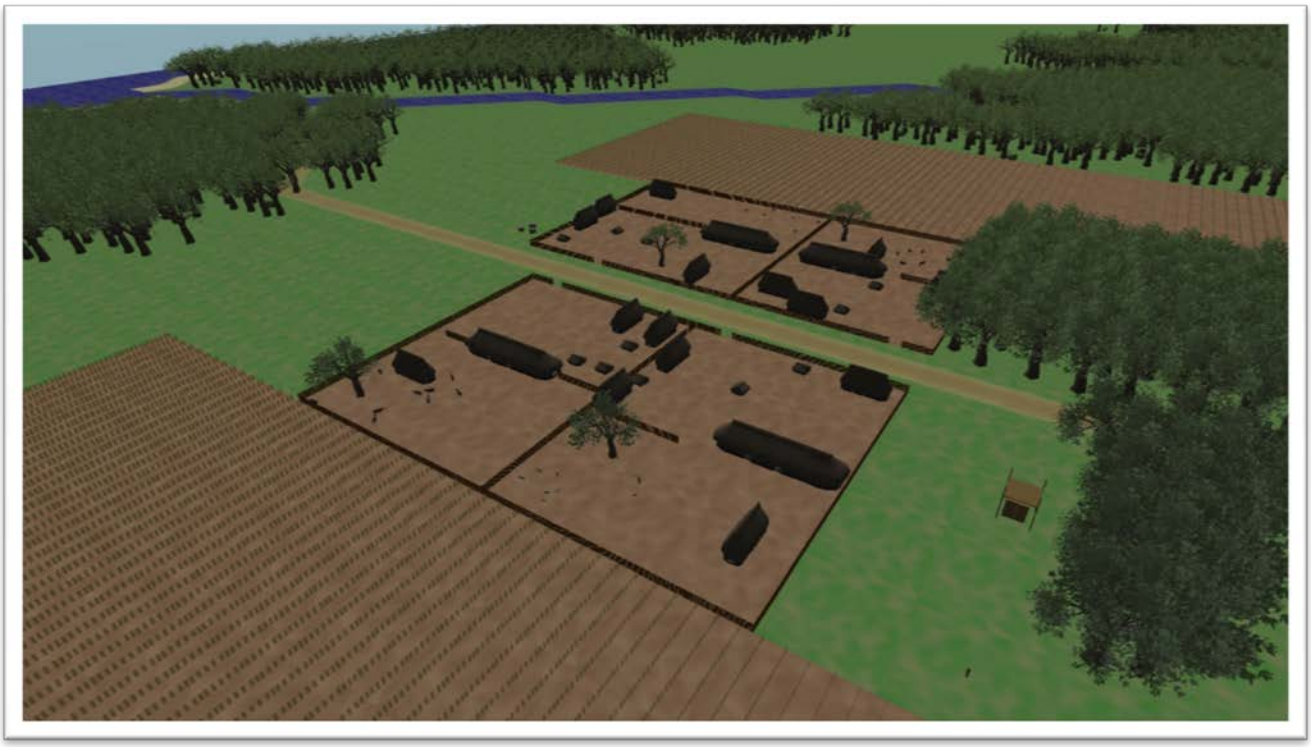
Investigating the Possibility of Procedurally Generated Serious Games

Medialogy Master's Thesis

Jesper Holst
2012



Aalborg University Copenhagen

Semester:
10

Title:
Investigating the Possibility of
Procedurally Generated Serious Games

Project Period:
01-02-12 to 24-05-12

Semester Theme:
Master's Thesis

Aalborg University Copenhagen
Lautrupvang 4B,
2750 Ballerup, Denmark

Semester Coordinator:

Secretary:

Abstract:

This master's thesis investigates the possibility of creating serious games with a procedural generation approach. The areas of serious games and procedural content generation is investigated and used to design and create a prototype serious game. All content in the game is procedurally generated through various approaches. Through a test on 19 subjects the generation process is evaluated to determine if a serious game can be procedurally generated. The learning aspect of the game is secondary but also investigated through the test. It is concluded that the content generation is working and some learning is likely to occur when the game is played.

Supervisor(s):
Emmanuel Blanchard

Project group no.:

Members:

Jesper Holst

Copies: 3
Pages: 63
Finished: 24-05-12

Table of Contents

1 Introduction

1.1 Combining Procedural Content Generation and Serious Games

Since the first video games were developed in the 1960is video games have evolved into many different forms. Today the majority of people is familiar with video games in some form and many use them regularly. This has to do with the large variety of video games and their availability on different technologies such as consoles, phones and tablets. Technology have become a part of our existence and especially the younger generations have a natural approach to the use of video games (Prensky, 2002). It has been argued that video games are a source of violence and generally bad behavior but this is a view that has becoming less common. People within the field of education and learning are arguing that games can have the complete opposite effect, namely that through the elements of playing a video game the player can obtain new knowledge usable in the real world. Normal video games that are developed for the fun and entertainment of their players have the ability to provide some form of learning. This learning could be motor skills gained when using game controllers or improved reading skills. Many games such as the Civilization series are based on historical facts which the player can learn while playing and others can provide a good strategic mindset. Games, as they are generally, can have a lot to offer when it comes to education.

Video games has the ability to capture and hold children's attention for long periods of time, which is something that can be a problem in classroom teaching. The idea of integrating video games directly into education is therefore of great interest. It is believed that video games someday will become an integrated part of education just like other media forms like text, images and video have (Squire, 2007). This requires much research into the area of video games created for an educational purpose.

Video games created for an educational purpose are called serious games (Blanchard et al. 2012). Serious games are often based on constructivist learning methods such as Discovery Learning where the learner obtains knowledge through reflection on obtained information and problem solving. Serious games can be used for teaching in many different areas such as physics, history and math. One important aspect of creating serious games is combining normal game development techniques with learning. Good game play is important to facilitate good learning so the two areas need to be combined. One such combination that, to the authors knowledge, has not been seriously explored, is the combination of procedural content generation and serious games. Procedural content generation has the benefits of greater replayability and the option of creating large varied environments suited for exploring. These benefits might work well in combination with serious games.

One of the first examples of procedural content generation is seen in the game Elite from 1984 where planets and solar systems are created algorithmically to form a detailed universe. Since then procedural content generation has been used commonly within the games industry being used in games such as Diablo, Civilization and The Elder Scrolls. Procedural content generation can be used in many different areas in game development to create content before a game is released or in runtime. Procedural content generation has often been used to create randomized maps in strategy games and levels/dungeons in role playing games. It can be used to create large areas of vegetation of varying detail to take workload off graphics artists in a production. Most aspects of games can be procedurally generated.

1.2 Problem Definition

It is possible to imagine a serious game where content and knowledge elements are procedurally generated and it is this idea that will be explored in the following report. The idea leads to the following problem definition:

"In serious games based on the Exploratory Learning paradigm, how could learning content be procedurally generated while ensuring that semantic congruence is maintained?"

1.3 Approach

The goal of this project is the development of a serious game prototype where the content of the game is procedurally generated. To understand and create serious games, research and theories within the field will be investigated and ultimately used to design a prototype game. The content in the prototype game will be procedurally generated and therefore an understanding of the area of procedural content generation is necessary as well. Previous research and background information in this area are investigated in section two.

Based on knowledge in the areas of serious games and procedural content generation, a game combining aspects of these areas will be designed. The game will be adaptable to different areas of knowledge but the created prototype game will focus on one small area. The area of knowledge chosen for the prototype game is Viking nutrition. To keep the game environment as believable as possible, the environment design is based on historical references of Viking history.

The prototype game is created in Unity3D using several different procedural methods for environment generation. The learning method in the game is based on the "Exploratory Learning" concept. The game will contain several knowledge elements placed in the game environment and the player will have to explore

the game environment and interact with the different elements to obtain knowledge. Design and implementation of the game is described in section three.

An online test based on the problem definition will be used to evaluate the game. The test is mainly designed to evaluate the strength of the content generation through the notion of semantic congruence but will also attempt to investigate whether learning is occurring when test participants play the game. Method and results of the test is described in section five.

2 Background

2.1 Learning Science

In the following section aspects of learning science will be investigated in relation to serious games. General concepts of learning and learning in games will be introduced as well as research in the area of serious games and their learning methods. Finally a model for Exploratory Learning will be described.

2.1.1 Constructivist Learning

In the beginning of the 1960ies researchers of cognitive psychology began questioning the effectiveness of the instructional learning approach that were used in schools (Bruner, 1971). It was claimed that schools did not educate properly for the society they were a part of. This led to a series of radical ideas about teaching and learning based on constructivism. In constructivist learning knowledge is obtained when the learner interprets information within the context of their own experience. Bruner (1971), who was one of the pioneers in the area, talks about this period in "The Process of Education Revisited" and tries to reflect on the problems and progress in the educational system. He suggests that the way of teaching in the 60ies was one of the main problems. The outcome of Bruner's studies was the idea of Discovery Learning (Neber, 2012) where the student is no longer just a receiver of knowledge but a participant in the process of education.

Discovery Learning takes its roots in problem solving where the learner has to use prior experience to solve problems. By discovering new facts and reflecting upon them, new knowledge can be derived. Discovery Learning became a catalyst for several other learning theories.

Two of those learning theories are the closely related "Problem Based Learning" (PBL) and "Inquiry Learning" (IL) (Neber, 2012). PLB is often used in medical education (Jonassen & Hung, 2012) and IL in the practice of Scientific Inquiry(Caliskan, 2012). PBL often uses text based scenarios for hypothetical-deductive reasoning whereas IL is focused on posing questions and acquiring data for the construction of arguments.

Hmelo-Silver et al. (2007) argues that PBL and IL are close to indistinguishable and often overlaps. Furthermore they argue that when PBL and IL are used it is often in guided goal-based environments and not in open learning scenarios where the student is free to explore.

Another derivative of Discovery Learning is "Exploratory Learning". As de Freitas and Neumann(2009) argues with their Exploratory Learning Model (ELM): independent exploration in the learning environments, combined with prior experience, will lead to new experience (learning) through reflection, forming abstract concepts and testing. In contrast to PBL and IL Exploratory Learning is an independent and free exploration of the learning scenario. The ELM will be more thoroughly described later.

2.1.2 Game Based Learning

When using games for educational purposes there are three approaches: games designed by others to teach students (serious games), games created by students themselves and ordinary games that are not designed as serious games (Van Eck, 2006). Games created by students can through the activity of designing and creating the game teach the students about the games domain. This is a method that encourages students to work with educational domains in a different way to improve their willingness to learn. The aspect of teaching others is also an effective way to improve a student's own learning.

Ordinary games that are in no way intended to educate can also be used for education. Marc Prensky argues (Prensky, 2002) that despite a reputation of meaningless entertainment and possibly encouragement to violence, games have a positive effect. Some of the more obvious effects of playing is the motor skills acquired by using the controller. Furthermore Squire(2005) showed in a study that ordinary games like Civilization 3 can be used for educational purposes.

Ordinary games can teach about strategy and provide cultural and environmental information (Prensky, 2002). While the knowledge might not be immediately applicable to players real life situation it can be recalled when a situation occurs in the real world which resembles something in a game. One important concept of learning in games is the choices a player encounters and if the outcome of those choices are normally considered right or wrong (Prensky, 2002). Some games allow the player to make decisions that has no influence on real life but still learn something about the outcome of such a decision in real life. Players often make decisions in games that are normally considered to be wrong in real life perspective, in order to see the effects of such a decision in a risk free environment. That does not make the player a bad person. It is similar to thinking about something and actually doing it. Most people have fantasies about things they would never do in the real world.

An example of a game with knowledge elements is the simulation game The Sims. Through this game you can among other things learn about behavioral rules in society, consequences of ordinary actions in everyday life and the general working of a household.

According to Prensky (2002) another game that has been heavily criticized is Grand Theft Auto. In Grand Theft Auto you steal cars, kill people, and break the law in various other ways, but doing so you learn the rules of society and that there are consequences for your actions. In general video games also have a positive effect on the players. By experiencing what is bad and wrong one will know what is right.

2.1.3 Serious Games

Constructivist learning methods can be used in games and other multimedia as these provide highly customizable environments wherein learning can take place in various forms. Games that are designed to provide the player with knowledge are often called serious games. Blanchard and colleagues(2012) defines serious games in the following way:

"an application where learning and game objectives are both of primary concern"

According to cognitive psychologists such as Piaget (1962) children learn through the act of playing. The learning takes place through assimilation with the real world and prior experience. Several researchers within the field of serious games argue that the same is true for video games, and the elements of playing a game and learning is closely related (Gee, 2008; Royle, 2008). They propose that learning takes place in all games through problem solving and mastering the skills needed to overcome obstacles. According to Gee good game design is equivalent to good learning because it is based on the same principles and it is likely that the two areas can benefit from each other.

In serious games the player is taught about a specific domain of knowledge through a professional of that domain(Shaffer, 2004; Gee, 2005; Squire, 2007). The learning takes place through a procedure Gee has described as "Authentic Professionalism"(Gee, 2005). By experiencing first-hand how the character of a profession thinks, behaves and solves problems the player of a serious game becomes an expert within the profession. The character could be a firefighter, chemist or a shopkeeper. The important thing is that the player can identify himself with this character in a believable setting with authentic content (Royle, 2008).

The learning in serious games is implemented as models of real life concepts that can be used for hypothesis testing (Gee, 2008). These models can be scientific theories, objects, situations or abstract concepts, the important thing is the models realism. In general, everything encountered in a serious game needs to be believable and enable generation of situated meaning with the player (Gee, 2008). Situated

meaning is knowledge obtained from serious games that can be applied to real world scenarios (Gee, 2008).

One of the benefits of serious games is their motivational capabilities compared to ordinary school. Gee(2008) points out that the price of failure in a game is lower than the price of failure in the real world. This motivates players to attempt overcoming obstacles where they are likely to fail and in the process of failing the player learns the skills required to defeat the boss. Another motivational element is competition. In contrast to ordinary school, competition in games is often sought out and seen as pleasurable (Gee, 2008). Cooperation with other players is a motivating factor as well and can lead to increased learning as the knowledge of a group can be higher than the knowledge of any individual in that group (Gee, 2008). Cooperation can be players working together to overcome obstacles in the game environment or sharing information and strategies through online communities.

Another benefit of serious games is the challenge they can provide (Balasubramanian & Wilson, 2006). Ordinary school work can be either too easy or too hard for many students but the challenge of a game can more easily be adapted to suit the individual player. By providing a challenge suitable to the players skills the player is more easily motivated and engaged (Balasubramanian & Wilson, 2006).

Serious games are often addressed as something to be integrated into schools but Royle argues that a good serious game also will encourage the students to play at home (Royle, 2008). When serious games are used in schools it is important that they are integrated into the teaching instead of replacing it (Squire, 2007; Balasubramanian & Wilson, 2006). A serious game is a tool for teachers to use in education just like other media. One of the problems when using serious games is that classroom teaching often is incompatible with game playing(Royle, 2008; Balasubramanian & Wilson, 2006; Squire, 2005). The education in schools should evolve to include serious games just like any other media.

Balasubramanian and Wilson (2006) provides a list of requirements for serious games to be properly integrated in education:

1. The design of games and simulations should be sophisticated and challenging enough for students to be cognitively engaged with the game.
2. The content of games and simulations should be aligned with the standards and viable curriculum in schools.
3. The logistics and usability of the games should reflect classroom realities and time constraints in schools.

4. The feedback and assessments embedded in the games should embody measurable learning outcomes.
5. The teacher guides accompanying the games should provide sufficient ideas, activities and resources to enhance students learning.

2.1.4 Creating Serious Games

Serious games can be created by educators alone or in combination with professional game developers(Van Eck, 2006). Such games can be created in many different ways, but most commonly is using some form of Discovery Learning or theories closely related such as the previously described Inquiry Learning or Exploratory Learning. Closely related to games are simulations and microworlds (Rieber, 2005). These systems are often used for learning in the same way as serious games but do not use game elements such as goals and rules. Simulations can be adapted into games by applying goals and rules for the player. Classic examples are seen in the simulation game genre including games like Sim City, The Sims, train simulators and flight simulators.

When creating serious games it is important to acknowledge that not all types of games necessarily are good for all types of learning (Squire, 2007, Van Eck, 2006). Card and board games can be better for learning to manipulate numbers, pattern recognition and matching concepts. Quiz like games are better at simple facts and the like. Arcade like games are better at improving motor skills and visual processing. Adventure like games in various forms are better at open-ended learning environments, promoting hypothesis testing and problem solving. Finally the different types of games and learning can be overlapping in different ways to create more interesting and varied games.

Some of the more commonly agreed features necessary in a good serious game are authentic content in believable settings (Royle, 2008; Gee, 2005). It is important that the knowledge of the domain within a serious game is authentic, but it should be created in a setting that is believable to the player to facilitate learning. Furthermore obstacles encountered in a serious game should be directly linked to the domain of knowledge (Royle, 2008). In this way the player is encouraged to acquire knowledge used to overcome the obstacle.

Examples of serious games developed with focus on learning taxonomies are Environmental Detectives (Environmental Detectives, 2003), Hazmat: Hotzone (Hazmat: Hotzone, 2005), and River City (River City, 2009). River City was tested on approximately 2000 students to investigate the effects of Inquiry Learning in a game, the effects in learning with such a system compared to ordinary learning and to find strengths and weaknesses of Multi-User Virtual Environments. Initial results indicated that both students and

teachers were highly engaged when using the system, the students attendance improved and disruptive behavior decreased. Furthermore results indicated that the system could facilitate good Inquiry Learning (Ketelhut et al. 2010).

When creating serious games the actual context of the game is important when deciding on the type of game and learning theories to be used. The area of knowledge in the game can call for or rule out specific types of games. Game users, game type, and knowledge area have an effect on the learning taxonomy used. Based on the fact that the game will be procedurally generated, the notion of an open-ended exploratory environment fits right in. Such an environment should be easily adaptable to Exploratory Learning.

2.1.5 The Exploratory Learning Model

De Freitas and Neumann (2009) have developed a model for the use of Exploratory Learning in virtual environments that is well applicable to a game created with procedural content generation. This model is referred to as the "Exploratory Learning Model" (ELM). The following section will describe and argue why this model is well suited to the intended educational game.

ELM is a descriptive model of a person's learning. It is created based on previous learning models and research in virtual learning environments including e-learning (web-based learning) and other technology enhanced learning methods. The basis of the model is to use learners' prior experiences in order to scaffold new experience-based learning by exploring the environment. ELM is illustrated in fig. 1.

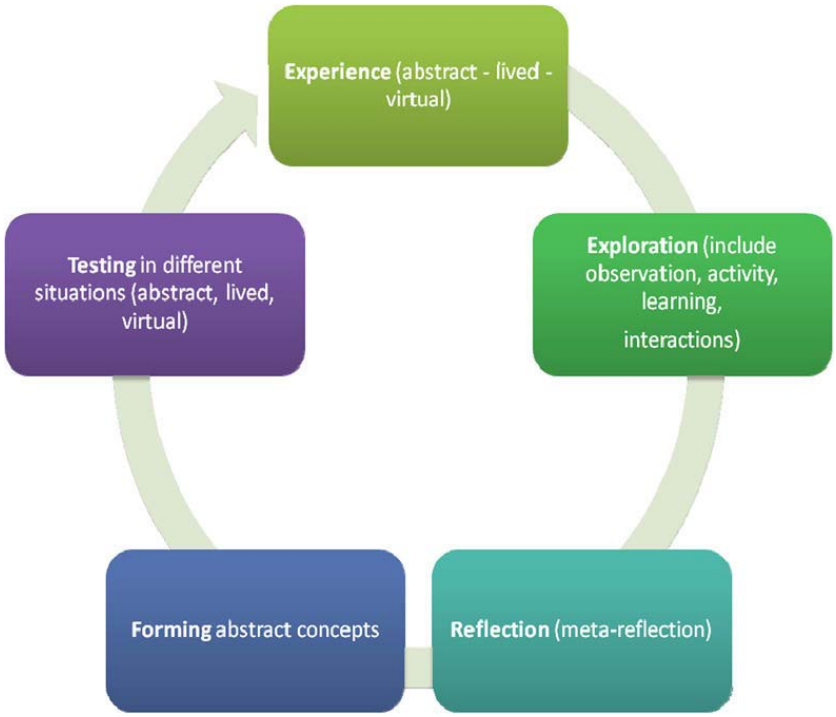


Fig. 1: The Exploratory Learning Model (de Freitas & Neumann, 2009)

The ELM works with five steps, Experience, Exploration, Reflection, Forming and Testing. The five steps run in a circle when learning through a virtual learning environment.

Experience

The learner uses previous experiences as the basis for creating new knowledge. These previous experiences can come from the real world experiences, abstract ideas, or virtual experiences. Prior to the ELM only real world experiences were considered but de Freitas and Neumann (2009) argue that experiences obtained from virtual (learning) environments are just as important.

Exploration

Through the learning environment the learner will observe occurrences in the environment and perform activities. He will also interact with the environment or receive direct knowledge. This is the exploration part of the model and where the actual created environment comes into play. Different elements within the game environment will provide the learner with information in various forms. According to Royle (2008) various forms such as text, speech, and video should be used to maximize literacy.

Reflection

Various aspects of the exploration are reflected upon and new meta-reflections are generated. The information gathered through the exploration is possibly used to form new concepts through the next step.

Forming

Through reflection, new abstract concepts are developed. This is where potential new knowledge is generated. The new knowledge is based on the previous experience and experience gained from exploring put into ideas of how new concepts might work.

Testing

The newly formed abstract concepts are tested to validate the potential knowledge. This validation can take place through abstract ideas, the real world or in the virtual environment.

Through the five steps of ELM, new knowledge is developed, and therefore the purpose of an educational game using ELM is to keep the learner going through the five iterations. The overarching goal of ELM is the transfer of learning between virtual environments and real life.

The ELM has been used in two different educational games in medical education (de Freitas & Neumann, 2009). The first game was a simulation of a bomb going off in a crowded place and leaving many casualties. The game was designed to train medics in triage and therefore called the Triage Trainer. The game was designed to mimic a real life disaster and the impact it might have on the medic and for that reason the game had high levels of fidelity to increase realism as much as possible. After testing upon several different medics it was concluded the system had a slightly significant difference compared to normal face-to-face training which were positive in regards to learning.

The second game was a game designed to teach clinical procedures to hospital staff in areas of infection control. Although not fully developed the Infection Control game demonstrated a good use of the ELM. In general both games help validate the ELM.

2.2 Procedural Content Generation

The following section will introduce the area of procedural content generation with investigations into the field of research and examples.

2.2.1 Introducing Procedural Content Generation

Procedural content generation (PCG) in games is the science of creating game content automatically through algorithms based on a few specific parameters. PCG is for the most part developed within the games industry but some scientific research is also available.

Once of the first examples of PCG is seen in the game Elite created in 1984 (Togelius et al. 2010). Elite is a space trading and adventure game taking place in a large universe of hundreds of star systems. This is not a big challenge to create today but with the limited amount of memory and storage available in computers from 1984 such a large game world was a challenge. The game world was created through a seeding system where all the planets were represented with a few numbers. When running the seed trough the generation algorithm the universe would be "created" and detailed information about the planets could be extracted. This reduced the amount of memory needed to an acceptable level.

Elite is a good example of the advantages of PCG and since the creation of Elite other and more advanced methods of PCG has been developed. PCG is often used in the Real Time Strategy (RTS) genre for map generation and to minimize the size in bytes of a map when sending it through a network to other players. PCG is also used for generating large 3D worlds with high fidelity and realism to take some of the workload off the 3D artists and game designers (Nareyek, 2007). PCG is an area of ongoing research and development to raise aspects such as realism, variation and performance in games to a higher level.

Classic examples of games using PCG is Diablo 2 using PCG for making varied but controlled environments, Minecraft using PCG for generating a large open-ended 3D world, and the Civilization series using PCG for map generation. The area of where PCG can be used is large and varied and thus the approaches to PCG are just as varied. Togelius et al. (2007) have attempted to clarify some of the basic aspects of PCG and the following sections is based on their work.

2.2.2 Major Distinctions in using Procedural Content Generation

One of the major distinctions in PCG is between Offline use of PCG and Online use of PCG (Togelius et al. 2010). With the Offline approach PCG is used to create game content prior to the game release. An example is creation of large vegetation areas or other game sections that can take a large workload off the developers. With the Online approach PCG is used at runtime. This is for example to create the second level

of a game as the first level is completed. The second level simply does not exist until then. Online use can also be the process of creating a random map while loading a new RTS game.

Another major distinction of PCG is between necessary and optional content (Togelius et al. 2010). If PCG is used to create content in a game that is necessary for a player to progress, then the PCG must always be correct or the player will not be able to complete the game. This requires a lot of control over the PCG algorithm used. If PCG is used for optional content there is more room for error and variation because the player can but is not required to use this content. This can for example be weapons and equipment found in a game.

2.2.3 Using Procedural Content Generation

PCG will normally apply a set of parameters to create expanded game content. The representation of these parameters can vary a lot. How the parameters are used also has an effect on the control or "randomness" of the created content. One example of a very compact representation is to have a seed used with the systems random number generator. The content will then be generated based on the seed. The parameters can also be represented as real-valued parameters in multidimensional vectors or a mix of the two approaches that best suits the system (Togelius et al. 2010).

Deterministic vs. Stochastic

Using parameters to create game content can be done in a more or less controlled way. If an algorithm always produces the same result, given the same parameter, it is said to be deterministic generation (Togelius et al. 2010). The generation is based on what the parameters actually represent. In a deterministic system everything is represented by the parameters but it is possible and often the case that not everything is directly represented by the parameters. An algorithm could produce several different outcomes based on the same parameters thus making the system stochastic (Togelius et al. 2010). Which approach to use is a design issue but it is often easier to customize a system when not using a deterministic approach for generation.

Constructive vs. Generate-and-test

When generating the content there are two main approaches available. The first is a constructive approach where the content generated by the system is final after generating and will not be changed (Togelius et al. 2010). This approach requires the PCG algorithm to only run once and can therefore be the best option for Online generation where speed and performance are issues. With a constructive approach it is important that the generated content is correct as there is no room for correction.

The second approach is a generate-and-test approach where content is generated and then tested according to certain criteria (Togelius et al. 2010). If the generated content does not fulfill the criteria all or some of the content will be regenerated. The process of generate-and-test continues until all criteria are met. This method allows the system to make errors and ensure that the final content is always good enough if the right criteria has been set. The drawback of such an approach is that running many iterations of generation can take a lot of time often making it unsuitable for Online generation.

2.2.4 Search-Based Procedural Content Generation

A more advanced approach to PCG is described by Togelius et al. (2007) as search-based PCG (SBPCG). This approach uses the generate-and-test method but instead of generating once and then regenerating SBPCG generates a range of candidates and grades them according to a fitness function. The aim is to modify or regenerate the candidates in a way that will give them a higher fitness value. One way to handle this is with an evolutionary algorithm (EA). A good example of SBPCG using an EA is seen in Sorenson and Pasquirs research (Sorenson & Pasquir, 2010).

An EA ranks the candidates with a fitness function for each generation and the candidates with a low fitness are replaced with mutated (randomly modified) versions of high fitness candidates. An important aspect of designing such a system is how the data structures handled by the EA (genotypes) are mapped to data structures handled by the fitness function (phenotypes). One approach is direct encoding where the size of genotypes is linearly proportional to phenotypes and the elements of the genotype maps to a corresponding element in the phenotype. Another approach is indirect mapping where the genotype is indirectly mapped to the phenotype and thus does not need to be proportional to the phenotype (Togelius et al. 2010).

There are several approaches to generating the fitness function and the most simple is to have it evaluating specific features such as the number of trees generated or if a generated path leads to the right place. This is called a direct fitness function. Other approaches include simulation-based fitness functions where an AI agent plays the game to simulate a real play-through, and interactive fitness functions where game play is evaluated through a test person (Togelius et al. 2010).

2.2.5 Procedural Content Generation in Practice

There are several benefits of PCG. Oone of the more notable being replayability (Smith et al. 2011). Games where core content is procedurally generated often have the advantage that each play-through of the game will be different from the others. This means that the player can get a completely new experience from playing the same game twice. Closely related is the notion of adaptability where the game

environment adapts to the skill level and/or playing style of the player (Smith et al. 2011). This can help the player to a better experience and enhance replayability because the second play-through will adapt to the skills the player gained from the first play-through.

PCG also has the ability to affect game mechanics and dynamics (Smith et al. 2011). In games where core mechanics are affected by PCG, the player will have the option to adapt playing styles that utilizes the mechanics provided. This could be achieved through procedurally generated weapons and equipment. PCG can also allow the player to have some control over the game environment (Smith et al. 2011). This can be achieved through adaption to the players choices and strategies in the game. This allows the player to actively take action to change the game environment. An example could be that the player deliberately use a secondary strategy to overcome trivial obstacles because the boss will be generated in a way that counters this strategy.

Random Map Generation

One of the areas where PCG is often used is in map generation. This can be maps of all types and for all types of games. One of the earlier and commonly used methods is the random map generation. Random map generation can give players a map looking different every time a new game is started. A random map often has some controllable attributes such as water to land ratio, land type (islands, rivers, continents), vegetation amount, resource amounts, and size. The approaches for creating such maps are varied but one of the more commonly known is using a random number generator with a seed and based on the seed the algorithm will create a map. One common use of this is with a perlin noise generator. Perlin noise can be well suited for generating random maps as it can be used for both 2D and 3D maps and is highly controllable.

Roden and Parberry (Roden & Parberry, 2004) have shown that massive 3D worlds can be created through PCG in real-time. The general idea was to represent the terrain data at the highest level of abstraction and then add details as the player progressed. Their approach was to create a high level 2D representation of the map from where information about the 3D map were extracted. Later procedures added higher details and features according to the players position. This made it possible to create very large 3D environments in real-time.

Level Generation

Generating levels in games that satisfy the general game criteria is seen in 2D games (Sorenson & Pasquir, 2010) and 3D games (Roden & Parberry, 2004). Sorenson and Pasquir uses SBPCG to create playable 2D

platform levels based on certain criteria for the obstacles the player must overcome. The approach is a generic nature, and can be applied to all types of platform games. Sorenson and Pasquir uses a constructive approach where the content is created and applied in layers.

Trees and Vegetation

Game developer Mick West(2008) demonstrated an approach to randomly scattering trees while still making it look natural. In nature trees are not randomly placed but grows in a natural order. A tree only grows where it is not shadowed by other trees and if there are the necessary recourses in the ground. They also don't randomly appear but are growing from seeds. As Mick West demonstrated this requires an approach that does not appear random. The suggested approach was a Mass-Spring scattering where each tree is connected to its neighbors through springs. When "relaxing" the trees the springs will take care of the positioning and maintaining constraints such as trees not being too close.

In general there are many different approaches for creating trees and vegetation but the key is that nature works according to advanced processes and, though appearing random, often are not. This pseudo randomness is the goal when creating natural content.

When looking at the possibilities and uses of PCG it is feasible to think that content in a game created for education, can be distributed through a PCG approach. If a game world is created through PCG the distribution of educational content will most likely be another step in the process. Considering the approach of Exploratory Learning a procedurally generated game world could probably support the requirements as well as making the game more appealing to players by providing a different world for every game instance.

3 Development

This section is dedicated to the development and implementation of the prototype game that will be used for testing. The game is based on the principals from the previous section regarding serious games, the Exploratory Learning model and procedural content generation. The game can be found at http://megacraft.dk/VikingGame/WebPlayer.html or on the provided CD.

3.1 System Description

There are many areas of knowledge that can be provided through games, but any type of game might not support any type of knowledge area (Squire, 2007). A game designed to teach math would have a lot of different properties than a game designed to teach history. Learning math is based on solving problems whereas history is more based on obtaining specific information. The activities performed in the history or math teaching games, will probably need to be very different to facilitate the best learning. This makes it difficult to create a system that easily could be adapted to teaching history instead of math. To better facilitate the knowledge area of the games using the proposed system, a specific area of knowledge must be chosen before system design.

There are several approaches to better facilitate the use of PCG combined with Exploratory Learning. A large procedurally generated game environment could be used. This would allow for exploration of the environment and make each game instance more distinguished. Having the environment represent something that is easily linked with the knowledge domain of the game will help facilitate the learning process or provide actual information.

It should be noted that the Exploratory Learning concept is not necessarily connected with the exploration game type. An exploration game is one that is directed at the explorer player-type. In such a game the main aspect of the game is to explore the environment and learn how it is structured and works. Although this can lead to learning within the knowledge domain of the game it is not necessarily the case. Exploratory Learning is a process the player goes through when learning and it is this process that one attempts to produce through the game. Making the player learn through actually exploring the game will possibly make the process of Exploratory Learning easier.

Some common areas of knowledge that might be applicable with the above mentioned criteria are geography, culture, and history. The present system will be designed for these or similar areas.

The proposed system is one that will automatically generate a game world when the game is started. The game world must be different every time and facilitate Exploratory Learning as described by de Freitas & Neumann (2008). The system must be such that it can easily be adapted to similar topics of learning.

Specific system goals:

1. Procedural generation of a 3D game world that can be changed to fit the specific learning area, including terrain, vegetation, oceans and rivers, hills, and mountains.
2. Procedural generation of content in the game world that is related to the specific learning area.
3. Procedural distribution of elements in the game that provides the player with knowledge about the specific learning area.
4. Strong semantic congruence between the generated elements of the game.

Generating the Game

As the game environment is created when the game is started it will use Online generation. This means that there should be some focus on speed and performance of the generation. The amount of time a person is willing to wait while a game is loading varies, but the longer time, the more players will lose interest in the game and might stop before actually playing. A maximum load time on slow computers should not be more than one minute.

To accomplish a short loading time the system should not use a generate-and-test approach as this has the potential of drastically increasing the loading time. The system will be based on a constructive approach where all content is generated in one cycle. With a constructive approach the system must be designed in a way that minimizes the amount of errors. To accomplish this a strict structure of the PCG is needed.

Procedural generation structure:

1. Base terrain: the actual walk-able terrain with features such as water and forest.
2. Frame for learning content: content that does not necessarily provide knowledge but contains relations to the area of knowledge in the game.
3. Knowledge elements: the actual content in the game that provides knowledge.

Each of the three points in the structure can be divided into smaller parts that can be separately created. With a structure consisting of many smaller parts, changes can easily be made without interfering with the general content generation by taking one part and altering it. This way, aspects of the system can be changed to fit that of another learning area.

Base Terrain

There are several approaches to consider for terrain generation, but the main goal is to make it's performance as effective as possible. A height map generation based on given information might be the best approach if generated in an efficient way. The main design issues for the system however are not exactly how the terrain is generated but how the features of the terrain is determined. The features must be changeable to fit any knowledge domain in the chosen area. Some of the main features are:

- Size of the world
- Amount and type of water
- Terrain elevations
- Base terrain types
- Amount and type of vegetation

Creating a game world based on these features and making them easy to change will provide a good base for a terrain generator suitable for adaption to different knowledge areas. It is also important that the features of the terrain can be changed by later steps in the generation process.

Frame for Learning Content

When the base terrain is created, features that are directly connected to the knowledge area are added. These features can be very specific such as historical or cultural places, specific environmental details or natural recourses. Common for all are that they are placed in the game world at specific positions according to the generated terrain. A village could be placed close to a river, a mine on a hill or specific animals in the forest. The elements created will act as a frame for the knowledge elements and might be the topic of that knowledge or the knowledge itself.

The elements can get a certain portion of the terrain assigned such as a village, and change the terrain to fit the element. Other elements can be combined with the terrain. In general the terrain needs to be dynamically changeable to support the need of these elements.

As the elements are very diverse they don't necessarily have any attributes in common, but they do require some specific methods to be properly placed within the game world such as the ability to identify locations in the terrain with specific features and modifying the terrain. Some elements should also have the ability to start a new string of PCG to create advanced features. This could for example be a village generating houses and streets.

Knowledge Elements

The knowledge elements in the game are the objects providing the actual knowledge to the player. Often it will be better to add knowledge elements during one of the previous generation steps, especially when creating the frames for learning content. The key is to not only randomly scatter the actual knowledge elements throughout the game world, but to add them at places where they are relevant and can be accessed by the player. The actual placement of a knowledge element will vary based on the knowledge it provides.

A knowledge element could be created during the creation of vegetation throughout the map. This could be an element such as a type of tree that is relevant for the player to investigate. This tree then comes in two types, one is a basic game object and the other is the same basic game object but with a knowledge component added, making the tree a knowledge element. When the tree is created it can then either be a standard game object or a knowledge element.

In other cases it would be most simple to integrate the creation of knowledge elements with the frameworks created for providing knowledge. In the case where a village is generated as such a framework it makes better sense to generate the knowledge elements as part of the village. As with the tree in the forest, objects in the village can occur in two versions, one as a normal object and one as a knowledge element. With this approach it can easily be decided which objects will provide the knowledge available in the village and could even vary from instance to instance of the game based on the PCG approach.

In yet another case, knowledge elements could be added directly by determining relevant positions for this element. This could be specific animals spawned in the forest, fish in the water, or lava rocks from a volcano, serving as a framework for learning. These are the final elements to generate as they require information about the final environment to be properly positioned.

Learning objects can be created from already existing objects or as completely new objects. Common for them is the way they will work when interacting with the player. The knowledge elements will have standardized ways of conveying their knowledge.

The knowledge elements will have a conversation system that can be used with common Non Player Characters (NPC's) and a helper NPC that will follow the player around. This conversation system should make it possible for the player to ask questions to further explore the topic being discussed with a NPC. The content of the conversation should be easily modified to suit all the conversations needed in the game.

Another way to provide knowledge will be through videos playing when interacting with a knowledge element. This could be a video from the real world or of animated demonstrations.

Semantic Congruence

Semantic congruence is related to the meaning obtained when combining information from cross-modal elements or two uni-modal elements (Laurienti et al. 2004). A main goal for the system is to provide a strong semantic congruence between the elements in the game. This is both related to the knowledge areas, where the knowledge being provided should fit the surroundings such as a fisherman close to the water, but also to the physical appearance of objects in the game. A tree should not be placed on top of a house, the house should not appear to be levitating above the ground and a person's arm should not be sticking out through a wall. Looking for the semantic congruence will help determine the strength of the PCG and make sure the game environment is believable which is a requirement for serious games.

3.2 System Implementation

To evaluate the proposed system, a prototype game was created. The goal of the prototype game was to test the core principals of distributing knowledge elements in a procedurally generated game. The game was in no way focused on graphics or game play, these elements were only used to facilitate a feasible evaluation of the PCG.

The first step in the development process was to decide upon the area of knowledge that should be available in the game. The chosen area was Viking history as there is a lot of content readily available. To narrow the area down, information about the Viking diet was chosen to be the main area of knowledge provided in the game.

Creating the game, based on Viking history, did not just require information about the area of knowledge, but also the general environment of the Vikings. Information about the geographical properties of the areas where Vikings lived, how they lived, and other areas were needed as well. All this information was acquired through historical literature on Vikings and will be further described during the description of implementing the different areas.

3.2.1 The Game Engine

Unity3D(Unity3D, 2012) was decided as the best choice for a quick and efficient way of creating the desired prototype game, and the author had prior experience with this game engine. Unity3D provides a strong physics based 3D engine that is designed to make game development easy. It is a program where much of the design and implementation of graphic elements are taking place on a high level, and there are many premade elements that can be used.

For scripting, Unity3D had two main options; C# or JavaScript. C# scripting was used throughout the game. Unity3D uses a system of prefabs that can be instantiated during runtime. Prefabs can be any type of game object such as a box, a tree or an advanced animated and scripted object. These prefabs are the actual objects that is created through the content generation.

3.2.2 Overview

There are three main scripts that composes the structure of the procedural generation of the game. The flow of the three main scripts (ImprovedMapGeneration, VillageGenerator and FarmGenerator) are:

ImprovedMapGeneration:

Generates base terrain

Generates River

Generates Ocean

Generates forest

Finds the best village location and generates roads

Starts village generation (separate process in VillageGenerator)

VillageGenerator:

Generates farms (separate process in FarmGenerator)

Generates fields

Generates separate village knowledge elements

FarmGenerator:

Generates fence

Generates main building

Generates buildings of type 2

Generates buildings of type 3

Generates fence within the farm

Generates building of type 4

Generates vegetation

Generates animal types 1

Generates animal types 2

Generates farm inhabitants

3.2.3 ImprovedMapGeneration

The ImprovedMapGeneration script is the main component of the PCG system. The scripts most important function is to generate the terrain. The terrain is generated with a tile-map system. With the tile-map system the base terrain is divided into small, square, 2-dimentional parts (tiles) that are combined to make the whole world. The size of the game world is decided by the amount of tiles created, which made it possible to easily adjust the desired size during development. One tile consists of a 2D plane with a texture and a script named TileScript which holds all the tile information.

TileScript

The TileScript contains and manages all the different terrain types and their attributes. The type of the tile is stored in an enum variable named tileType. The main use of the TileScript is to change a tiles type from for example grass to river as is used in the river generation. Two public methods are used to get the tile type or set the tile type. When the tile type is changed, a check will be made to see if any specific actions are needed, such as removing vegetation. It is sometimes necessary to modify the neighbors of a tile so a function for finding and storing the neighbors are available:

```
SetNeighbors (GameObject[,] gameTiles, int Width, int Height)
```

Fig. 2: Function in TileScript used to set neighbor tiles

SetNeighbors takes the array of all map-tiles, map width and map height as inputs and finds the tiles surrounding the current tile. These tiles can then be easily accessed to modify them. An example of modifying neighbor tiles is the function for making beaches which checks if tiles are neighbors to the ocean and then makes them the right type of beach.

Trees and vegetation are spawned as part of the map-tiles, and thus the TileScript also contains functions for this. The generation of vegetation is described in the forest generation section later.

Pre-generation of Map Tiles

The first step of the PCG is to generate the base terrain. This is done by making a 2D array called gameTiles with a size, based on the public variables Width and Height which are the width and height of the map in number of map-tiles. Every tile is already loaded into the game to save generation time so only positioning and storing the map tiles in the designated 2D array is needed. The default terrain of the tiles is grass and after all tiles of the map are created, the terrain will be altered. To manage progression and avoid choking the CPU, map-tiles are created in small portions for every frame. This allows other code in the game to run while the map is being created.

```
int currentWidth
int currentHeight
int totalGeneratedTiles
int tilesPerFrame
if generating map
            int tilesGeneratedBeforeThisFrame = totalGeneratedTiles
            while totalGeneratedTiles < tilesGeneratedBeforeThisFrame + tilesPerFrame
                        mapTiles[currentWidth, currentHeight] = new tile
                        Set worldposition of tile
                        Set tile options
                        currentWidth ++
                                    if currentWidth > mapwidth -1
                                                set currentWidth to 0
                                                currentHeight ++
                        totalGeneratedTiles ++
```

Fig. 3: Pseudo code of the pre generation of map tiles

The pseudo code in fig. 3 illustrates the pre generation of map tiles. currentWidth and currentHeight is the positions of the tile being generated. TotalGeneratedTiles is the total amount of tiles generated in the game and tilesPerFrame is the amount of tiles to be generated each frame. Every frame, while the map is being generated, map tiles are generated until the total number of tiles generated is increased by the amount of tiles to generate per frame. The tiles are positioned in x/y coordinates based on the total number of tiles generated . After map-tiles are created and positioned, the terrain generation process begin.

Generation Step 1: Ocean

There is a 50% chance that an ocean is generated. The ocean has several attributes that could be changed to customize it to the need of the game and world size. The current ocean generation was designed to fit the nature of a game about Vikings but could be tweaked to fit other scenarios or completely replaced with another system for generating ocean.

The ocean is based on Simplex noise, an improved version of Perlin noise. The Simplex noise produces three dimensional coherent noise where the x and y coordinates are equivalent to the position of map tiles and the z coordinate is the theoretical height of the tile used to determine if the tile is water or not. In a normal implementation using Simplex noise, values below zero would be water, zero would be ground level and everything above zero would be elevated terrain. Fig. 4 illustrates how Simplex noise might look, and Fig. 5 how it would be translated into water:

Fig. 4: Simplex noise. White indicates higher points in the terrain (elevation) and black are the lower points (water)

Fig. 5: Water from Simplex noise found using a threshold. Black is water and white is land

The Simplex noise system is easily tweaked through several attributes and can produce a varied terrain. In the case of this prototype game, the Simplex noise is used to generate a random looking ocean along the edge of the map. This is done to keep the amount of ocean from covering too much of the game world as it is not very large.

The ocean is generated by first determining a point along the edge of the map as a starting point, and from this point it is spreading out. To keep the ocean from spreading too far into the map, only a small ocean is created at the starting point, but several other points to the left and right will act as starting points as well. In the chosen map size the starting points for ocean generation might be distributed as seen in fig. 6:

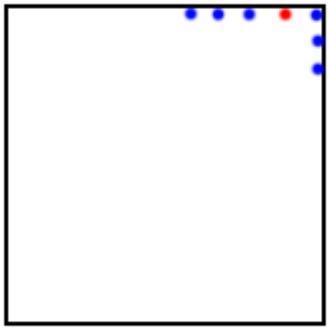


Fig. 6: Ocean spawn points. Red is the initial spawn point, blues are additional spawn points

The distance between spawn points, and the number of spawn points were adjustable. All tiles around the spawn point will have its Simplex noise z coordinate (height) checked to see if it is above or below zero. If it is on the same side of zero as the spawn point representing ocean it will be turned into ocean.

```
OpenTiles = list of unchecked water tiles
int tilesToGenerate
int tilesGenerated
while OpenTiles contains tiles and tilesGenerated <= tilesToGenerate
            currentTile = first tile in OpenTiles
            for all surrounding tiles of currentTile
                        if spawn height has the same sign as surrounding tile height
                                    make tile ocean
                                    add tile to OpenTiles
                                    tilesGenerated ++
            remove currentTile from OpenTiles
```

Fig. 7: Pseudo code for the generation of ocean tiles for one spawn point

The pseudo code in fig. 7 illustrates the generation of water tiles for one spawn point. tilesToGenerate is the total number of ocean tiles to be generated to the spawn point. tilesGenerated is the amount of tiles already generated for the spawn point. While there is tiles in the list OpenTiles or the amount of tiles to generate is reached, the tiles surrounding the first tile in OpenTiles will be turned into ocean, if it has the same sign (based on the simple noise) as the ocean spawn tile. All tiles turned to ocean is added to OpenTiles and all tiles who have had their neighbors processed is removed from OpenTiles.

Generation Step 2: River

If no ocean has been generated, a river is always generated to make sure the game world has some form of water. If an ocean has been generated there is a 50% chance a river will be generated as well. The river will start on the edge of the map and then move through the map until it reaches another edge. If an ocean has been generated, the river will start at the same point as the ocean to make sure they are combined.

The river is generated with the GenerateRiver function which takes three attributes of the river as input. These attributes are the starting point of the river, the direction of the river, and whether the river flows horizontally or vertically. If no ocean has been created the starting point of the river is at a random position along the edge of the map.

The direction of the river is a 2D vector and is based on where the starting point is. For example if the river starts at the left side of the map, it will move towards the right side. If the starting point is below the middle of the map height, it will move upwards. This makes sure the river will always cross the map.

```
startPoint(x,y)
direction(x,y)
if startPoint.x == 0
            direction.x == 1
            if startPoint.y < mapWidth/2
                        direction.y = 1
            else
                        direction.y = -1
if startPoint.x == mapWidth
            direction.x == -1
            if startPoint.y < mapWidth/2
                        direction.y = 1
            else
                        direction.y = -1
if startPoint.y == 0
            direction.y == 1
            if startPoint.x < mapHeight/2
                        direction.x = 1
            else
                        direction.x = -1
if startPoint.y == mapHeight
            direction.y == -1
            if startPoint.x < mapHeight/2
                        direction.x = 1
            else
                        direction.x = -1
```

Fig. 8: Pseudo code of river direction determination

Fig. 8 illustrates the determination of the river direction. startPoint is the starting tile position of the river and direction is the direction of the river. The direction is determined by the startPoint and the width and height of the map. When the direction is determined the GenerateRiver functions generates the river:

```
GenerateRiver(startPosition, direction, horizontal)
tile currentTile = startPosition
while not at the edge of the map
            randomX = 0 or 1
            randomY = 0 or 1
            if randomX == 0
                        randomY = 1
            tile nextTile
            if river is horizontal
                        nextTile.position.x = randomX * direction.x +
                        currentTile.position.x
                        nextTile.position.y = randomY * direction.y +
                        currentTile.position.y
            else
                        nextTile.position.x = randomY * direction.x +
                        currentTile.position.x

                        nextTile.position.y = randomX * direction.y +
                        currentTile.position.y
            set nextTile to river
            currentTile = nextTile
```

Fig. 9: Pseudo code of the river generation

The pseudo code in fig. 9 illustrates the generation of river tiles. startPosition is the starting tile of the river, direction is the rivers direction, and horizontal determines if the river flows horizontally or vertically. Based on currentTile the next tile to be turned into water is determined. The position of the next tile in relation to the current tile is based on direction and horizontal. When nextTile is determined it is turned to river and currentTile is set to nextTile. This process continues untill the edge of the map is reached.

Generation Step 3: Smooth

This small generation step will smooth the river and ocean so they look less square. If two sides of the same non-river tile is bordering a river, half the tile will be turned into river by changing its texture. At the ocean, non-ocean tiles bordering ocean tiles will be turned into beach tiles. The tiles bordering beach tiles will be smoothed in the same way as tiles bordering a river. Example can be seen in fig. 10:

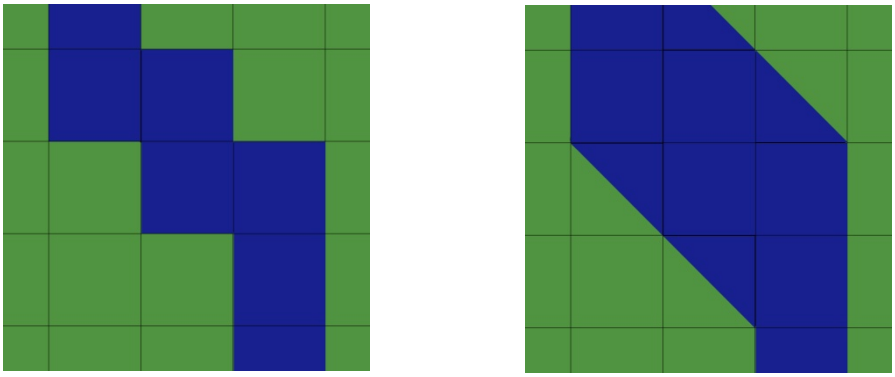


Fig. 10: River before(left) and after(right) smoothing

Generation Step 4: Forest

Forest is generated as smaller patches randomly scattered over the remaining grass areas of the map. The number of forest patches being generated is:

(mapWidth * mapHeight) / forestAmmount

and the size of a forest patches is:

(mapWidth + mapHeight) / forestSize

This way the amount of forest being created is based on the size of the map but can also be adjusted if needed. In the prototype game forestAmmount is an arbitrary value of 200 and forestSize is and arbitrary value of 2. The forest patches are created with the function GenerateForest(), forestDensity can be modified to adjust the density of the generated forest patches:

```
int forestDensity = the density of the forest
List Open = list of forest tiles generated
int tilesGenerated = 0
while Open contains tiles and tilesGenerated < number of tiles to generate
            foreach tile T in Open
                        foreach tile S in surrounding tiles of T
                                    int randomValue = random value between 0 and 100
                                    if randomValue < forestDensity
                                                make tile S forest
                                                add tile S to Open
                                                tilesGenerated ++
                        remove tile T from Open
            currentTile = nextTile
```

Fig. 11: Pseudo code for generation of a forest patch

forestDensity determines the chance that a tile next to a forest tile also becomes a forest tile. forestDensity can be modified to decrease or increase the density of the forest tiles. tilesGenerated counts the forest tiles being generated and while it is lower than the number of forest tiles to generate, new forest tiles is generated. All newly created forest tiles are added to the Open list. The surrounding tiles of all the tiles in Open are turned to forest tiles if randomValue is lower than de predefined forestDensity. If a tile is turned to forest it is added to Open. All tiles in Open that have had its surrounding tiles checked are removed from Open. The process continues as long as Open contains tiles or the number of tiles to generate is reached.

After the forest tiles have been created, a function in the TileScript is called in all forest tiles to spawn vegetation. In the current state, two types of vegetation is spawned: tree and bush. The bush is the first type of knowledge element being generated. It is simply placed instead of some trees, and the same method could be used to create different types of trees in future versions. A vegetation element is spawned inside each spawn area in fig. 12. The spawn areas are positioned a certain distance from the edge and center of the tile to avoid vegetation elements overlapping each other.

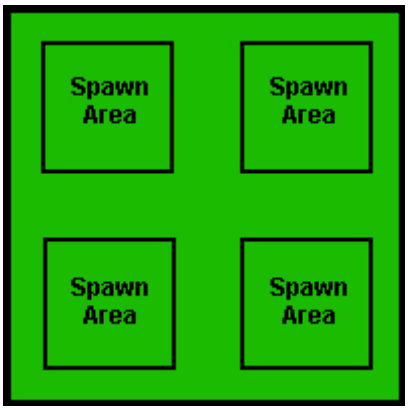


Fig. 12: Vegetation spawn areas on a tile

The forest is the last part of actual terrain generation but more steps could easily be added if necessary. The individual steps of terrain generation could easily be modified or replaced with more advanced approaches to the generation.

Generation Step 5: Generate Village Area and Roads

When all terrain has been generated, focus is shifted towards the learning content of the game. As a framework for knowledge elements, a Viking village is generated, and the first step in this process is to determine the best location for the village. Certain criteria have been set for the village placement:

1. The village must not be too close to the edge of the generated environment. This criteria is set to ensure that the whole village area can be properly used and roads to and from the village will naturally lead to other places within the environment. Furthermore the village will be the center of knowledge for the game so it should be placed fairly central on the map.
2. The village must be placed on areas of the map where the terrain is open. This is to ensure that the village is not generated in the middle of a forest or other obstacles.
3. The village must be fairly close to a river or ocean so the player does not need to cross the entire game world to reach these places. Furthermore villages in the real world are often placed close to sources of water.
4. The village area must not overlap certain terrain types such as river or ocean. This is to ensure that the village will be created on ground that is suited for a village.

The village area will span a certain number of tiles according to the size of the village. In the prototype game the village area is a square of 21x21 tiles, 10 tiles in each direction from the village center tile. The village center tile is used to check for the best position of the village. Every tile on the map will be tried out as the center tile of the village, and based on the other tiles within the village area a score is calculated. The tile with the highest score will be the center of the village. All tiles where the village area would be outside the map or closer than two tiles to the edge of the map are excluded.

The evaluation value for village fit is calculated and assigned to each tile with the function findVillageValues(). The actual calculations of values based on tile type is done for a tile in the following way:

```
findVillageValues(centerTile)
float grassValue = value for grass tiles
float forestValue = value for forest tiles
float halfTileValue = value for half tiles
float value = 0

foreach tile T in the village area
            if T is grass
                        value += grassValue
            else if T is forest
                        value += forestValue
            else if T is halfTile
                        value += halfTileValue
            else
                        break
value *= villagePositionValue()
centerTile.villageValue = value
```

Fig. 13: Pseudo code for determining village value for a tile

findVillageValues() as described by pseudo code in fig. 13 uses predefined values(grassValue, forestValue, halfTileValue) for all tiles in the village area to calculate the village value for the tile in question(centerTile). The villagePositionValue returns a float value based on the position of the tile in question. The value of the tile is then multiplied by this float. The closer the tile is to the center of the map the larger a value will be returned.

When the tile best suited as the village center is found, a 2D array is filled with the tiles of the village area and loaded into the VillageGenerator script.

After the village area is found, a road is created from one side of the village area to the other. This road will connect in both ends to roads going out into the game world. The roads going from each side of the village are created with the custom RoadMaker class.

The RoadMaker class is using two main functions: RoadAToB() and RoadAToBToC(). The first will create a road from some tile A to another tile B and the second will create a road going from tile A to tile C through tile B. The core of the two functions is the same and very similar to AI path finding. In general the goal is to find the shortest path from start to end and make each tile on the path a road:

```
tile currentTile = start
tile end = end
bool atEnd = false
while atEnd == false
            list surroundingTiles = tiles surrounding currentTile
            if surroundingTiles does not contain end
                        tile bestTile = null
                        float bestDistance = infinity
                        foreach tile T in surroundingTiles
                                    float distance = distance from T to end
                                    if distance < bestDistance
                                                bestTile = T
                                                bestDistance = distance
                        bestTile.type = road
                        currentTile = bestTile
```

Fig. 14: Pseudo code for road generation

In the road generation shown in pseudo code in fig. 14 the distance from all surrounding tiles of currentTile to the end is checked. The tile with the shortest distance(bestTile) is turned into road and set to be currentTile. The process continues until end is reached. The textures used for the road tiles are smoothed out in a similar way to the ocean and river tiles. This ensures that the road can go in 8 different directions and still look smooth.

The RoadMaker is used to create roads leading to areas on the map where knowledge elements are placed, so that the player will be more likely to find them. The first road will lead down to the closest water tile (river or ocean) where a fisherman is created. The second road will lead to the closest forest tile where a hunter is created, and from there continue to the edge of the map. At the point where this second road ends the player will spawn when the game starts.

Generation Step 6: Generate the Village

In the final generation step, the village generation is started. The village generation is a separate process that takes place in other scripts. It is based on the village area it has been assigned, and will only act on those tiles. The village is the only framework for knowledge elements in the prototype game, but if others existed they could be designed and initiated in the same way.

3.2.4 Village Generation

To keep the environment as realistic as possible the village is based on excavations of a real Viking village found in Denmark (Sawyer, 2002; Hvass, 2011). Through analysis of the village features it was possible to create a system that will procedurally generate a village with the same characteristics, as the one seen in the excavation.

The first step was to find an overall structure of the village. The Vikings village consisted of a few large farms on either side of a road. The village from the excavation had 6 farms with a road dividing them and other excavations of similar villages have revealed 5 or 6 farms (Sawyer, 2002). This resulted in the decision of creating four farms in the generated village, two on either side of a road. The number of farms in the generation was lower than those found in the excavation to keep the village from being too large. If the village was too large it would take up too much of the game world area and would require more content.

The Viking farms of the excavated village were lying next to each other, separated only by a wooden fence. Each farm covered an area of about 4000-12.000 m^2 but the farms generated in the game will be scaled down a bit to save space in the game environment. The generation of the content within the farms will be described later.

The village generation takes place in the VillageGenerator script. The process is started by a call from ImprovedMapGeneration to the function StartGenerating(). When StartGenerating() is called, the first farm is generated. The farm is generated through the generateFarm() function which takes the farms height, width and type as input. The height and width are the number of village tiles the farm will cover and type is describing the knowledge elements the farm will contain. A list of the types of farms to be created is transferred in a random order to an array that will be used when the farms are created. This way the knowledge elements of the farms will be different from game instance to game instance.

```
GameObject Farm = farm prefab

generateFarm(height, width, type)
tile start = starting tile
array[,] farmTiles = array[height,width]
            for int i = 0; i < width; i++
                        for int j = 0; j < height; j++
                                    farmTiles[i,j] = VillageTiles[ j + start.x, i+
start.y]
gameObject farm = Instantiate Farm
farm.MakeFarm(farmTiles)
return farm
```

Fig. 15: Pseudo code for farm generation

The pseudo code in fig. 15 illustrates the generation of farm objects. Farm is the prefab all farms are created from. The generateFarm function uses height, width and type of the farm to fill the array farmTiles with the correct number of tiles. The gameObject farm is then instantiated as a Farm and its generation is started with the MakeFarm() function which takes the farmTiles as input.

All farms are game objects returned from this function and they are not directly connected to the tiles they are created on, meaning that they easily can be moved around. When the first farm is created it will be moved to a location next to the road and the tiles it covers are changed to dirt through the TileScript. The second farm is then created and moved just like the first. When all the farms have been created and properly placed, fields within the village area are created.

The Vikings planted grain on fields behind the farms, although not in large quantities (Sawyer, 2002; Hvass, 2011). The fields were small, rectangular patches of different grains or fallow fields. The VillageGenerator creates rectangular areas of field right behind the generated farms. The fields are generated with the generateField() function which takes a tile as starting point, width and height as input. The width is the remaining village tiles between the farms and the end of the village area. The height is randomized as 3,4 or 5 tiles. The field area can be seen in fig. 16.

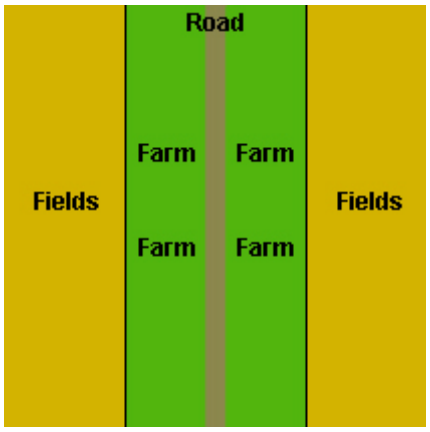


Fig. 16: Village Overview

As a last step of the VillageGenerator script, knowledge elements that are not a part of the individual farms are generated. These knowledge elements are placed on the village area that is left over from the field and farm generation. In the prototype game three knowledge elements are created: beehives, a stack-barn and a farmer. The stack-barn is placed just outside one of the farms and the farmer is close to the fields in the same area as the stack-barn. The beehives are placed just outside a farm in the other end of the village area. If any of these knowledge elements is placed on forest tiles, this and the surrounding tiles are changed to grass tiles.

3.2.5 Farm Generation

Content of the farms is generated by the FarmGenerator script. This script is applied to a game object being instantiated when a farm is created. The actual generation is started when the function MakeFarm() is called. The farm generator creates all the game objects on the farms, and connects them to the farm game object so the whole farm becomes one object and is easily moved.

The MakeFarm() function takes the farm tiles, width, height and farm type as input. This information is then stored in the script. Next it starts the first generation step. The farms can be 3, 4 or 5 tiles wide, so all the generation steps are created to handle different sized farms. The height of all the farms is four tiles.

Farm Generation Step 1: Fence

Based on the excavation, all the farms are surrounded by a wooden fence. This fence is generated with the makeFence() function. The function creates a fence on the farm tiles based on their position. All tiles at the edge of the farm will have one or two fence sections instantiated based on their specific location. The gate into the farm is placed at a random tile connected to the road.

Farm Generation Step 2: Main Building

The excavations revealed that the farms had one main building which were placed halfway between the road and the back fence (Sawyer, 2002; Hvass, 2011). The main building faced the road and served as part of a separation between the front yard and the back yard of the farm. The function makeMainBuilding() creates the building at one of five different locations: dead center, close to the left fence, hugging the left fence, close to the right fence or hugging the right fence. The main building has few possible locations to make the generation of the fence and secondary building more simple. These were used to separate front yard from back yard. The main building positions are illustrated in fig. 17.

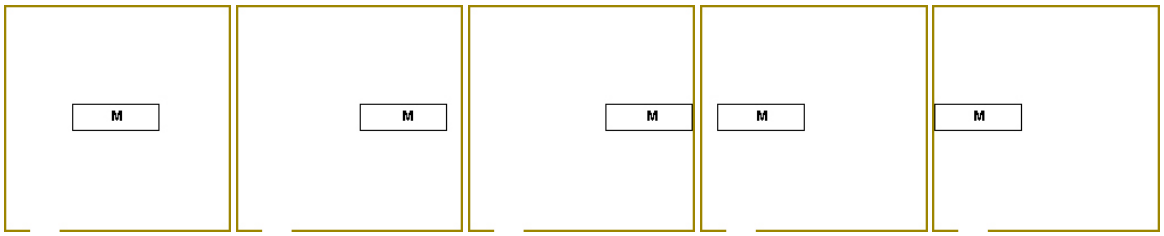


Fig. 17: Main Building positions

Farm Generation Step 3: Type 2 Buildings

In the Viking farms from the excavation there were several types of buildings with different purposes. One type of building was similar to the main building but smaller. These buildings were often seen in the front yard close to the fence surrounding the farm. In the game, two of these buildings are generated in the front yard and a possible third is generated as an element to further separate the front yard from the back yard as seen in some of the farms from the excavation.

The two buildings in the front yard are generated parallel to the fence or at a right angle outwards from the fence. The generation occurs at a random position along the fence of the front yard. The buildings proximity to the fence are varied by adding a small random value to the position of generation. An example of two generated buildings can be seen in fig. 18.

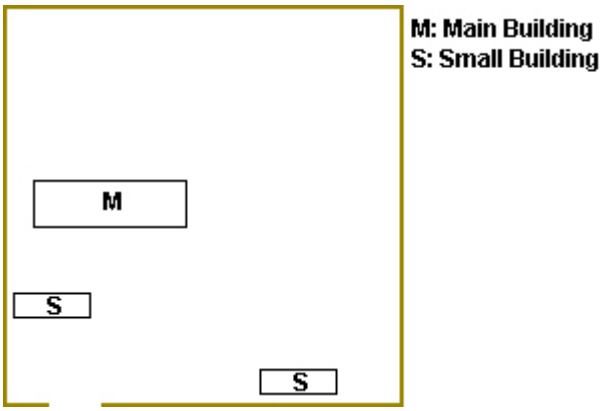


Fig. 18 Farm with Main and Small buildings

Each of the two buildings in the front yard is generated through a generate-and-test approach. The purpose is to check if a generated object is colliding with another object, and if so, replacing it with a new generation of the same object. The approach is used for several other objects being generated and can is illustrated in pseudo code in fig. 19:

```
list ObjectsToCheck = list of game objects
while ObjectsToCheck contains objects
            foreach object O in ObjectsToCheck
                        if O.collides is true
                                    remove O from ObjectsToCheck
                                    object newO = generate new object of type O
                                    add newO to ObjectsToCheck
                        else
                                    remove O from ObjectsToCheck
```

Fig. 19: Pseudo code for the generate-and-test method

The way new objects are generated varies from object to object but the process of checking all the generated objects for collision is the same. All the objects using this generate-and-test approach has a script that checks for collision and sets a public boolean to true if the object is colliding.

A third small building is created unless the Main Building is exactly in the center of the farm. This third small building will serve as a part of the separation of front yard and back yard. The building is created in the opposite side of the Main Building either parallel to the fence or at a right angle just as the other small buildings. An example of a building created at a right angle is shown in fig. 20.

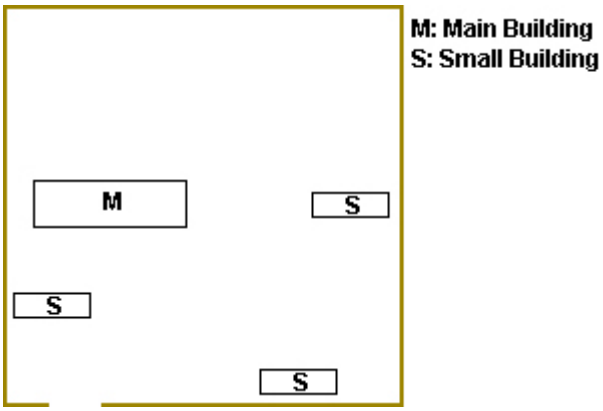


Fig. 20: The third small building have been generated

Farm Generation Step 4: Type 3 Building

In the excavation of the Viking farms a type of building were found, that were only used for work purpose. This building type was seen in the front yard of the farms and was smaller than the other buildings. The position of these buildings seemed to be more or less random, and therefore generation of this building type is random as well. A total of three work buildings are generated in the front yard of each farm. They are generated with the same generate-and-test approach as the small buildings, although their placement is spread over the entire front yard and not just along the fence. In fig. 21 the work buildings have been generated.

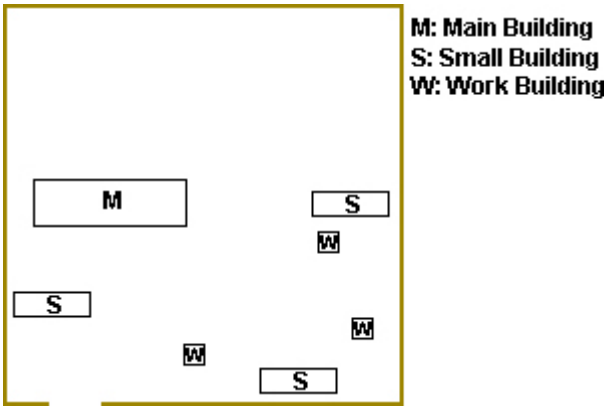


Fig. 21: Work Buildings have been generated

Farm Generation Step 5: Fence Inside the Farm

In many of the farms from the excavation, fences were used in combination with buildings to separate front yard from back yard. The fence in the game is generated based on the position of the Main Building and the third small building if existing. A large if/else structure manages the different cases of how the fence is placed, based on Main Building position and the third small building's position and rotation. The fence is

always generated in a way that will leave at least one opening into the back yard. The fence can be seen in fig. 22.

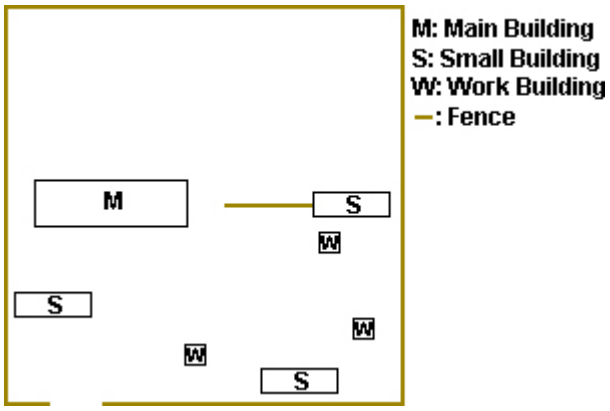


Fig. 22: Fence separating front yard from back yard have been generated

Farm Generation Step 6: Backyard Building

The back yards of Viking villages were mainly used for animals and storage and generally at least one building were present in the back yard. This building is generated with the generate-and-test approach with a location at a random position within the back yard area, and a rotation either parallel to or at a right angle with the Main Building. The building can be seen in fig. 23.

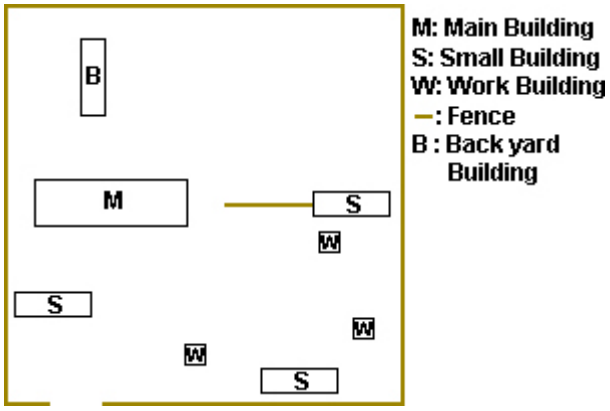


Fig. 23: Back Yard building have been generated

Farm Generation Step 7: Vegetation

The Viking farms were believed to possibly contain a few trees or bushes and other plants or herbs. The vegetation generation step will generate these objects and for the prototype game a single tree is created at a random position within the farm. The tree is generated with the generate-and-test approach used

previously for buildings. Other vegetation elements could have been easily created in the same way. The tree is often generated in the back yard as there, at this generation step, is more room as seen in fig. 24.

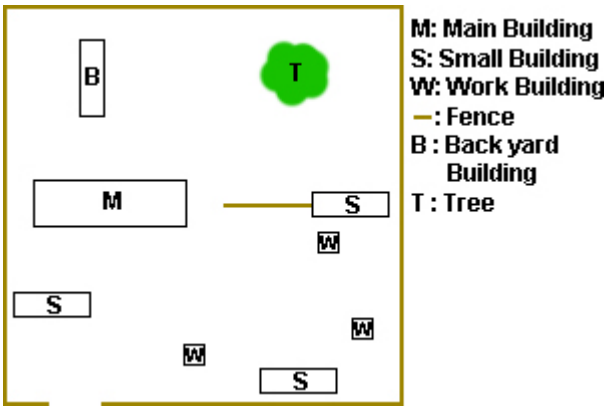


Fig. 24: A tree has been generated

Farm Generation Step 8: Front Yard Animals

So far, no knowledge elements have been generated inside the farm because buildings and trees are not directly connected to the area of knowledge provided in the prototype game. If, however, buildings and building style had been part of the area of knowledge, the buildings themselves could have been knowledge elements.

The Vikings were known to have several types of farm animals, one being chickens. The chickens are generated in the same way as many of the buildings and vegetation; with the generate-and-test method. The only difference between knowledge elements and normal elements is that the player can interact with knowledge elements. Knowledge elements are further described in section 3.3.

As chickens were fairly common, all farms being generated will have 3 to 6 chickens in the front yard. They are all generated at a random position with a random rotation. The generated chickens can be seen in fig. 25.

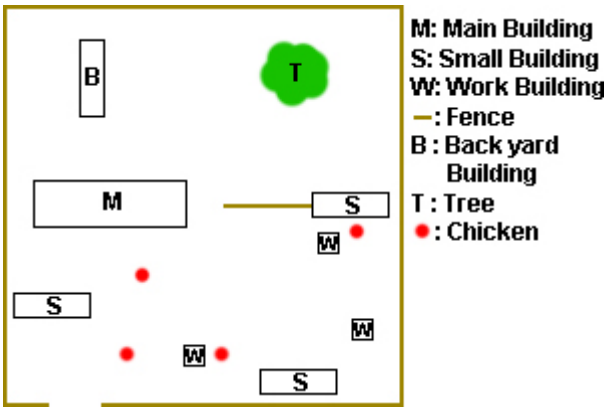


Fig. 25: Chickens have been generated

Farm Generation Step 9: Back Yard Animals

The back yard of the farms contains larger animals. These animals are determined by the type of farm; cow, sheep or pig. The generation of the different types of animals is the same, only the object generated is different. On each farm 6 - 10 animals of the farm type are generated in the back yard. The animals are generated within a smaller area of the back yard to simulate a flocking behavior. In the current version, the area is fixed to be in the right side of the back yard, but changing the area to a random position should not provide great difficulties. The animals are generated at a random position within the area and have a random rotation. They are generated with the generate-and-test approach and can be seen in fig. 26.

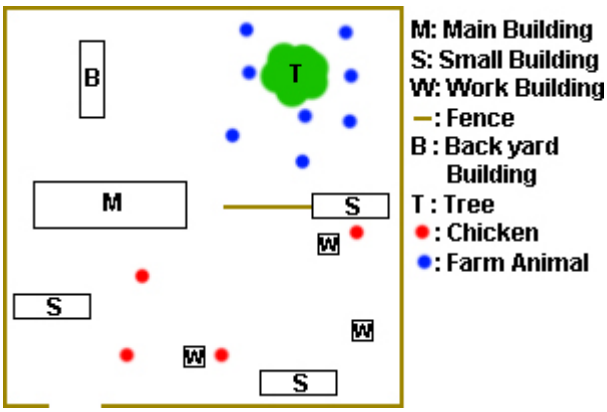


Fig. 26: Farm animals have been generated in the back yard

Farm Generation Step 10: Viking

To provide extra knowledge a Viking is created in the same area as the farm animals in the back yard. The Viking is a caretaker of the animals and provides information about them, therefore each type of farm animal has a Viking. The Viking is generated with the exact same approach as the animals and can be seen in fig. 27.

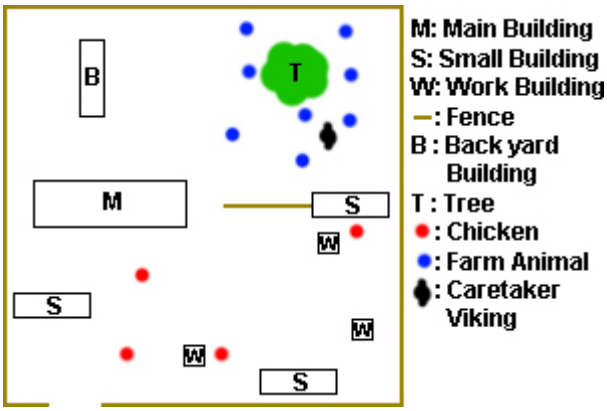


Fig. 27: The animal caretaker have been generated

3.3 Learning Content

In the prototype game there are 13 different knowledge elements, all generated through different processes in the procedural generation of the game. All the knowledge elements are using the same framework for interaction with the player, and providing knowledge. The knowledge elements are created as standard game objects, with the learning framework applied. Through this approach any game object can be turned into a knowledge element.

List of knowledge elements in the game:

Vikings:

- Fisherman
- Hunter
- Farmer
- Caretaker of the cows
- Caretaker of the sheep
- Caretaker of the pigs

Animals:

- Cow
- Sheep
- Pig
- Chicken

Other Objects:

- Berry bush
- Beehive
- Stack-Barn

The framework provides two ways of interacting with a knowledge element and receiving knowledge. One way is through direct conversation with NPCs (Vikings) and the other is through interaction with the two helpers. The helpers are Hugin and Munin, two ravens from the North Mythology (Sawyer, 2002). These ravens are servants of Odin, the god of wisdom.

All conversation with the helpers or Viking NPC's use the same conversation system. A Conversation class is used to build the actual conversation and manage the different steps of the conversation. A conversation is made up of sequences of text which the NPC or raven helpers will say and corresponding responses for the player. The progress of the conversation is stored in the Conversation class. Each knowledge element holds an instance of the Conversation class created for that specific element. When the player is interacting with a knowledge element a script, called HelperScript(), will run the conversation.

All the information being provided by the knowledge elements comes from the excavation used for the village generation or similar places (Sawyer, 2002; Hvass, 2011).

3.4 System Overview

In the final prototype game the player will spawn after the content generation has finished. The spawn position is on the edge of the game environment where the road to the village starts. When the player spawns, a conversation with the two helper ravens is activated to provide a few instructions. The player is told, that following the road will lead him to a village.

If following the road the player will be lead through the terrain to the village. On his way there he should pass a hunter, located close to a forest. The forests are scattered throughout the map and also contains berry bushes.

When the player makes it to the village he will find beehives, a stack-barn and a farmer outside the four farms. All four farms are accessible and contain various animals and Vikings. The road continues through the forest and will end at either the river or the ocean. Wherever the road ends, a fisherman will be waiting. An overview of a generated game environment can be seen in fig. 28 and the village in fig. 29.

Fig. 28: Overview of the game environment

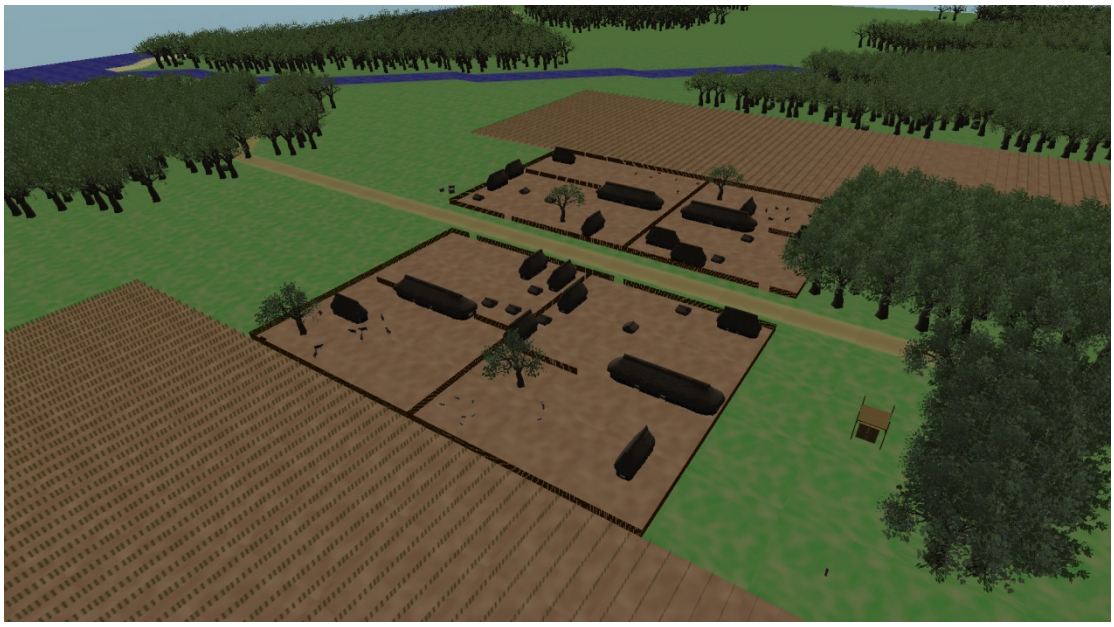


Fig. 29: Overview of the village

There are no in-game goals for the player so he can explore the environment as he sees fit. The player will be notified when 5 minutes has passed and again when 10 minutes has passed. These notifications are for testing purposes.

4 System evaluation

In the following sections the test procedure used to evaluate the prototype game is explained and the results of the test is put forward and used for conclusions. The questionnaires and test procedure can be found in the Appendix. The whole test procedure can also be seen online at www.megacraft.dk/VikingGame/VikingGame.html or on the provided CD.

4.1 Method

To validate specific aspects of the developed prototype game, a questionnaire based test was used. The primary purpose of the test was to validate the procedural content generation in general and to validate the distribution of knowledge elements in the game environment. Secondary purposes was to find indications that learning was occurring.

The procedurally generated content in the prototype game had to be placed properly in relation to other content, producing a strong semantic congruence in the game. Examples of poorly placed content could be a tree on top of a house or objects intersecting each other. The semantic congruence is an aspect of ensuring that the game environment is believable which is required for a good serious game. Furthermore, the semantic congruence also validates the actual algorithms used for generating the content.

Although a secondary objective, knowing if learning could be achieved through the prototype game was of some interest. The game was designed for learning purposes so any indication that the players were actually learning while playing the game could further validate the effectiveness of the system.

The test was conducted as an online survey, where the test participants had to play the game and answer questionnaires. It consisted of four main parts: two play sessions and two questionnaires. Prior to the actual test the test participants were informed about the project and its purpose through the introduction page. Here it was made clear that the focus of the test was on the effectiveness of the procedural content generation and that graphics and game-play elements were of no consequence for the testing purpose. The game itself was introduced along with the goal of the player. Important elements such as the helper ravens, interact-able objects, and the Vikings were likewise introduced to make sure that the test participants understood their purpose in the game. The last part of the introduction was an overview of the four steps in the test.

The first step of the test was playing the game for 10 minutes. The test participants were provided the required information to access the game in the browser through Unity Player. A link to a downloadable version of the game was also provided to ensure that all test participants could play the game even if the

browser version did not work. Before playing the game, the controls used to play were introduced, and the process of interacting with objects was explained. Furthermore all types of Vikings that could be encountered in the game were presented in such a way that the test participants would know them and hopefully look for them. The game opened in a new browser window so the test participants were asked to return and proceed to the next step of the test after playing the game.

When the test participants had played the game they would proceed to the second step of the test. This second step was the first questionnaire called Questionnaire One. The whole questionnaire can be seen in **Appendix A**. The test participants were required to provide age, gender, country of origin and a unique nick name that was used to link a test participants answers with the second questionnaire. Furthermore, the test participants were asked about their gaming experience and experience with serious games. All these preliminary questions were used to profile the test participants in order to better understand their answers.

The next group of questions was focused on the actual playing session. The amount of time spent on playing the game was important to know if the time spent met the requirements of the test. Furthermore the test participants were asked if any bugs or game crashes had occurred or if they had experienced general playing or reading difficulties within the game. These questions were important to identify the test participants premises for answering the rest of the questions and to validate the general stability of the system.

Questions regarding the placement of procedurally generated content were then asked. The test participants were asked how many objects they had encountered that seemed to be placed in a non-realistic fashion. If they had encountered any, they were asked to specify which type of object they had encountered as well as describe how they were placed. These questions were asked specifically to find out how often game objects were not correctly placed by the procedural content generation and what type of objects they were.

The last questions were regarding the knowledge elements in the game. The test participants were asked how many different knowledge elements they had encountered and if any were placed in a non-realistic fashion. They were also asked to provide a description of the knowledge elements found that were placed in a non-realistic fashion. The amount of different knowledge elements a test participant found was important in relation to how many was found to be non-realistically placed. In general, the questions regarding knowledge elements were created to identify problems in the procedural generation specific for

knowledge elements. Finally the test participants were asked if they had learned anything about Viking nutrition while playing the game and what previous knowledge they had in the area of Vikings.

As with the game, the questionnaire opened in a new window so test participants were again asked to return to the testing site and proceed to the next step after completing the questionnaire. The third step in the testing procedure was identical to the first step where test participants would play the game and the same information were provided. The only difference were that in the third step test participants were asked to play the game for 5 minutes instead of 10.

The fourth step in the testing procedure was the second questionnaire called Questionnaire Two. This questionnaire can be seen in **Appendix B**. In this questionnaire the test participants would answer the same questions as in Questionnaire One except the profiling questions, and whether they had learned anything about Viking nutrition. The answers were of course based on the second play session.

One of the main reasons for having the test participants playing the game twice was to double the amount of data regarding the general generation of objects and to identify bugs. Furthermore, results regarding discovered knowledge elements and acquired knowledge could indicate differences between the two play sessions. The test participants were asked to complete the whole test without taking long breaks between the steps to make the data of different test participants consistent.

4.2 Sample Description

The test was made available online in a 2 week period and invitations were sent to test participants. The invited test participants were both males and females in the age of 20-40 years. Since the game is in English as well as a prototype game requiring some understanding of normal gaming concepts, it is not suited for children although they could be a target group for such a game. The test participants were more or less ordinary people with no requirement to gaming experience as the intentions with the game was to enable education of a large variety of people in different areas.

In total 20 persons participated in the test. The results of one person was excluded as this person only completed half the test procedure and did not seem to have read any of the instructions. The actual test population was therefore 19 people.

The average age of the 19 test participants was 25,7 years with a standard deviation of 4,27 years. Four out of the 19 participants were female. 14 (73,7%) test participants reported that their country of origin was Denmark and the remaining test participants were from France, Italy and Canada. The amount of time test participants used on video games in general can be seen in fig. 30. The time test participants used on video

games was used to evaluate their general experience in the area. Test participants experience with serious games can be seen in fig. 31.

Fig. 30: The amount of time test participants use on video games

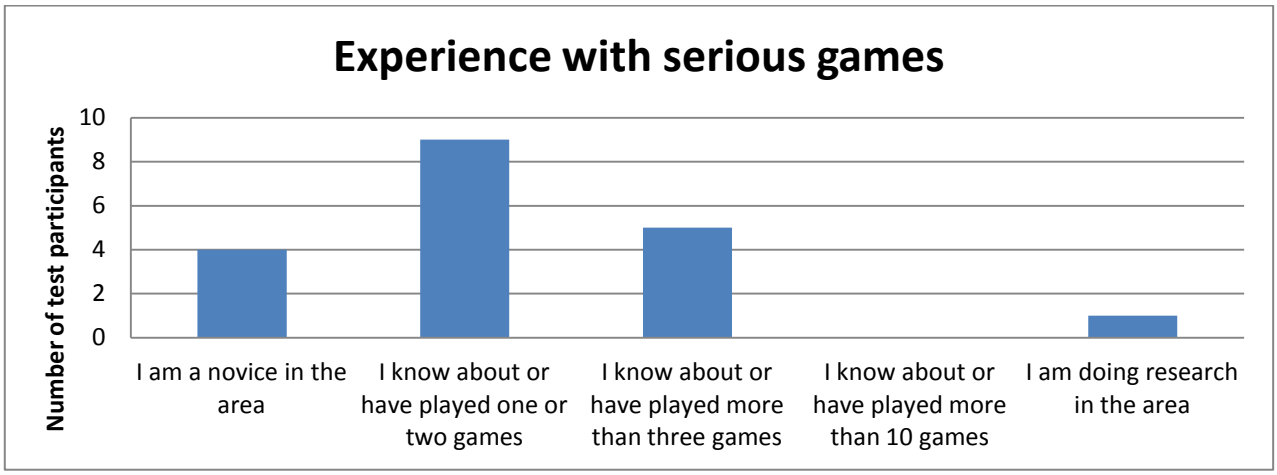


Fig. 31: Test participants experience with serious games

Test participants prior knowledge in the area of Vikings prior to playing the game is shown in fig. 32.

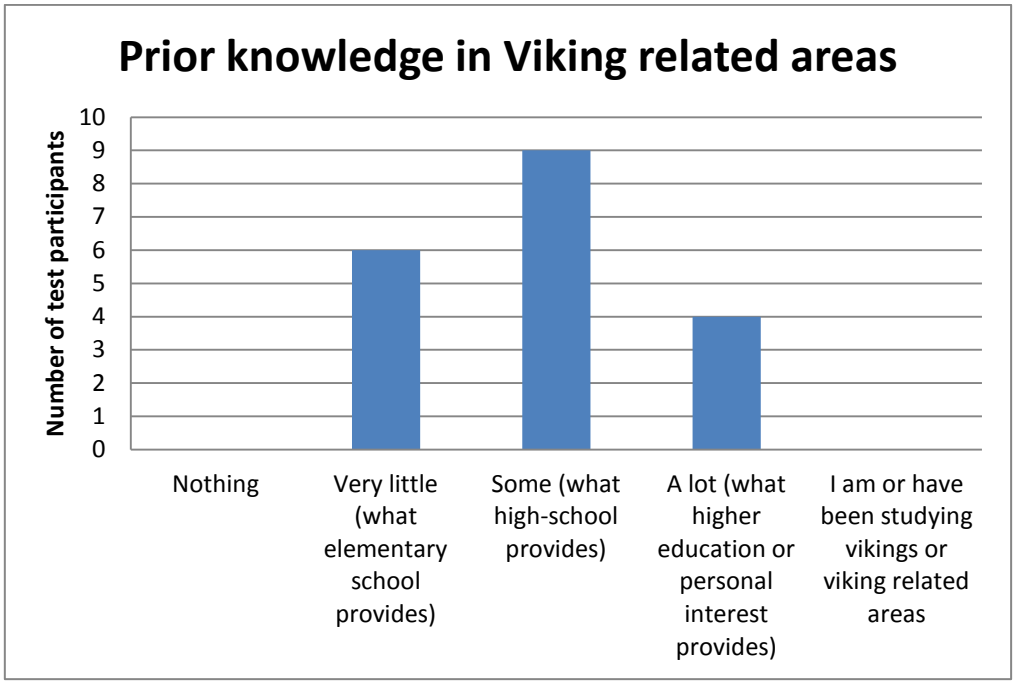


Fig. 32: Test participants knowledge in the area of Vikings prior to playing the game

4.3 Results and Discussion

Play Time

The results of the test were more consistent since most participants spent the same amount of time playing the game in the different playing sessions. All test participants reported that they played the game for about 10-15 minutes in the first play session, with the exception of one who played less than 10 minutes. In the second play session all participants played the game for about 5-10 minutes, with the exception of one who played more than 10 minutes. The amount of time participants spent playing the game matches the desired time.

Playing Difficulties

The good stability and playability of the system during participants play sessions were making the results of the test more valid. All test participants reported that they did not experience any playing difficulties, reading difficulties, crashes or bugs during the first play session. In the second play session, 18 of the 19 participants reported that they did not experience any playing difficulties, reading difficulties, crashes or bugs and one reported to have experienced playing difficulties, crashes and bugs.

Generation of Game Objects

Based on the low number of game objects test participants found to be placed in a non-realistic fashion, the general content generation is considered close to flawless. During the first play session, 18 of the 19 test participants reported no objects that were placed in a non-realistic fashion and during the second play session 15 reported no such misplaced objects. The amount of objects placed in a non-realistic fashion, that each participant found in the two play sessions, can be seen in fig. 33.

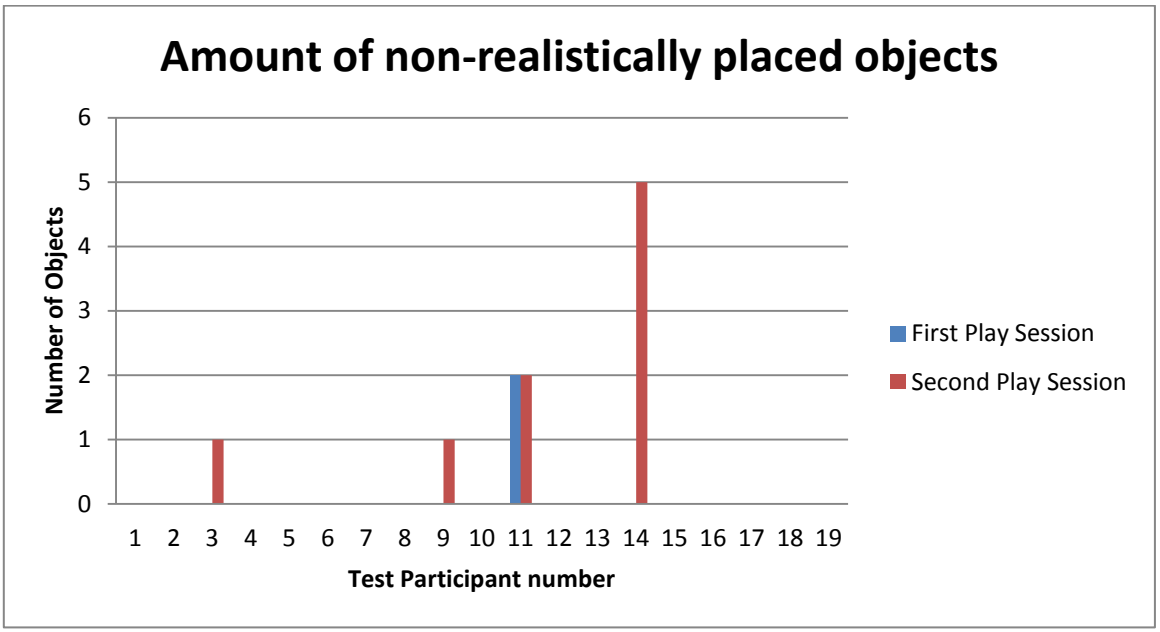


Fig. 33: The amount of non-realistically placed game objects found by test participants in first and second play session. 5 or more objects is rounded to 5

After the second play session test participant number 3 reported the following when asked for placement of objects in a non-realistic fashion:

> *" I went searching the woods for branches that "grew" into eachother as a result of models overlapping. I did find a few instances but it is only something one notices if one really looks for it."*

Other test participants also reported that the objects placed in a non-realistic fashion were trees or vegetation. The following was reported by test participant number 11 after the first playing session:

> *"There were a few trees that was on the road (in the side but stil on the brown part that marked the road)"*

And the following were reported by test participant number 11 after the second play session:

" 1 tree and 1 bush on the road"

The objects reported by test participant number 11 were trees and vegetation on the sides of the road but this is expected because of the nature of the road generation. When the road is smoothed, the tiles that are not actual road tiles, but has a part of a road texture in a corner, do not get their vegetation elements removed. This makes it possible for vegetation to appear in the side of the roads but not on the middle of the road.

The few trees with overlapping branches that test participant number 3 found, helps validate the generation of vegetation on individual tiles. It should be noted that test participant number 14 reported 5 or more objects to be placed in a non realistic way but did not report their type or placement. In general only trees and vegetation objects were reported to be placed in non-realistic fashion so the remaining elements in the system are properly generated every time the game is played.

Knowledge Elements

The placement of knowledge elements in the game is considered flawless based on the test results. All test participants reported that they had not found any knowledge elements that were placed in areas with no relation to the knowledge of the element. The amount of different knowledge elements found by test participants was high enough to properly validate whether they were positioned correctly. The precise number of different elements found by test participants in both play sessions can be seen in fig. 34.

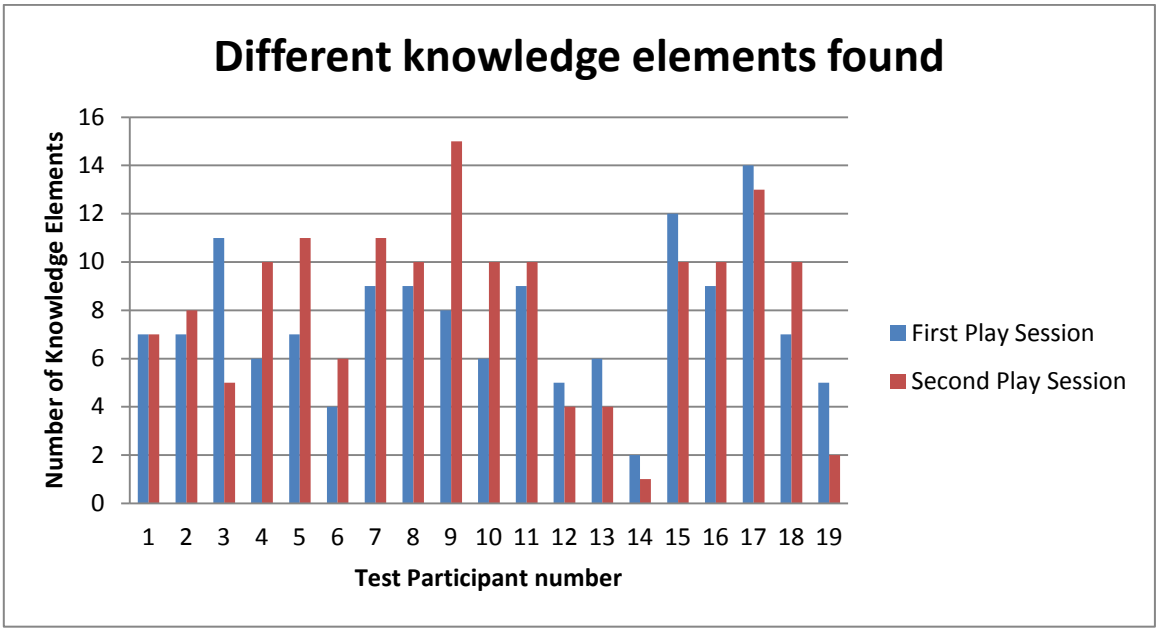


Fig. 34: The amount of different knowledge elements found by each test participant in first and second play session. 15 or more objects are rounded to 15. SD for the first Session was 2,78 and Mean was 7,53. SD for the second session was 3,63 and Mean was 8,26

The amount of discovered knowledge elements varied from the first play session to the second play session. Fig. 35 shows the increase or decrease in knowledge elements found for each test participant.

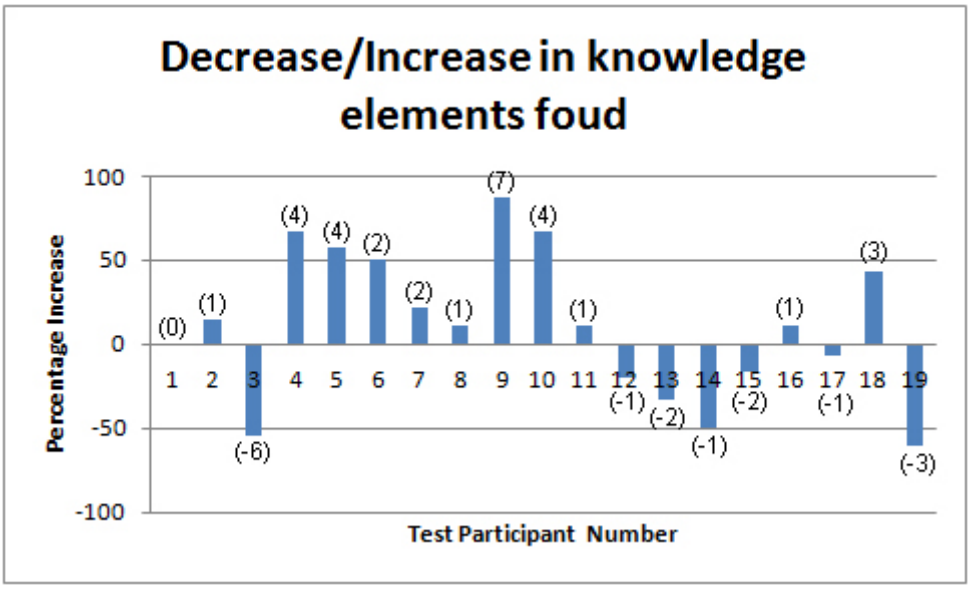


Fig. 35: Increase or decrease in knowledge elements found by test participants from playing session one to playing session two. The average increase was 10,47 percent. The value in brackets is the difference in actual elements

The average difference in elements found is a 10,47 percent increase. Seven participants had a decrease in knowledge elements found in the second play session and for test participant number 3 this is most likely due to the fact that he went looking for overlapping branches in the forest and thus did not look for

knowledge elements. For the remaining 6 participants the decrease might be caused by the shorter play time. It might also be caused by their lack of experience when playing video games. Fig. 36 shows the average decrease or increase in knowledge elements found based on test participants gaming experience.

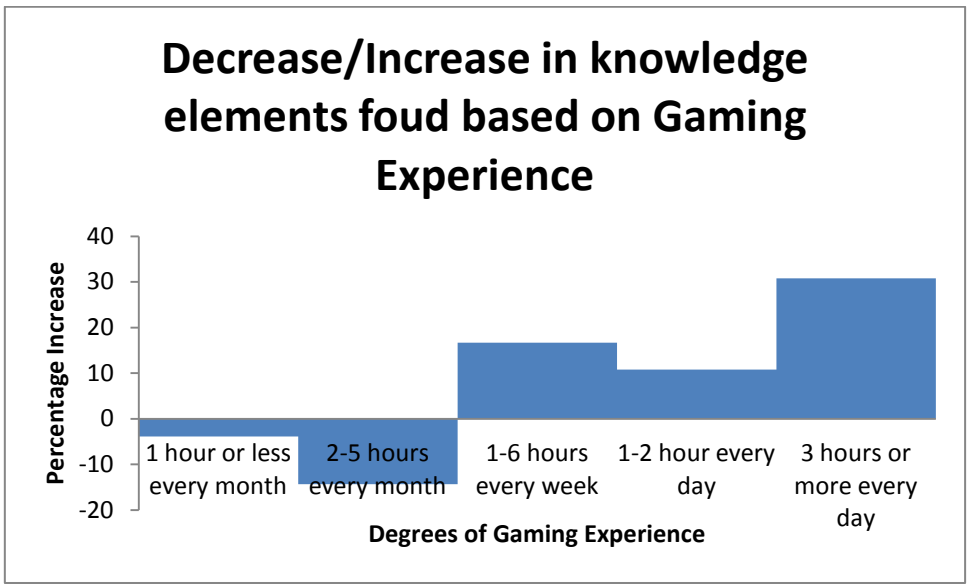


Fig. 36: Average Decrease or increase in knowledge elements found by test participants based on gaming experience

It is clear that test participants with a low gaming experience had a decrease in knowledge elements found whereas participants with high gaming experience had an increase. If test participants were not familiar with video games they might have a decreased interest in the game and they might have had more difficulty understanding the game than other test participants. Fig. 37 shows the average amount of knowledge elements found in relation to test participants gaming experience. It is evident that there is a tendency for more experienced players to find more knowledge elements.

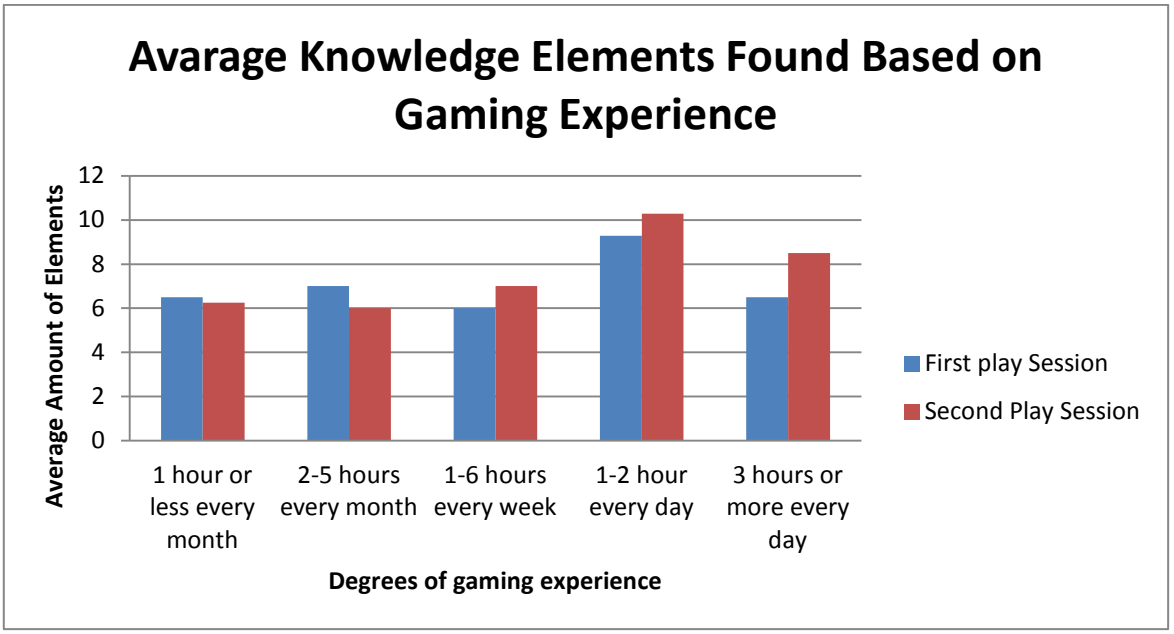


Fig. 37: Average amount of knowledge elements found in the two playing sessions based on test participants gaming experience

Learning

Only 4 participants out of a total of 19 reported that they did not learn something about Viking nutrition after the first play session. When asked about their prior knowledge about Vikings, two of the four who did not learn anything reported that they had "A lot (what higher education or personal interest provides)" and two reported they had "Some (what high-school provides)". These results indicate that the prototype game is capable of facilitating learning. The fact that learning is taking place can also be agued based on the general increase in knowledge elements found from play session one to play session two. Play session two was only half the time of play session one, but many participants still found more knowledge elements, indicating that they learned how the game works and used this knowledge in play session two.

5 Discussion

The general content generation was in most cases correct as the majority of test participants did not find any game objects that were placed in a non-realistic fashion while playing the game. The only game objects that test participants did report to be placed in a non-realistic fashion were trees and vegetation. Because of the road generation procedure, some trees were placed on the side of the road, but none on the middle of the road. In the real world a dirt road going through a forest would have to adapt to the trees in the forest and threes might grow onto the road. The trees that were reported to be placed on the side of the road are therefore of no significant consequence. The fact that those trees were reported to be placed in a non-realistic fashion, might be caused by the test participant thinking about the technology which is generating the environment, instead of thinking of the realism. an example of such was observed with the test participant who deliberately went looking for overlapping branches. This test participant were specifically looking for flaws in the content generation instead of finding the flaws based on real game-play. Flaws in the content generation can be acceptable if they have no influence on the experience during ordinary game-play, therefore the flaws in object generation in this game is considered acceptable.

Knowledge elements generated in the game were never reported to be placed in locations that had no relation to the knowledge it provided. This is most likely due to the high level of control over the generation of these elements. The level of detail in the game environment might also play a role in the good placement of knowledge elements. With so few details there is not many other objects do define an area than the actual knowledge elements. If the variety and amount of game objects in the village had been greater, there might have been a much higher chance for knowledge elements to appear improperly placed because each area would be much better defined.

A serious game must of course be capable of facilitating learning, however this was not the main focus in this prototype game. Still, based on the test results, it can be argued that the prototype game is capable of facilitating learning. Most test participants reported that they had actually learnt something new while playing the game. The reason for this might be the chosen area of knowledge. Viking nutrition is a very special topic and probably not the main topic in schools when learning about Vikings, so the level of knowledge that test participants had in this specific area could have been low. For example, if Viking warfare had been the topic, the amount of test participants who learnt something might have been lower. It is considered a success that so many test participants gained knowledge about Viking nutrition as the amount of knowledge was very limited.

That learning is taking place in the prototype game can also be argued based on the number of knowledge elements test participants found. Averagely, test participants found more knowledge elements in the second play session although this session was shorter than the first. It is possible that this is caused by test participants becoming familiar with the game concept in the first play session, and the applying this knowledge in the second play session. Although it is not knowledge on Viking nutrition within the domain of the game it is still learning.

It is a possibility that the system can be used as a skeleton for a full implementation of a procedurally generated serious game. The prototype game is created through simple procedural generation methods. With a game where the whole game world is procedurally generated, and the time and development recourses are limited, it was impossible to implement more advanced methods. The step based generation of different elements does, however, allow for improvements or replacements of individual parts of the content generation. Considering the nature of Unity3D, graphics can also be easily improved, and each game object can be individually updated.

6 Conclusion

Creating serious games where the game environment and knowledge elements are procedurally generated, is an area of research that is in no way fully explored. The idea of merging the two has been investigated through the creation of a prototype game where content were procedurally generated. The learning aspect of the prototype serious game was based on a model for Exploratory Learning. Although not fully implemented, the concept around the prototype game was designed to be applied to different knowledge domains. The concept and actual generation procedures for the game was explained, and the game was tested to primarily evaluate the strength of the content generation and secondarily to see if the game could facilitate learning.

A group of 19 test participants evaluated the game through an online test procedure. All test participants played the game twice to heighten the chance of finding wrongly placed objects. The test results proved that the content generation was working to great satisfaction in most areas. Only trees and vegetation were reported to be placed in a non-realistic fashion, and this only by a few participants. Knowledge elements were also procedurally generated, and none of the test participants reported any of those to be placed in a location out of context with the knowledge it provided. The actual placement of knowledge elements is therefore considered flawless. Based on these results it is evident that the game is capable of ensuring a strong semantic congruence.

Strong semantic congruence is important in generating a believable environment for learning, which is a requirement for serious games. Although the learning was not a main factor when evaluating the game, results from the test indicate that learning was occurring in two ways. Even though the second play session was shorter, many test participants found more knowledge elements here than in the first play session. This is an indication that players learn how the system works and applies this knowledge when playing a second time. Test results also showed that the majority of test participants actually gained some form of knowledge about Viking nutrition while playing the game. This indicates that the game is capable of facilitating learning through its well-implemented procedurally generated content.

7 Future work

Since the focus of the project primarily was on the actual generation of content in serious games, the learning potential of such games was not fully investigated. To conclude that procedurally generated serious games are equally good or better at providing knowledge than "normal" serious games, more research in this area is necessary. One of the first steps towards such conclusions would be developing a procedurally generated game where the actual learning is in focus. The method used in his project could act as a skeleton for such a game, but several areas would need improvement. The believability of the game environment is important in serious games, so graphics and game-play would need to be applied in a satisfying way. The game environment can also be improved by updating the generation algorithms. The terrain could be generated as 3D terrain with hills and valleys, and vegetation could be generated with algorithms that are more suited for creating believable vegetation.

The current game is not designed for a specific target audience and thus might be hard to master. For future work the game would need to be designed for and tested upon a specified target audience. This could probably be children in elementary school or even high school. It is important that the game and game-play is designed in such a way that the target audience will be drawn to the game.

References

Balasubramanian, Nathan; Wilson, Brent G.; (2005). Games and Simulations. ForeSITE, vol. 1

Blanchard, Emmanuel G.; Frasson, Claude.; Lajoie, Susanne P.; (2012). Learning With Games. In N. Seel (Ed.), Encyclopedia of the Sciences of Learning, New York: Springer, pp. 2019-2024

Bruner, Jerome S.; (1971). "The Process of Education" Revisited. The Phi Delta Kappan, Vol. 53, September, pp. 18-21

Caliskan, Hasan; (2012). Inquiry Learning. In N. Seel (Ed.), Encyclopedia of the Sciences of Learning, New York: Springer, pp. 1571-1574

de Freitas, Sara; Neumann, Tim; (2009). The use of 'exploratory learning' for supporting immersive learning in virtual environments. Computers & Education, Vol 52, pp. 343-352

Environmental Detectives; (2003). http://education.mit.edu/ar/ed.html, last accessed 18-05-2012

Gee, James Paul; (2005). What would a state of the art instructional video game look like?. Innovate, Vol 1

Gee, James Paul; (2008). “Learning and Games." The Ecology of Games: Connecting Youth, Games, and Learning. The MIT Press, pp. 21–40

Hazmat Hotzone; (2005). http://www.etc.cmu.edu/projects/hazmat_2005/, last accessed 18-05-2012

Hmelo-Silver, Cindy E.; Duncan, Ravit Golan; Chinn, Clark A.; (2007) Scaffolding and Achievement in Problem-Based and Inquiry Learning:A Response to Kirschner, Sweller, and Clark (2006). Educational Psychologist, Vol 42, pp. 99-107

Hvass, Steen; (2011). Bebyggelse og politik i Danmarks vikingetid - udgravningerne i Vorbasse. Natinalmuseets arbejdsmark 2011

Jonassen, David H.; Hung, Woei; (2012). Problem-Based Learning. In N. Seel (Ed.), Encyclopedia of the Sciences of Learning, New York: Springer, pp. 2687-2690

Ketelhut, Diane Jass; Nelson, Brian C.; Clarke Jody; Dede, Chris; (2010). A Multi-user Virtual Environment for Building Higher Order Inquiry Skills in Science. British Journal of Educational Technology, Vol. 41, pp. 56–68

Laurienti, Paul J.; Kraft, Robert A.; Maldjian, Joseph A.; Burdette, Jonathan H.; Wallace, Mark T.; (2004). Semantic congruence is a critical factor in multisensory behavioral performance. Experimental Brain Research, Vol. 158, pp. 405-414

Nareyek, Alexander; (2007). Game AI is Dead. Long Live Game AI!. Intelligent Systems, Vol. 22, pp. 9-11

Neber, Heinz; (2012). Discovery Learning. In N. Seel (Ed.), Encyclopedia of the Sciences of Learning, New York: Springer, pp. 1010-1013

Piaget, Jean; (1962). Play Dreams and Imitation. W. W. Norton and Company, Inc.

Prensky, Marc; (2001). Digital Game-Based Lerning. McGraw-Hill

Prensky, Marc; (2002). What Kids Learn That's POSITIVE From Playing Video Games.

Rieber, Lloyd P.; (2005). The Cambridge Handbook of Multimedia Learning. Cambridge university Press, pp. 549-567

River City; (2009). http://muve.gse.harvard.edu/rivercityproject/, last accessed 18-05-2012

Roden, Timothy; Parberry, Ian; (2004). From Artistry to Automation: A Structured Methodology for Procedural Content Creation. Entertainment Computing – ICEC 2004: Lecture Notes in Computer Science, Springer

Royle, Karl; (2008). Game-Based Learning: A different perspective. Innovate, Vol 4

Shaffer, David Williamson; (2004). Pedagogical Praxis: The Professions as Models for Postindustrial Education. Teachers College Record, Vol. 106, pp. 1401–1421

Sawyer, Peter; (2002). Gyldendal og Politikkens Danmarkshistorie. Vol. 3: Da Danmark Blev Danmark. 2nd edition

Smith, Gillian; Gan, Elaine; Othenin-Girard, Alexei; Whitehead, Jim; (2011). PCG-Based Game Design: Enabling New Play Experiences through Procedural Content Generation. Proceeding PCGames 2011, No. 7

Sorenson, Nathan; Pasquier, Philippe; (2010). Towards a Generic Framework for Automated Video Game Level Creation. EvoApplications, part I, pp. 131-140

Squire, Kurt D.; (2005). Changing the Game: What Happens When Video Games Enter the Classroom?. Innovate, Journal of Online Education, Vol. 1

Squire, Kurt D.; (2007). Games, Learning, and Society: Building a Field. Educational technology, September-October

Togelius, Julian; Yannakakis, Georgios N.; Stanley, Kenneth O.; Browne, Cameron; (2010). Search-Based Procedural Content Generation. Applications of Evolutionary Computation: Lecture Notes in Computer Science, Springer

Unity3D; (2012). http://unity3d.com/support/documentation/, last accessed 18-05-2012

Van Eck, Richard; (2006) Digital Game-Based Learning: It's Not Just the Digital Natives Who Are Restless..... EDUCAUSE Review, vol. 41

West, Mick; (2008). Random Scattering: Creating Realistic Landscapes. Gamasutra, http://www.gamasutra.com/view/feature/1648/random_scattering_creating_.php?page=1, last accessed 18-05-2012

Appendix A:

Questionnaire One

The Viking Game Questionnaire 1

This questionnaire should be answered after playing The Viking Game once.
* Required

Questionnaire 1

Fill out this part after playing the game for the first time.

Age *

Gender *

- Male
- Female

Nickname *
Please provide a unique nickname that is used to identify your answers here with the next questionnaire. IMPORTANT: Remember the nickname for the next questionnaire.

Country of origin *
Please help determine if you come from a viking related culture by providing your country of origin.

- Denmark
- Norway
- Sweeden
- Iceland
- Other:

Gaming Experience *
How much time do you averagely spend on playing video games ?

- 1 hour or less every month
- 2-5 hours every month
- 1-6 hours every week
- 1-2 hour every day
- 3 hours or more every day

Experience with Learning Games *
How much experience do you have with games whos purpose is to teach the player something specific ?

- I am a novice in the area

- I know about or have played one or two games
- I know about or have played more than three games
- I know about or have played more than 10 games
- I am doing research in the area

Play Time *
How many minutes have you spent playing The Viking Game before answering this question ?

- Less than 10 minutes
- About 10-15 minutes
- More than 20 minutes

Playing difficulties *
Did you experience any problems controlling the character or interacting with objects throughout the game ?

- Yes
- No

Reading difficulties *
Did you have trouble reading or understanding text in the game ?

- Yes
- No

Crashes and bugs *
Did the game crash at any point or did you notice some bugs that seemed to interfere with your gaming experience ?

- Yes
- No

Game Objects 1 *
How many objects(eg. tree, bush, cow, person, fence) did you see in the game that were placed in a non-realistic fashion ? (e.g. A cow on top of a house, a tree in the ocean or river, animals intersecting one another)

- 0
- 1
- 2
- 3
- 4
- More than 5
- More than 10

Game Objects 2
If any, which of these objects did you see in the game that were placed in a non-realistic fashion ?

- [ ] Trees and plants
- [ ] Animals
- [ ] People
- [ ] Buildings and fences
- [ ] Other

Game Objects 3
Could you describe situations where some of the game objects were placed in a non-realistic fashion ?

Learning Objects 1 *
How many different objects that provided you with knowledge(e.g. vikings, animals) did you encounter in the game ?

1

Learning Objects 2 *
Considering the objects in the game that provided you with knowledge(e.g. vikings, animals), did any of these objects appear to be placed in an area that had nothing to do with the provided knowledge ?

- ( ) Yes
- ( ) No

Learning Objects 3
Could you describe situations where an object that provided you with knowledge appeared to be placed in an area that had nothing to do with the provided knowledge ?

Learning 1 *
How much did you know about vikings before you played this game ?

- ( ) Nothing
- ( ) Very little (what elementary school provides)
- ( ) Some (what high-school provides)

- A lot (what higher education or personal interest provides)
- I am or have been studying vikings or viking related areas

Learning 2 *

Did you learn something about vikings nutrition while playing this game ?

- Yes
- No

Submit

Appendix B:

Questionnaire Two

The Viking Game Questionnaire 2

This questionnaire should be answered after playing The Viking Game two times.
* Required

Nickname *
Please provide the nickname you used for the first questionnaire.

Play time, second play through *
How many minutes have you spend playing The Viking Game on your SECOND play through ?

- Less than 5 minutes
- About 5-10minutes
- More than 10 minutes
- More than 20 minutes

Playing difficulties *
Did you experience any problems controlling the character or interacting with objects throughout the game ?

- Yes
- No

Reading difficulties *
Did you have trouble reading or understanding text in the game ?

- Yes
- No

Crashes and bugs *
Did the game crash at any point or did you notice some bugs that seemed to interfere with your gaming experience ?

- Yes
- No

Game Objects 1 *
How many objects(eg. tree, bush, cow, person, fence) did you see in the game that were placed in a non-realistic fashion ? (e.g. A cow on top of a house, a tree in the ocean or river, animals intersecting one another)

- 0
- 1
- 2
- 3
- 4
- More than 5

- ○ More than 10

Game Objects 2
If any, which of these objects did you see in the game that were placed in a non-realistic fashion ?

- ☐ Trees and plants
- ☐ Animals
- ☐ People
- ☐ Buildings and fences
- ☐ Other

Game Objects 3
Could you describe situations where some of the game objects were placed in a non-realistic fashion ?

Learning Objects 1 *
How many different objects that provided you with knowledge(e.g. vikings, animals) did you encounter in the game ?

1

Learning Objects 2 *
Considering the objects in the game that provided you with knowledge(e.g. vikings, animals), did any of these objects appear to be placed in an area that had nothing to do with the provided knowledge ?

- ○ Yes
- ○ No

Learning Objects 3
Could you describe situations where an object that provided you with knowledge appeared to be placed in an area that had nothing to do with the provided knowledge ?

Submit

Powered by Google Docs

Appendix C:

Testing procedure

Intro

The Project

The Viking Game is a simple game created for my Medialogy master's thesis about educational games and procedural content generation. Educational games is games that are created to provide the player with some form of knowledge(in this case history) while playing. They are normally created in a static way where the experience for players is always the same. In the master's thesis I attempt to create a system that will generate the game environment and learning content in a random way, making the game experience different every time. The game is a proof of concept with the purpose of testing the content generation the game is built upon, therefore it is limited in graphical and game-play elements.

The Game

In the game you will find yourself in Denmark at the time of the Vikings. Your mission is to explore the game environment and learn as much as you can about the Vikings nutrition. In the game you can interact with certain objects, animals and Vikings. There are several Vikings in the village and a few out in the open environment. To help you the two ravens Hugin and Munin have been sent. They and the Vikings you encounter will provide information about the Vikings nutrition.

The Test

The test is divided in the following four steps:

1. Playing the game (min. 10 minutes)
2. Answering a questionnaire
3. Playing the game again (min. 5 minutes)
4. Answering a questionnaire

I ask that you take all the steps without long breaks. To begin the test press Begin.

Step 1

In this first step you will be playing the game for at least 10 minutes. The game will open in a new window and after you have played the game for 10 minutes come back here and go to step 2.

The game requires Unity Web Player to run, if it does not automaticly direct you, download it here: Unity Web Player.

Game controls:

To move around use the Arrow Keys or W,A,S,D

To look around use the Mouse

To sprint hold down Shift

To interact with objects press E

To continue in a conversation press 1

If an object in the game has an exclamation mark (!) floating above it, you can interact with this object.

You can also interact with the Vikings you encounter, there are six different vikings: fisherman, hunter, farmer, shepherd, a caretaker of the pigs and a caretaker of the cows.

Play

Play the game in your browser with the following link and close the game window when you are finished. If the game does not start: try refreshing the game window(F5):

Play the game in your browser

If you have problems playing the game in your browser the game can be downloaded here.

Step 2

Please fill out this questionnaire(opens in a new window), then come back here and go to step 3:

Questionnaire

Step 3

In this third step you will be playing the game for at least 5 minutes. The game will open in a new window and after you have played the game for 5 minutes come back here and go to step 4.

The game requires Unity Web Player to run, if it does not automaticly direct you, download it here: Unity Web Player.

Game controls:

To move around use the Arrow Keys or W,A,S,D

To look around use the Mouse

To sprint hold down Shift

To interact with objects press E

To continue in a conversation press 1

If an object in the game has an exclamation mark (!) floating above it, you can interact with this object.

You can also interact with the Vikings you encounter, there are six different vikings: fisherman, hunter, farmer, shepherd, a caretaker of the pigs and a caretaker of the cows.

Play

Play the game in your browser with the following link. If the game does not start: try refreshing the game window(F5):

Play the game in your browser

If you have problems playing the game in your browser the game can be downloaded here.

Step 4

Please fill out this questionnaire(opens in a new window), then come back here to finish:

Questionnaire

The End

Thank you very much for helping test this game. If you are interested in more information about my master's thesis or my education please contact me at Blacex@gmail.com or visit Medialogy Copenhagen.