignment of pre-defined textures is tricky. Also, the distribution of the polygons is not even, as the walls only consist of one stripe of triangles. Texture synthesis could be used to solve this. Furthermore, there could be scenery objects placed to increase the realism, for example rocks in caves and all kinds of man made objects in dungeons like tables, chairs and wall paintings. Furthermore, player models and points of interest are currently marked by simple placeholder objects, more suitable ones would increase immersion.

## 8.3. Adaptivity

Currently, the game has three difficulty settings which only affect the structure of the missions generated in caves and dungeons. Missions on the landscape are only generated once which is intended as changing them without a reason would most certainly lead to confusion. They could be extended, depending on the performance of the player or expand the landscape if needed.
Caves and dungeons are generated every time the player enters them and could be adapted to reflect changes in the difficulty settings. Caves are based on Perlin noise and different settings to create more or less complex structures are possible, but have not been tested to an extend to be used in the current state. Dungeons are based on a different structure, therefore other adjustments are available: the simplest one would be to change the number of rooms, which is currently already done as every objective is placed in exactly one room. But it is debatable if more rooms increase the challenge or lead to frustration. A different type of graph could also be used, the currently used random neighborhood graph was not chosen with this criteria in mind.
The difficulty is currently fixed and can only be changed in the development UI of Unity. The difficulty levels are defined internally as an `enum`, making it possible to be changed during the game without the UI. It is planned that they should be changeable by external factors, such as the performance or the physical condition of the player as determined by sensors. As an example, the current stress level of the player could be determined and the difficulty could be decreased to lower the stress put on the player. Another way could be to change the difficulty during a trial by the conductor, based on data he sees

on a console which shows the performance and physical condition of the player. This requires extensive expansion of the current structure.
To increase the capability of the adaptiveness, a detailed model of the players could be used to determine and save the players skill and physical condition. Some examples are given in section 2.3

## 8.4. Interfaces

To support adaptivity described in the previous section, gathering data from sensors measuring bodily functions such as heart rate, blood pressure could be useful to determine the users physical condition. Additionally, it would be useful to adjust parameters of the game from a separate program, which could be operated by the conductor of a study to measure impacts of changing conditions for the players.

# A
# Appendix

## A.1. Cost-Utility Analysis

Table A.1.: considered game engines in cost-utility analysis

| **Name** | **Supported Platforms** | **Language Engine** | **Language other** | **Licence** |
|---|---|---|---|---|
| Blender Game Engine | Windows, Linux, MacOS X | C/C++, Python | C/C++, Python | GPL |
| Cafu Engine | Windows, Linux | C/C++ | C/C++ | GPL |

| | | | | |
|---|---|---|---|---|
| Crystal Space | Windows, Linux, MacOS X | C/C++ | C/C++ | LGPL |
| Esenthel Engine | Windows, Linux, MacOS X, Browser, Android, iOS | C/C++, Java, Obj-C, JavaScript | C/C++, Obj-C, JavaScript | Proprietary, free available |
| Irrlicht | Windows, Linux, MacOS X | C/C++, C#, VB.net | C/C++, C#, VB.net | ZLIB |
| jMonkey | Windows, Linux, MacOS X, Browser, Android | Java | Java, Ruby, Python, Javascript, PHP | BSD |
| libgdx | Windows, Linux, MacOS X, Browser, Android, iOS | Java, C/C++ | Java | Apache 2 |
| Neoaxis 3D Engine | Windows, MacOS X | C/C++, C# | C/C++, C# | Proprietary, free available |
| OGRE | Windows, Linux, MacOS X | C/C++ | C/C++ | MIT |
| Panda3D | Windows, Linux, MacOS X | C/C++, Python | C/C++, Python | BSD |
| ShiVa Engine | Windows, Linux, MacOS X, Browser, Android, iOS | C/C++ | C/C++ | Proprietary, free available |

| | | | | |
|---|---|---|---|---|
| Unity | Windows, Linux, MacOS X, Browser, Android, iOS | C/C++ | C#, JavaScript, Python | Proprietary, free available |
| | | | | |
| C4 Engine | Windows, Linux, MacOS X | C/C++ | C/C++ | Proprietary, cost |
| DX Studio | Windows | C/C++, VB | C/C++, VB | Proprietary, free available |
| Leadwerks Engine | Windows | C/C++, C#, Java, Perl, Python | C/C++, C#, Java, Perl, Python | Proprietary, cost |
| Visual3D Game Engine | Windows, Browser | C# | C/C++. C#, Ruby, Python, F#, Lua | Proprietary, cost |
| | | | | |
| Unreal Development Kit | Windows, Linux, MacOS X, Browser, Android, iOS | C/C++ | C/C++ | Proprietary, cost |

Table A.2.: Criteria for cost-utility analysis

| **Criteria** | **Weight** |
|---|---|
| *licence* | *10* |
| open source | 8 |
| closed source | 2 |
| *technical aspects* | *30* |
| programming language engine | 4 |
| programming language gamecode | 7 |
| supported platforms | 6 |

| | |
|---|---|
| functionality | 5 |
| architecture | 8 |
| *activity of developers* | *20* |
| frequency of updates | 7 |
| time since last update | 7 |
| presence in forums/wikis/chats | 6 |
| *tool support* | *16* |
| integration in IDEs | 7 |
| support for modelling/animation tools | 5 |
| provided tools | 4 |
| *community* | *17* |
| number of projects | 5 |
| activity of projects | 6 |
| activity in forums/wikis/chats | 6 |
| *ease of learing* | *33* |
| learning curve | 9 |
| available literature/tutorials | 9 |
| documentation | 8 |
| support by developers | 7 |
| **Total:** | **135** |

## A.2. Engine Cost-Utility Analysis

| **Blender Game Engine** | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *40* |
| open source | 8 | 5 | 40 |
| closed source | 2 | 0 | 0 |
| *Technical* | *30* | *15* | *96* |
| Programming Language Engine | 4 | 2 | 8 |
| Programming Language Gamecode | 7 | 4 | 28 |
| Supported Platforms | 6 | 3 | 18 |
| Functionality | 5 | 2 | 10 |
| Architecture | 8 | 4 | 32 |
| *Activity Developer* | *20* | *7* | *46* |
| Frequency of updates | 7 | 2 | 14 |
| Time since last update | 7 | 2 | 14 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *Tool Support* | *16* | *11* | *58* |
| Integration in IDEs | 7 | 3 | 21 |
| Support for Modeling/Animation Tools | 5 | 5 | 25 |
| provided tools | 4 | 3 | 12 |
| *Community* | *17* | *5* | *29* |
| Number of Projects | 5 | 1 | 5 |
| Activity of Projects | 6 | 2 | 12 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 2 | 12 |
| *ease of learning* | *33* | *11* | *92* |
| learning curve | 9 | 3 | 27 |
| available literature/tutorial | 9 | 3 | 27 |
| documentation | 8 | 3 | 24 |
| support by developers | 7 | 2 | 14 |
| **Total** | **126** | **54** | **361** |

Table A.3.: cost-utility analysis of Blender Game Engine

| **Cafu Engine** | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *40* |
| open source | 8 | 5 | 40 |
| closed source | 2 | 0 | 0 |
| *Technical* | *30* | *15* | *96* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 4 | 28 |
| Supported Platforms | 6 | 2 | 12 |
| Functionality | 5 | 4 | 20 |
| Architecture | 8 | 3 | 24 |
| *Activity Developer* | *20* | *7* | *46* |
| Frequency of updates | 7 | 3 | 21 |
| Time since last update | 7 | 1 | 7 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *Tool Support* | *16* | *9* | *46* |
| Integration in IDEs | 7 | 2 | 14 |
| Support for Modeling/Animation Tools | 5 | 4 | 20 |
| provided tools | 4 | 3 | 12 |
| *Community* | *17* | *8* | *46* |
| Number of Projects | 5 | 2 | 10 |
| Activity of Projects | 6 | 3 | 18 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *ease of learning* | *33* | *12* | *97* |
| learning curve | 9 | 3 | 27 |
| available literature/tutorial | 9 | 2 | 18 |
| documentation | 8 | 3 | 24 |
| support by developers | 7 | 4 | 28 |
| **Total** | **126** | **57** | **371** |

Table A.4.: cost-utility analysis of Cafu Engine

| **Crystal Space** | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *40* |
| open source | 8 | 5 | 40 |
| closed source | 2 | 0 | 0 |
| *Technical* | *30* | *18* | *110* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 4 | 28 |
| Supported Platforms | 6 | 3 | 18 |
| Functionality | 5 | 4 | 20 |
| Architecture | 8 | 4 | 32 |
| *Activity Developer* | *20* | *7* | *46* |
| Frequency of updates | 7 | 3 | 21 |
| Time since last update | 7 | 1 | 7 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *Tool Support* | *16* | *5* | *27* |
| Integration in IDEs | 7 | 1 | 7 |
| Support for Modeling/Animation Tools | 5 | 4 | 20 |
| provided tools | 4 | 0 | 0 |
| *Community* | *17* | *9* | *51* |
| Number of Projects | 5 | 3 | 15 |
| Activity of Projects | 6 | 3 | 18 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *ease of learning* | *33* | *10* | *81* |
| learning curve | 9 | 1 | 9 |
| available literature/tutorial | 9 | 3 | 27 |
| documentation | 8 | 3 | 24 |
| support by developers | 7 | 3 | 21 |
| **Total** | **126** | **54** | **355** |

Table A.5.: cost-utility analysis of Crystal Space

| **Esenthel Engine** | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *10* |
| open source | 8 | 0 | 0 |
| closed source | 2 | 5 | 10 |
| *Technical* | *30* | *20* | *120* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 3 | 21 |
| Supported Platforms | 6 | 5 | 30 |
| Functionality | 5 | 5 | 25 |
| Architecture | 8 | 4 | 32 |
| *Activity Developer* | *20* | *14* | *94* |
| Frequency of updates | 7 | 5 | 35 |
| Time since last update | 7 | 5 | 35 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 4 | 24 |
| *Tool Support* | *16* | *13* | *69* |
| Integration in IDEs | 7 | 4 | 28 |
| Support for Modeling/Animation Tools | 5 | 5 | 25 |
| provided tools | 4 | 4 | 16 |
| *Community* | *17* | *8* | *46* |
| Number of Projects | 5 | 2 | 10 |
| Activity of Projects | 6 | 3 | 18 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *ease of learning* | *33* | *12* | *98* |
| learning curve | 9 | 2 | 18 |
| available literature/tutorial | 9 | 3 | 27 |
| documentation | 8 | 4 | 32 |
| support by developers | 7 | 3 | 21 |
| **Total** | **126** | **72** | **437** |

Table A.6.: cost-utility analysis of Esenthel Engine

| **Irrlicht Engine** | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *40* |
| open source | 8 | 5 | 40 |
| closed source | 2 | 0 | 0 |
| *Technical* | *30* | *15* | *90* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 3 | 21 |
| Supported Platforms | 6 | 3 | 18 |
| Functionality | 5 | 3 | 15 |
| Architecture | 8 | 3 | 24 |
| *Activity Developer* | *20* | *7* | *46* |
| Frequency of updates | 7 | 2 | 14 |
| Time since last update | 7 | 2 | 14 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *Tool Support* | *16* | *10* | *56* |
| Integration in IDEs | 7 | 4 | 28 |
| Support for Modeling/Animation Tools | 5 | 4 | 20 |
| provided tools | 4 | 2 | 8 |
| *Community* | *17* | *9* | *51* |
| Number of Projects | 5 | 3 | 15 |
| Activity of Projects | 6 | 2 | 12 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 4 | 24 |
| *ease of learning* | *33* | *13* | *107* |
| learning curve | 9 | 2 | 18 |
| available literature/tutorial | 9 | 4 | 36 |
| documentation | 8 | 4 | 32 |
| support by developers | 7 | 3 | 21 |
| **Total** | **126** | **59** | **390** |

Table A.7.: cost-utility analysis of Irrlicht Engine

| jMonkey Engine | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *40* |
| open source | 8 | 5 | 40 |
| closed source | 2 | 0 | 0 |
| *Technical* | *30* | *19* | *115* |
| Programming Language Engine | 4 | 4 | 16 |
| Programming Language Gamecode | 7 | 4 | 28 |
| Supported Platforms | 6 | 4 | 24 |
| Functionality | 5 | 3 | 15 |
| Architecture | 8 | 4 | 32 |
| *Activity Developer* | *20* | *9* | *60* |
| Frequency of updates | 7 | 3 | 21 |
| Time since last update | 7 | 3 | 21 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *Tool Support* | *16* | *7* | *39* |
| Integration in IDEs | 7 | 3 | 21 |
| Support for Modeling/Animation Tools | 5 | 2 | 10 |
| provided tools | 4 | 2 | 8 |
| *Community* | *17* | *9* | *52* |
| Number of Projects | 5 | 2 | 10 |
| Activity of Projects | 6 | 3 | 18 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 4 | 24 |
| *ease of learning* | *33* | *14* | *116* |
| learning curve | 9 | 3 | 27 |
| available literature/tutorial | 9 | 4 | 36 |
| documentation | 8 | 4 | 32 |
| support by developers | 7 | 3 | 21 |
| **Total** | **126** | **63** | **422** |

Table A.8.: cost-utility analysis of jMonkey Engine

| libgdx | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *40* |
| open source | 8 | 5 | 40 |
| closed source | 2 | 0 | 0 |
| *Technical* | *30* | *19* | *113* |
| Programming Language Engine | 4 | 4 | 16 |
| Programming Language Gamecode | 7 | 4 | 28 |
| Supported Platforms | 6 | 5 | 30 |
| Functionality | 5 | 3 | 15 |
| Architecture | 8 | 3 | 24 |
| *Activity Developer* | *20* | *14* | *94* |
| Frequency of updates | 7 | 5 | 35 |
| Time since last update | 7 | 5 | 35 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 4 | 24 |
| *Tool Support* | *16* | *6* | *37* |
| Integration in IDEs | 7 | 4 | 28 |
| Support for Modeling/Animation Tools | 5 | 1 | 5 |
| provided tools | 4 | 1 | 4 |
| *Community* | *17* | *8* | *46* |
| Number of Projects | 5 | 2 | 10 |
| Activity of Projects | 6 | 3 | 18 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *ease of learning* | *33* | *12* | *99* |
| learning curve | 9 | 3 | 27 |
| available literature/tutorial | 9 | 3 | 27 |
| documentation | 8 | 3 | 24 |
| support by developers | 7 | 3 | 21 |
| **Total** | **126** | **64** | **429** |

Table A.9.: cost-utility analysis of libgdx

| **NeoAxis Engine** | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *10* |
| open source | 8 | 0 | 0 |
| closed source | 2 | 5 | 10 |
| *Technical* | *30* | *17* | *101* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 4 | 28 |
| Supported Platforms | 6 | 2 | 12 |
| Functionality | 5 | 5 | 25 |
| Architecture | 8 | 3 | 24 |
| *Activity Developer* | *20* | *12* | *80* |
| Frequency of updates | 7 | 4 | 28 |
| Time since last update | 7 | 4 | 28 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 4 | 24 |
| *Tool Support* | *16* | *12* | *61* |
| Integration in IDEs | 7 | 3 | 21 |
| Support for Modeling/Animation Tools | 5 | 4 | 20 |
| provided tools | 4 | 5 | 20 |
| *Community* | *17* | *9* | *52* |
| Number of Projects | 5 | 2 | 10 |
| Activity of Projects | 6 | 3 | 18 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 4 | 24 |
| *ease of learning* | *33* | *11* | *89* |
| learning curve | 9 | 3 | 27 |
| available literature/tutorial | 9 | 2 | 18 |
| documentation | 8 | 2 | 16 |
| support by developers | 7 | 4 | 28 |
| **Total** | **126** | **66** | **393** |

Table A.10.: cost-utility analysis of NeoAxis Engine

| **OGRE Engine** | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *40* |
| open source | 8 | 5 | 40 |
| closed source | 2 | 0 | 0 |
| *Technical* | *30* | *14* | *85* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 3 | 21 |
| Supported Platforms | 6 | 3 | 18 |
| Functionality | 5 | 2 | 10 |
| Architecture | 8 | 3 | 24 |
| *Activity Developer* | *20* | *11* | *73* |
| Frequency of updates | 7 | 4 | 28 |
| Time since last update | 7 | 3 | 21 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 4 | 24 |
| *Tool Support* | *16* | *4* | *24* |
| Integration in IDEs | 7 | 2 | 24 |
| Support for Modeling/Animation Tools | 5 | 2 | 10 |
| provided tools | 4 | 0 | 0 |
| *Community* | *17* | *14* | *28* |
| Number of Projects | 5 | 4 | 20 |
| Activity of Projects | 6 | 5 | 30 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 5 | 30 |
| *ease of learning* | *33* | *15* | *123* |
| learning curve | 9 | 2 | 18 |
| available literature/tutorial | 9 | 5 | 45 |
| documentation | 8 | 4 | 32 |
| support by developers | 7 | 4 | 28 |
| **Total** | **126** | **63** | **425** |

Table A.11.: cost-utility analysis of OGRE Engine

| **Panda3D Engine** | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *40* |
| open source | 8 | 5 | 40 |
| closed source | 2 | 0 | 0 |
| *Technical* | *30* | *14* | *85* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 4 | 28 |
| Supported Platforms | 6 | 3 | 18 |
| Functionality | 5 | 3 | 15 |
| Architecture | 8 | 3 | 24 |
| *Activity Developer* | *20* | *6* | *39* |
| Frequency of updates | 7 | 1 | 7 |
| Time since last update | 7 | 2 | 14 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *Tool Support* | *16* | *6* | *31* |
| Integration in IDEs | 7 | 1 | 7 |
| Support for Modeling/Animation Tools | 5 | 3 | 20 |
| provided tools | 4 | 1 | 4 |
| *Community* | *17* | *9* | *52* |
| Number of Projects | 5 | 2 | 10 |
| Activity of Projects | 6 | 3 | 18 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 4 | 24 |
| *ease of learning* | *33* | *13* | *107* |
| learning curve | 9 | 3 | 27 |
| available literature/tutorial | 9 | 3 | 27 |
| documentation | 8 | 4 | 32 |
| support by developers | 7 | 3 | 21 |
| **Total** | **126** | **55** | **366** |

Table A.12.: cost-utility analysis of Panda3D Engine

| **Shiva3D Engine** | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *10* |
| open source | 8 | 0 | 0 |
| closed source | 2 | 5 | 10 |
| *Technical* | *30* | *19* | *112* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 3 | 21 |
| Supported Platforms | 6 | 5 | 30 |
| Functionality | 5 | 5 | 30 |
| Architecture | 8 | 3 | 24 |
| *Activity Developer* | *20* | *6* | *39* |
| Frequency of updates | 7 | 2 | 14 |
| Time since last update | 7 | 1 | 7 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *Tool Support* | *16* | *9* | *43* |
| Integration in IDEs | 7 | 1 | 7 |
| Support for Modeling/Animation Tools | 5 | 4 | 20 |
| provided tools | 4 | 4 | 16 |
| *Community* | *17* | *8* | *46* |
| Number of Projects | 5 | 2 | 10 |
| Activity of Projects | 6 | 3 | 18 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *ease of learning* | *33* | *11* | *90* |
| learning curve | 9 | 2 | 27 |
| available literature/tutorial | 9 | 3 | 27 |
| documentation | 8 | 3 | 24 |
| support by developers | 7 | 3 | 21 |
| **Total** | **126** | **58** | **340** |

Table A.13.: cost-utility analysis of Shiva3D Engine

| Unity Engine | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *10* |
| open source | 8 | 0 | 0 |
| closed source | 2 | 5 | 10 |
| *Technical* | *30* | *21* | *127* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 4 | 30 |
| Supported Platforms | 6 | 5 | 30 |
| Functionality | 5 | 5 | 30 |
| Architecture | 8 | 4 | 32 |
| *Activity Developer* | *20* | *13* | *88* |
| Frequency of updates | 7 | 5 | 35 |
| Time since last update | 7 | 5 | 35 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 3 | 18 |
| *Tool Support* | *16* | *9* | *41* |
| Integration in IDEs | 7 | 0 | 0 |
| Support for Modeling/Animation Tools | 5 | 5 | 25 |
| provided tools | 4 | 4 | 16 |
| *Community* | *17* | *15* | *85* |
| Number of Projects | 5 | 5 | 25 |
| Activity of Projects | 6 | 5 | 30 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 5 | 30 |
| ease of learning | *33* | *17* | *142* |
| learning curve | 9 | 4 | 36 |
| available literature/tutorial | 9 | 5 | 45 |
| documentation | 8 | 5 | 40 |
| support by developers | 7 | 3 | 21 |
| **Total** | **126** | **80** | **493** |

Table A.14.: cost-utility analysis of Unity Engine Engine

| Unreal Development Kit | | | |
|---|---|---|---|
| **Criteria** | **Weight** | **degree of fulfillment** | **Score** |
| *licence* | *10* | *5* | *10* |
| open source | 8 | 0 | 0 |
| closed source | 2 | 5 | 10 |
| *Technical* | *30* | *20* | *121* |
| Programming Language Engine | 4 | 3 | 12 |
| Programming Language Gamecode | 7 | 4 | 28 |
| Supported Platforms | 6 | 5 | 30 |
| Functionality | 5 | 5 | 30 |
| Architecture | 8 | 3 | 24 |
| *Activity Developer* | *20* | *14* | *94* |
| Frequency of updates | 7 | 5 | 35 |
| Time since last update | 7 | 5 | 35 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 4 | 24 |
| *Tool Support* | *16* | *12* | *59* |
| Integration in IDEs | 7 | 2 | 14 |
| Support for Modeling/Animation Tools | 5 | 4 | 24 |
| provided tools | 4 | 5 | 20 |
| *Community* | *17* | *14* | *80* |
| Number of Projects | 5 | 4 | 20 |
| Activity of Projects | 6 | 5 | 30 |
| Activity in Forums/Wiki/Chats/etc. | 6 | 5 | 30 |
| ease of learning | 33 | 15 | 124 |
| learning curve | 9 | 2 | 18 |
| available literature/tutorial | 9 | 5 | 45 |
| documentation | 8 | 5 | 40 |
| support by developers | 7 | 3 | 21 |
| **Total** | **126** | **80** | **486** |

Table A.15.: cost-utility analysis of Unreal Development Kit

## A.3. Production Rules for Mission Generation

Table A.16.: Constants and variables defined in the grammar for mission generation

| **Constant (non-terminal symbol)** | **Symbol** |
|---|---|
| Start | $S$ |
| End | $E$ |
| Key | $K$ |
| Chest | $C$ |
| Find | $F$ |
| Enemy | $E$ |
| Overworld[1] Dungeon | $Od$ |
| Overworld[1] Cave | $Oc$ |
| **Variable (terminal symbol)** | **Symbol** |
| Location | $L$ |
| Objective | $O$ |
| Replacer | $R$ |

Table A.17.: Grammar and corresponding code

| **Rule name** | **formal rule** | **code (parameters for `new LSystemProductionRule()`)** |
|---|---|---|
| remove replacer | $R \rightarrow$ | `m_variables["Replacer"], string.Empty` |
| remove location | $L \rightarrow$ | `m_variables["Location"], string.Empty)` |
| FindKeysAndChest | $O \rightarrow RC$ | `m_variables["Objective"], m_variables["Replacer"] + " " + m_constants["Chest"]` |
| keychest1 | $R \rightarrow RK$ | `m_variables["Replacer"], m_variables["Replacer"] + " " + m_constants["Key"]` |

[1]old name for landscape

| | | |
|---|---|---|
| keychest2 | $R \rightarrow K$ | `m_variables["Replacer"],` `m_constants["Key"]` |
| FindStuff | $O \rightarrow R$ | `m_variables["Objective"],` `m_variables["Replacer"]` |
| find1 | $R \rightarrow RF$ | `m_variables["Replacer"],` `m_variables["Replacer"] + " " +` `m_constants["Find"]` |
| find2 | $R \rightarrow F$ | `m_variables["Replacer"],` `m_constants["Find"]` |
| Enemy | $O \rightarrow E$ | `m_variables["Objective"],` `m_variables["Replacer"]` |
| kill1 | $R \rightarrow RE$ | `m_variables["Replacer"],` `m_variables["Replacer"] + " " +` `m_constants["Enemy"]` |
| kill2 | $R \rightarrow E$ | `m_variables["Replacer"],` `m_constants["Enemy"]` |
| EnterLocation | $L \rightarrow LR$ | `m_variables["Location"],` `m_variables["Location"] + " " +` `m_variables["Replacer"]` |
| enterDungeon | $R \rightarrow Od$ | `m_variables["Replacer"],` `m_constants["Overworld Dungeon"]` |
| enterCave | $R \rightarrow Oc$ | `m_variables["Replacer"],` `m_constants["Overworld Cave"]` |

# List of Figures

# List of Tables

# Listings

# Bibliography

[1] *Mechanical Horse Riding Simulator during WWI*. http://commons.wikimedia.org/wiki/File:Horse_simulator_WWI.jpg,

[2] ABT, C.C.: *Serious Games*. University Press of America, 1987 http://books.google.de/books?id=axUs9HA-hF8C. – ISBN 9780819161482

[3] PARKER, Andrew: *"OAPs say nurse, I need a Wii*. http://www.thesun.co.uk/sol/homepage/news/article294579.ece Last accessed: 2015/04/23, September 2007

[4] SPUFFORD, Francis: *Masters of their universe*. http://www.theguardian.com/books/2003/oct/18/features.weekend Last Accessed 2015/04/22, October 2003

[5] BRABHAM, David ; BELL, Ian ; ACORN SOFT (PUBLISHER): *Elite*. September 1984

[6] TOY, Michael ; WICHMAN, Glenn ; ARNOLD, Ken ; LANE, Jon: *Rogue*. 1980

[7] WICHMAN, Glenn R.: *A Brief History of "Rogue"*. http://www.wichman.org/roguehistory.html, 1997

[8] SOFTWORKS, Bethesda: *The Elder Scrolls II: Daggerfall*. August 1996. – Official Website: http://www.elderscrolls.com/daggerfall/ Last Accessed: 2015/04/22

[9] *The Elder Scrolls II: Daggerfall - The Elder Scrolls Wiki*. http://elderscrolls.wikia.com/wiki/The_Elder_Scrolls_II:_Daggerfall,

[10] BLIZZARD ENTERTAINMENT (DEVELOPER/PUBLISHER US) ; UBISOFT (PUBLISHER EU): *Diablo*. December 1996. – Official Website: www.diablo.com Last accessed 2014/04/22

[11] BETHESDA GAME STUDIOS ; 2K GAMES (PUBLISHER): *The Elder Scrolls IV: Oblivion*. March 2006. – Official Website: http://www.elderscrolls.com/oblivion/ Last accessed: 2015/04/22

[12] *Oblivion:Leveling - UESPWiki*. http://www.uesp.net/wiki/Oblivion:Leveling,

[13] MOJANG: *Minecraft*. November 2011. – Official Website: https://minecraft.net/ Last accessed 2015/04/22

[14] PERSSON, Marcus "notch": *Terrain generation, Part 1*. http://notch.tumblr.com/post/3746989361/terrain-generation-part-1 Last accessed 2015/04/22, March 2009

[15] FRONTIER DEVELOPMENTS: *Elite: Dangerous*. December 2014. – Official Website: http://www.elitedangerous.com/ Last accessed: 2015/04/22

[16] HELLO GAMES: *No Man's Sky*. 2015. – Official Website: http://no-mans-sky.com/ Last accesssed: 2015/04/22

[17] HIGGINS, Chris: *No Man's Sky would take 5 billion years to explore*. http://www.wired.co.uk/news/archive/2014-08/18/no-mans-sky-planets, August 2014

[18] *prodlist :: pouët.net*. http://www.pouet.net/prodlist.php?type[]=4k&platform[]=Windows&page=1 Last accessed: 2015/04/23, . – Demos of 4k category on Windows platform

[19] .THEPRODUKKT: *.kkrieger :: farbrausch.com :: your online worshipping resource*. http://www.farbrausch.de/prod.py?which=114 Last accessed: 2015/04/23, April 2004

[20] GIESEN, Fabian: *.kkrieger postmortem*. http://web.archive.org/web/20050216145754/http://game-face.de/article.php3?id_article=132 Archived version of 2015/02/16, last accessed 2015/04/25, Oktober 2004

[21] EVOLUTIONARY GAMES: *Galactic Arms Race*. 2010. – Official Website: http://galacticarmsrace.blogspot.de/ Last accessed: 2015/04/22

[22] HASTINGS, Erin J. ; GUHA, Ratan K. ; STANLEY, Kenneth O.: Evolving content in the galactic arms race video game. In: *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on* IEEE, 2009, S. 241–248

[23] DORMANS, Joris: Generating emergent physics for action-adventure games. In: *Proceedings of the The third workshop on Procedural Content Generation in Games* ACM, 2012, S. 9

[24] FERNÁNDEZ-VARA, Clara ; THOMSON, Alec: Procedural generation of narrative puzzles in adventure games: The puzzle-dice system. In: *Proceedings of the The third workshop on Procedural Content Generation in Games* ACM, 2012, S. 12

[25] *Code wheel - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Code_wheel Last accessed: 2015/04/22,

[26] LUCASFILM GAMES ; LUCASARTS (PUBLISHER): *The Secret of Monkey Island*. October 1990. – Entry in MobyGames database: http://www.mobygames.com/game/secret-of-monkey-island Last accessed: 2015/04/22

[27] DECK13 ; BVH SOFTWARE (PUBLISHER): *Ankh - Herz des Osiris*. October 2006. – Official Website: http://ankh-game.com/

[28] MANDELBROT, Benoit B.: *The fractal geometry of nature*. Bd. 173. Macmillian, 1983

[29] FOURNIER, Alain ; FUSSELL, Don ; CARPENTER, Loren: Computer rendering of stochastic models. In: *Communications of the ACM* 25 (1982), Nr. 6, S. 371–384

[30] PERLIN, Ken: An image synthesizer. In: *ACM Siggraph Computer Graphics* 19 (1985), Nr. 3, S. 287–296

[31] PERLIN, Ken: Improving noise. In: *ACM Transactions on Graphics (TOG)* Bd. 21 ACM, 2002, S. 681–682

[32] KELLEY, Alex D. ; MALIN, Michael C. ; NIELSON, Gregory M.: *Terrain simulation using a model of stream erosion*. ACM, 1988

[33] MUSGRAVE, F K. ; KOLB, Craig E. ; MACE, Robert S.: The synthesis and rendering of eroded fractal terrains. In: *ACM SIGGRAPH Computer Graphics* Bd. 23 ACM, 1989, S. 41–50

[34] ZHOU, Howard ; SUN, Jie ; TURK, Greg ; REHG, James M.: Terrain synthesis from digital elevation models. In: *Visualization and Computer Graphics, IEEE Transactions on* 13 (2007), Nr. 4, S. 834–848

[35] ZYDA, Michael: From visual simulation to virtual reality to games. In: *Computer* 38 (2005), Nr. 9, S. 25–32

[36] TATE, Richard ; HARITATOS, Jana ; COLE, Steve: HopeLab's approach to Re-Mission. (2009)

[37] STUTT, Tim: *Why Educational Games Fail*. `http://etcjournal.com/2010/10/18/why-educational-games-fail/`, October 2010

[38] *Marine Doom*. 1996

[39] RIDDELL, Rob: *Doom Goes To War*. `http://archive.wired.com/wired/archive/5.04/ff_doom.html` Last accessed: 2015/04/22, April 1997

[40] CRYTEK: *CRYENGINE 3*. October 2009. – Official Website: `http://www.cryengine.com/` Last accessed: 2015/04/22

[41] GIESELMANN, Hartmut: *US-Armee nutzt CryEngine 3 für Militär-Simulation*. `http://www.heise.de/newsticker/meldung/US-Armee-nutzt-CryEngine-3-fuer-Militaer-Simulation-1252007.html` Last accessed: 2015/04/22, May 2011

[42] UNITED STATES ARMY: *America's Army*. 2002 July. – Official Website: `http://www.americasarmy.com/` Last accessed: 2015/04/22

[43] TURSE, Nick: *Guestdispatch: Zap, zap, you're dead...* http://www.tomdispatch.com/post/1012/, October 2003

[44] PANDEMIC STUDIOS ; MASS MEDIA INC. (PS2) ; THQ (PUBLISHER): *Full Spectrum Warrior*. June 2004. – Entry in MobyGames Database: http://www.mobygames.com/game/full-spectrum-warrior Last accessed: 2015/04/22

[45] ATARI INC.: *Math Grand Prix*. 1982. – Entry in MobyGames database: http://www.mobygames.com/game/atari-2600/math-gran-prix Last accessed: 2015/04/22

[46] MAXIS SOFTWARE INC. ; OCEAN SOFTWARE LTD. (PUBLISHER): *SimEarth*. 1990. – Entry in MobyGames database: http://www.mobygames.com/game/simearth-the-living-planet Last accessed: 2015/04/22

[47] FAKT SOFTWARE: *Crazy Machines*. 2004. – Official Website: http://www.crazy-machines.com/ Last accessed: 2015/04/22

[48] *Preisträger | Deutscher Computerspielpreis*. http://deutscher-computerspielpreis.de/preistraeger Last accessed: 2015/04/22, 2015

[49] *Deutscher Computerspielpreis 2013: Auszeichnung für das missio-Spiel "Menschen auf der Flucht" als bestes Serious Game*. https://www.missio-hilft.de/de/aktion/schutzengel/fuer-familien-in-not-weltweit/missio-truck/2013-04-25-computerspielpreis.html Last accessed: 2015/04/22, April 2013

[50] STOBER, Jens M.: *1378km*. December 2010. – Official Website: http://1378km.de/ Last accessed: 2015/04/22

[51] VALVE CORPORATION ; SIERRA ENTERTAINMENT (PUBLISHER): *Half-Life 2*. November 2004. – Official Website: http://orange.half-life2.com/ Last accessed: 2015/04/24

[52] GRAUPNER, Hardy: *Computer game recreates horrors of former East German border*. http://www.dw.de/computer-game-recreates-horrors-of-former-east-german-border/a-6059839-1, September 2010

[53] REALTIME ASSOCIATES, INC. ; HOPELAB (PUBLISHER): *Re-Mission*. April 2006. – Official Website: http://www.re-mission.net/, Last accessed: 2015/04/22

[54] SHELDON, Josh ; PERRY, Judy ; KLOPFER, Eric ; ONG, Jennifer ; CHEN, Vivian Hsueh-Hua ; TZUO, Pei W. ; ROSENHECK, Louisa: Weatherlings: a new approach to student learning using web-based mobile games. In: *Proceedings of the Fifth International Conference on the Foundations of Digital Games* ACM, 2010, S. 203–208

[55] NIANTIC LABS ; GOOGLE INC. (PUBLISHER): *Ingress*. December 2013. – Official Website: https://www.ingress.com/ Last accessed: 2015/04/22

[56] EPIC GAMES ; DIGITAL EXTREMES ; ATARI SA (PUBLISHER): *Unreal Tournament 2004*. March 2004. – Entry in MobyGames database: http://www.mobygames.com/game/unreal-tournament-2004 Last accessed: 2015/04/22

[57] BLIZZARD ENTERTAINMENT: *World of WarCraft*. November 2004. – Official Website: http://www.warcraft.com Last accessed: 2015/04/22

[58] NATHERA (COMMUNITY MANAGER): *A Raid for All Seasons: Flexible Raid Preview*. http://us.battle.net/wow/en/blog/10175200/a-raid-for-all-seasons-flexible-raid-preview-6-6-2013 Last accessed: 2015/04/22, June 2013

[59] ARENANET, NCsoft (Publisher): *Guild Wars 2*. August 2012. – Official Website: http://www.guildwars2.com/en/ Last accessed: 2015/04/22

[60] *Dynamic level adjustment - Guild Wars 2 Wiki (GW2W)*. https://wiki.guildwars2.com/wiki/Dynamic_level_adjustment, . – Official Wiki

[61] *Level - The Fallout wiki - Fallout: New Vegas and more*. http://fallout.wikia.com/wiki/Level Last accessed: 2015/04/22,

[62] BETHESDA GAME STUDIOS ; BETHESDA SOFTWORKS (PUBLISHER): *Fallout 3*. October 2008. – Official Website: `http://fallout.bethsoft.com/` Last accessed: 2015/04/22

[63] VALVE ; ELECTRONIC ARTS (PUBLISHER RETAIL): *Left 4 Dead*. November 2008. – Official Website: `http://www.l4d.com/blog` Last accessed: 2015/04/22

[64] NEWELL, Gabe: *Gabe Newell Writes for Edge*. `https://web.archive.org/web/20120515225357/http://www.edge-online.com/opinion/gabe-newell-writes-edge` (Archived version of 15th May 2012) Last accessed: 2015/04/22, Decenber 2008

[65] GÖBEL, Stefan ; HARDY, Sandro ; STEINMETZ, Ralf ; CHA, Jongeun ; EL SADDIK, Abdulmotaleb: Serious Games zur Prävention und Rehabilitation Serious Games for Prevention and Rehabilitation. In: *Proceedings of Ambient Assisted Living-AAL-4. Deutscher Kongress: Demographischer Wandel-Assistenzsysteme aus der Forschung in den Markt*, 2011

[66] DOIGNON, Jean-Paul ; FALMAGNE, Jean-Claude: Spaces for the assessment of knowledge. In: *International journal of man-machine studies* 23 (1985), Nr. 2, S. 175–196

[67] GÖBEL, Stefan ; WENDEL, Viktor ; RITTER, Christopher ; STEINMETZ, Ralf: Personalized, adaptive digital educational games using narrative game-based learning objects. In: *Entertainment for Education. Digital Techniques and Systems*. Springer, 2010, S. 438–445

[68] MILLER, Scott: *Auto-dynamic difficulty*. `http://dukenukem.typepad.com/game_matters/2004/01/autoadjusting_g.html`, January 2004

[69] REMEDY ENTERTAINMENT ; GATHERING OF DEVELOPERS (PUBLISHER PC) ; ROCKSTAR GAMES (PUBLISHER CONSOLES AND MOBILE): *Max Payne*. July 2001. – Official Website: `http://www.rockstargames.com/maxpayne/main.html` Last accessed: 2015/04/21

[70] CHARLES, Darryl ; BLACK, Michaela: Dynamic player modeling: A framework for player-centered digital games. In: *Proc. of the International Conference on*

*Computer Games: Artificial Intelligence, Design and Education*, 2004, S. 29–35

[71] GOLDSMITH, Thomas T. J. ; RAY MANN, Estle: *cathode ray tube amusement device*. 1974. – Never sold or marketed to the public

[72]

[73] HIGINBOTHAM, William: *Tennis for Two*. October 1958. – Description of the game: `http://www.bnl.gov/about/history/firstvideo.php` Last accessed: 2015/04/23

[74] KENT, Steven: *The Ultimate History of Video Games: from Pong to Pokemon and beyond... the story behind the craze that touched our li ves and changed the world*. Three Rivers Press, 2010

[75] MARINO-NACHISON, David: *Ralph H. Baer, a father of video gaming, dies at 92*. `http://www.washingtonpost.com/national/health-science/ralph-h-baer-a-father-of-video-gaming-dies-at-92/2014/12/07/a24c8964-7e6e-11e4-8882-03cf08410beb_story.html` Last accessed: 2015/04/23, December 2014

[76] DREADBIT INC.: *Ironcast*. , March 2015. – Official Website: `http://www.dreadbit.com/Ironcast` Last accesesd: 2015/04/24

[77] CHAPPLE, Craig: *How procedural generation is being used to develop exciting new games | Analysis | Develo*. `http://www.develop-online.net/analysis/how-procedural-generation-is-being-used-to-develop-exciting-new-game` `0195646` Last Accessed: 2015/04/26, July 2014. – Section "cutting costs"

[78] *MegaTexture - Wikipedia, the free encyclopedia*. `http://en.wikipedia.org/wiki/MegaTexture`,

[79] *Gradient - Wikipedia, the free encyclopedia*. `http://en.wikipedia.org/wiki/Gradient` Last accessed: 2015/04/26,

[80] PERLIN, Ken: *Ken's Academy Award*. `http://mrl.nyu.edu/~perlin/doc/oscar.html#noise` Last accessed: 2014/04/23,

[81] PERLIN, Ken: *Simplex Noise - Noise Hardware*

[82] BURGER, Wilhelm: Gradientenbasierte Rauschfunktionen und Perlin Noise / School of Informatics, Communications and Media, Upper Austria University of Applied Sciences. Version: November 2008. http://staff.fh-hagenberg.at/burger/. Hagenberg, Austria, November 2008 (HGBTR08-02). – Forschungsbericht

[83] CHOMSKY, Noam: Three models for the description of language. In: *Information Theory, IRE Transactions on* 2 (1956), Nr. 3, S. 113–124

[84] CHOMSKY, Noam: *Syntactic structures*. Walter de Gruyter, 2002

[85] DORMANS, Joris: Adventures in level design: generating missions and spaces for action adventure games. In: *Proceedings of the 2010 workshop on procedural content generation in games* ACM, 2010, S. 1

[86] DORMANS, Joris: Level design as model transformation: a strategy for automated content generation. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* ACM, 2011, S. 2

[87] LINDENMAYER, Aristid: Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. In: *Journal of theoretical biology* 18 (1968), Nr. 3, S. 280–299

[88] INTERACTIVE DATA VISUALIZATION, INC. (IDV): *SpeedTree Animated Trees & Plants Modeling & Render Software*. http://www.speedtree.com/ Last accessed 2014/04/23,

[89] LEHRE, Per K.: Time-complexity Analysis of Evolutionary Algorithms. (2011)

[90] HASTINGS, Erin J. ; STANLEY, Kenneth O.: Interactive genetic engineering of evolved video game content. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* ACM, 2010, S. 8

[91] KIEFF HTTP://EN.WIKIPEDIA.ORG/WIKI/USER:KIEFF: *File:Gospers glider gun.gif*. http://commons.wikimedia.org/wiki/File:Gospers_glider_gun.gif Last accessed: 2015/04/25, 2005

[92] GARDNER, Martin: Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". In: *Scientific American* 223 (1970), Nr. 4, S. 120–123

[93] JOHNSON, Lawrence ; YANNAKAKIS, Georgios N. ; TOGELIUS, Julian: Cellular automata for real-time generation of infinite cave levels. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* ACM, 2010, S. 10

[94] TOMASSINI, Marco ; SIPPER, Moshe ; PERRENOUD, Mathieu: On the generation of high-quality random numbers by two-dimensional cellular automata. In: *Computers, IEEE Transactions on* 49 (2000), Nr. 10, S. 1146–1151

[95] CHOWDHURY, D R. ; BASU, Saugata ; GUPTA, I S. ; CHAUDHURI, P P.: Design of CAECC-Cellular automata based error correcting code. In: *IEEE Transactions on Computers* 43 (1994), Nr. 6, S. 759–764

[96] OLSEN, Jacob: Realtime procedural terrain generation. (2004)

[97] ODDLABS: *Tribal Trouble*. `http://oddlabs.com/tribaltrouble/` Last accessed: 2014/04/23, 2005

[98] ODDLABS: *sunenielsen/tribaltrouble · GitHub*. `https://github.com/sunenielsen/tribaltrouble` Last accessed: 2015/04/23, 2014

[99] NEIDHOLD, Benjamin ; WACKER, Markus ; DEUSSEN, Oliver: Interactive physically based fluid and erosion simulation. (2005)

[100] *The Elder Scrolls II: Daggerfall (Game) - Giant Bomb*. `http://www.giantbomb.com/the-elder-scrolls-ii-daggerfall/3030-1129/` Last accessed: 2015/04/23, June 2014. – Giant Bomb Wiki

[101] KUSHNER, David: *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture*. Random House, 2003

[102] *QuakeC - Wikipedia, the free encyclopedia*. `http://en.wikipedia.org/wiki/QuakeC`,

[103] GESTALT: *The Engine Licensing Game -The ups and downs of licensing a game engine*. http://www.eurogamer.net/articles/engines Last accessed: 2015/04/25, June 2000

[104] GOODKIND, Nicole: *How the video game industry became bigger than movies and music*. http://finance.yahoo.com/blogs/daily-ticker/how-the-video-game-industry-became-bigger-than-movies-and-music-171225174.html Last accessed: 2015/04/23, June 2014

[105] EPIC GAMES: *Unreal Engine*. 1998. – Official Website: https://www.unrealengine.com/ Last accessed: 2015/04/23

[106] PETERSON, Steve: *Next-gen consoles mean increased development costs*. http://www.gamesindustry.biz/articles/2012-04-03-next-gen-consoles-mean-increased-development-costs Last accessed: 2015/04/26, April 2012

[107] HAXE FOUNDATION: *Haxe*. 2005. – Official Website: http://www.haxe.org

[108] ASCII CORPORATION ; ENTERBRAIN ; AGETEC ; DEGICA CO., LTD.: *RPGMaker*. 1988. – Official Website:http://www.rpgmakerweb.com/

[109] UNITY TECHNOLOGIES: *Unity*. June 2008. – Official Website: http://unity3d.com/ Last accessed: 2015/04/23

[110] *List of game engines - Wikipedia, the free encyclopedia*. https://en.wikipedia.org/wiki/List_of_game_engines. – Last accessed: 2014/10/25

[111] *DevDB - Database of Game Development Resources | DevMaster*. http://devmaster.net/devdb/. – Last accessed: 2014/10/25

[112] *DevMaster - game development news, discussions, and resources*. http://www.http://devmaster.net/. – Last accessed: 2014/10/25

[113] DAVIS, Ray: *Unreal Engine 4 Goes Free for Academic Use*. https://www.unrealengine.com/blog/

unreal-engine-4-goes-free-for-academic-use. Version: 2014. – Last accessed: 2014/10/25

[114] BLENDER FOUNDATION: *Blender Game Engine*. October 2013. – Official Website: http://www.blender.org Last accessed: 2015/04/23

[115] ROOSENDAAL, Ton: *Blender roadmap - 2.7, 2.8 and beyond | Blender Code*. http://code.blender.org/index.php/2013/06/blender-roadmap-2-7-2-8-and-beyond/. – Last accessed: 2014/10/28

[116] EPIC GAMES: *Unreal Engine 4 Commercial Game Deployment Guidelines*. https://www.unrealengine.com/release Last accessed: 2015/04/26,

[117] DOCKTER, Hans ; MURDOCH, Adam ; FABER, Szczepan ; NIEDERWIESER, Peter ; DALEY, Luke ; GRÖSCHKE, Rene ; DEBOER, Daz ; APPLING, Steve: *Gradle*. February 2015. – Official Website: http://www.gradle.org/ Last accessed: 2015/04/23

[118] *Esenthel Engine - Next-Gen Game Engine for Windows, Mac, Linux, Android, iOS and Web*. http://esenthel.com/?id=feature_list Last accessed: 2015/04/23, . – Features List

[119] UNITY TECHNOLOGIES: *Unity - Fast Facts*. http://unity3d.com/public-relations Last accessed: 2015/04/23,

[120] INXILE ENTERTAINMENT: *Wasteland 2*. September 2014. – Official Website: https://wasteland.inxile-entertainment.com/

[121] INXILE ENTERTAINMENT: *Wasteland 2 by inXile entertainment - Kickstarter*. https://www.kickstarter.com/projects/inxile/wasteland-2/description Last accessed: 2015/04/23, March 2013

[122] UNITY TECHNOLOGIES: *Unity - Collaboration - Unity Team License*. http://unity3d.com/unity/collaboration Last accessed: 2015/04/24,

[123] ID SOFTWARE (JOHN CARMACK, JOHN ROMERO, DAVE TAYLOR): *Doom engine (id tech 1)*. December 1993. – Developer Website: http://www.idsoftware.

com/ Last accessed: 2015/04/23. Source Code: https://github.com/id-Software Last accesed: 2015/04/23

[124] SILVERMAN, Ken: *Build Engine*. – Source Code: http://advsys.net/ken/buildsrc/ Last accessed: 2015/04/23

[125] SANGLARD, Fabien: *Doom engine code review*. http://fabiensanglard.net/doomIphone/doomClassicRenderer.php, January 2011

[126] SCRAWK: *Simple procedural terrain in Unity | ScrawkBlog*. http://scrawkblog.com/2013/05/15/simple-procedural-terrain-in-unity/, May 2013

[127] BLOOM, Charles: *Terrain Texture Compositing by Blending in the Frame-Buffer*. http://www.cbloom.com/3d/techdocs/splatting.txt, November 2000

[128] UNITY TECHNOLOGIES: *Procedural Examples*. https://www.assetstore.unity3d.com/en/#!/content/5141 Last accessed: 2015/04/25, November 2012

[129] DINH, Phi: *TinyKeep*. http://store.steampowered.com/app/278620/ Last accessed: 2015/04/23, September 2014. – Official Website: http://tinykeep.com/

[130] DINH, Phi: *Phi talks about Random Dungeon Generation in Unity - YouTube*. https://www.youtube.com/watch?v=XwNXtSFQF8Q, 02 2014

[131] *Gabriel graph - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Gabriel_graph,

[132] TOUSSAINT, Godfried T.: The relative neighbourhood graph of a finite planar set. In: *Pattern recognition* 12 (1980), Nr. 4, S. 261–268

[133] SYNTAXTREE: *Visual Studio Tools for Unity*. http://www.unityvs.co, 2011

[134] XAMARIN: *Mono*. June 2004. – Official Website: http://www.mono-project.com/ Last accessed: 2015/04/23

[135] GEIGER, Christian ; POGSCHEBA, Patrick: Spieltrieb - Unity Tutorial, Teil 1 - Erste Schritte mit der Game Engine. In: *iX Developer Sonderheft Spiele entwickeln* (2015), S. 129–137

# Glossary

**2.5D terrain** Simple form of terrain description, using a 2D map with a value at each X/Y-coordinate (X/Z in some cases) defining the height. Major downside is that multiple levels of terrain and overhangs are not possible. 37, 41, 44, 46

**bot** Computer-controlled characters in an otherwise for multiple human players made game. 12

**graphical user interface** Drawn interface on screen with which the user can interact with an application. Typical elements are text-labels, buttons and drop-down lists. Possible input devices mouse and keyboard or touch-based devices.. 35

**GUI** graphical user interface. 35, 57

**heightmap** Raster image that stores values that specify the elevation of a point in relation to a presumed floor plainss. 8, 19, 38, 41, 42, 44, 53, 54, 85

**IDE** Integrated Development Environment. 28, 32

**massivly multiplayer online role-playing game** description. 12

**MMORPG** massivly multiplayer online role-playing game. 12

**PCG** procedurally generated content. 34

**Perlin noise** Noise function created by Ken Perlin, generates plasma fractals.[30] and [31]. 41, 44, 53