Using Serious Games to Teach Cause Effect Relationships in Asphalt Quality

---

By Peter Verzijl s1317369 University of Twente EWI Creative Technology

Supervisor Job Zwiers **2nd** Robby van Delden **Client** Janine Profijt

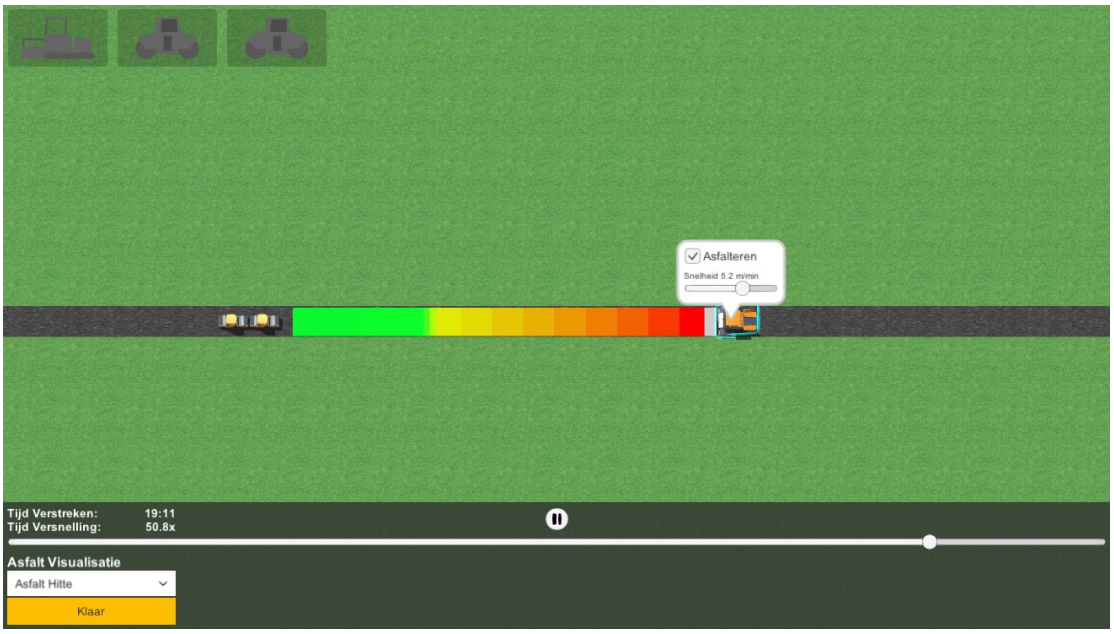

Figure 1: Asphalt Pave Simulator '17 Screenshot

Introduction

In 2018 the minister of OCW, Education Culture and Science (Onderwijs Cultuur en Wetenschap in Dutch), plans to make minor like modules mandatory for all MBO (Secondary vocational education, Middelbaar Beroepsonderwijs in Dutch) education [1]. Minors, which are similar to electives, where students spend half a semester deepening their knowledge in the chosen subject. The University of Twente and the department of Civil Engineering are developing such a minor for infrastructure education. This minor will deepen the students' knowledge on asphalt road construction and the emerging technologies that are developed in the field. This was needed as the study material did no longer reflect the state of the industry, as stated by two teachers from both the ROC of Twente (Regional Education Center, Regionaal Opleidingen Centrum in Dutch) and SOMA College (Foundation for the Education of Machinists for Contractor Companies, Stichting tot Opleidingen van Machinisten voor Aannemersbedrijven in Dutch). Janine Profijt develops this minor in cooperation with the construction conglomerate ASPARi. Janine is also the client supervisor for this project.

The minor consists of an introduction, five assignments and an evaluation in the form of a presentation and a portfolio. In the third assignment of the minor, a serious game is played by the student in order for them to better understand the cause effect relationships in asphalt quality. The learning objective of the game is to prepare the player/learner to make real time decisions in the field based on the data that emerging technologies bring. These technologies include: real time weather data, real time heat maps from thermal cameras attached to waltzes and asphalt spreader, location data from both machines and microwave density sensors. The student needs to be able to prepare for weather conditions and make real time decisions when these conditions change to preserve the quality of the asphalt.

The design serious game and its development is the objective of this project. Therefore, the research question is stated as:

> How can an effective serious game be developed for teaching students to make better decisions in both the planning phase as well as during the construction work, based on both real-time data and environmental factors?

From this main research question, two sub-questions need to be answered before the main question can be resolved:

> How could the cause effect relationships between external factors and the quality of the asphalt be best taught by means of this serious game?
>
> What type of interactions with the simulation are best suited to train students to make better decisions based on real-time weather, temperature and geolocation data in the real word?

In the next chapter, the state of the art on the creation of serious games will be handled as well as a short introduction to asphalt quality. After the state of the art, a methodology for designing and development of the game and a method for testing the

serious game will be discussed. The results of the test and a discussion on the resulting serious game will be discussed in the section thereafter. Finally, a discussion on the results will follow, which includes recommendations for future work, limitations and a figure and reference list.

Table of Contents

State of the Art

Existing Serious- / Simulation Games

Graphical Simulations

XPactor

In 2005, Turkiyyah et al. [2] released a paper on virtual training tools for transportation infrastructure construction. This paper describes the development and need for cost-effective 3D (three-dimensional) training environments for hot mix asphalt paving called XPactor. The software allows for multiple users with different roles to work together on a single virtual construction site. It also allows for multiple scenarios like different geometries, multi-lane or inclined roads, weather conditions, time of day and traffic patterns. These parameters can be set at runtime. They stated that future simulations should stress the importance of planning and timing of a broader range of paving tasks and implement artificial intelligence (AI) to allow users to focus on timing and coordination of tasks rather than execution of the paving process.

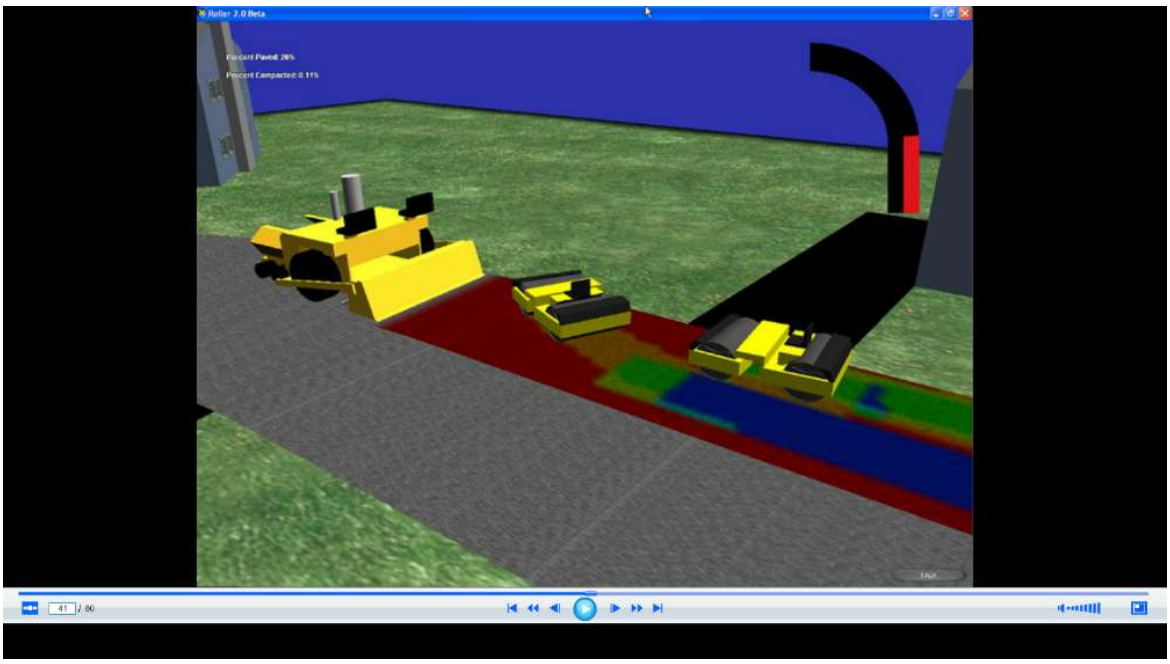

Figure 2: XPactor

A *presentation in the same year, done by Mahoney* [3], one of the contributors on the XPactor project also mentions the importance of heat map usage and mentions interactive pavement guides. He also includes a list of websites that describe guides on hot asphalt paving. Mahoney also stresses that the demands for data are higher due to the following factors: better decision-making, long-term planning, modeling, risk assessment and standards and accountability measures.

In 2011, Merge games created the simulation game Road Construction Simulator that simulates being a road construction worker. The simulation is a very simplified simulation of what road construction entails. It consists of the following activities: driving delivery vehicles and positioning road signs, laying asphalt by driving a truck filled with HMA (hot mix asphalt), paving the asphalt by driving a roller over the asphalt once to complete the asphalt, stripping old asphalt, drilling away cracks in asphalt and putting down line markings. Asphalt temperature and waltzing strategies nor the types of asphalt are simulated.

Figure 3: Road Construction Simulator - Screenhost 1

Figure 4: Road Construction Simulator - Screenhost 2

Construction Simulator 2

Figure 5: Construction Simulator 2 - Screenshot

The second version of Construction Simulator will be released in the fourth quarter of 2016. In includes a more realistic representation of HMA paving. As can be shown in the video (link: https://youtu.be/fJDt5DmG4oo?t=8m45s), it includes realistic CAT (Caterpillar Inc.) models of pavers, milling machines and rollers. Although no heat

simulation is done and the thickness of the asphalt is not simulated, it provides a reasonable simulation of how the machines move around while working.

University of Twente Simulations

In august of 2013, two simulations of the paving process were made at the University of Twente in the faculty of Civil Engineering at the department of Infrastructure. In one simulation (figure 5), a paver moves with a certain speed along the blue path that can be seen in the image. The roller can be moved with the left mouse button, the path that the roller has traveled can be seen as the red line on the image. The red dot is the desired new destination given by the user by pressing the left mouse button.

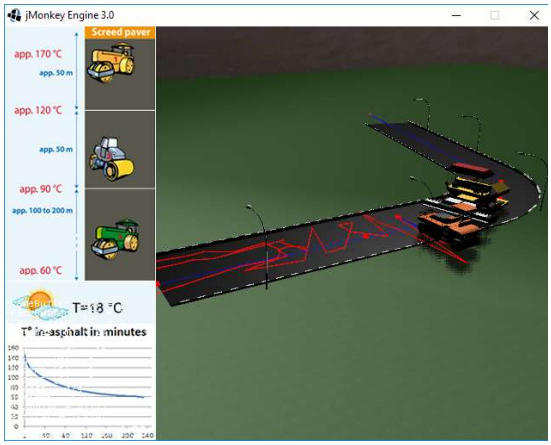

Figure 6: Roller simulation University of Twente

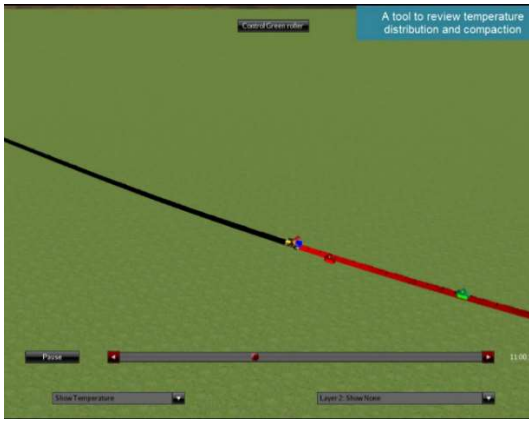

Figure 7: Roller compaction and temperature simulator

The right image (figure 6) is a frame taken from a video [4] that was made of the second prototype that was created after the one shown in figure 5. This shows an asphalt paver and two rollers that both autonomously and in the case of the roller, user controlled move around. The temperature and the compaction can both be shown as red to black colors (red is hot, and black is cold) on the road. The black to green colors indicate compaction (green is sufficiently compact, black is not compacted). By controlling the roller, the effect of waltz strategies on the compaction can be shown.

Non Graphical Simulations

CCT international has an asphalt-paving simulator [5] that uses a non-graphical representation in the way that the other simulations have created. The tool lets a user create an abstract model of the asphalt road construction. The interface is divided in three sections: input, process logic and animation & output.

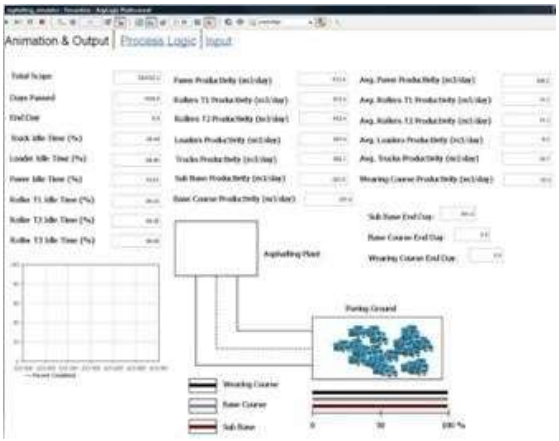

Figure 8: Animation & Output

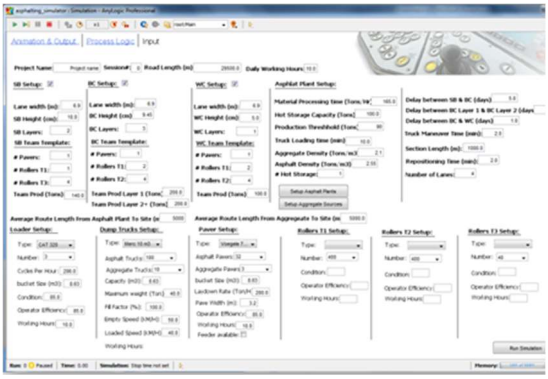

Figure 9: Input

In the input section, the user is able to select between different real life pavers, rollers, loaders and dump trucks and chose how many of each are used in the construction. Each have six or more settings. For example, the paver has settings for: the amount of asphalt pavers, aggregate pavers, asphalt containment capacity, pave width, operator efficiency and working hours. Besides setting up the machinery, the user also specifies the wearing course, base course and sub-base course layers that are to be paved, including lane width, layer height and the amount of layers.

The process logic section describes the mathematical model that is based on input and output nodes where the flow of data can be controlled. All three layers are modeled separately. The animation and output panel shows the resulting information based on the model and input. Here construction costs, asphalt plant requirements, equipment function and other factors are described. From this data, potential bottlenecks can be determined and acted upon in the planning phase of asphalt road construction.

Serious Game Technologies

There are many different technologies with which you could create serious games, with equal variety of control, ease of use and capabilities. The most used game technologies and some technology used by the mentioned existing simulations are also mentioned.

Unity 3D

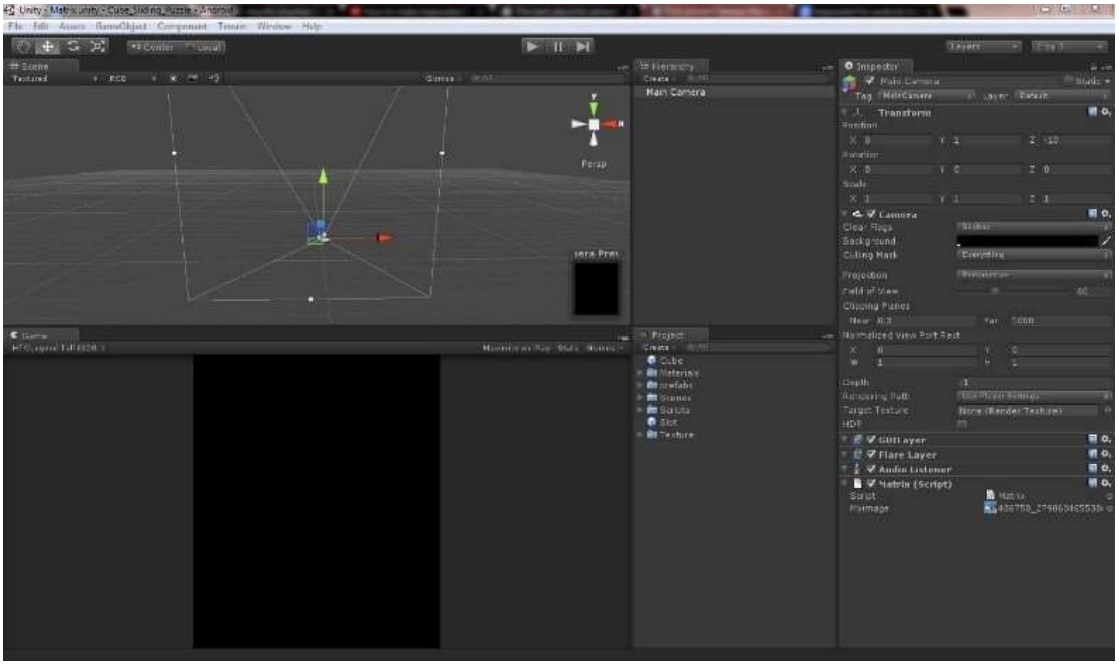

Figure 10: Unity Editor

Unity 3D, or now commonly referred to as Unity is a game engine combined with various editors that is used a lot in the independent- and mobile game industry [6]. This is because created games can be easily ported to a wide variety of platforms such as: Windows, Mac, Linux, Android, IOS, Tizen OS, PS4, Xbox One, PSVita, WebGL, Nintendo Switch, Wii U, 3DS, VR (virtual reality) platforms and Facebook [7]. It comes with a multitude of tools [8] to help design such as a level editor, animation editor, state machines, lights, multiple rendering engines, 2D tools, audio mixers, path finding and much more. It also has an API for creating your own editor windows and scripts for game behavior in Mono C#. It is also able to use compiled DLL files from other languages.

Unreal Engine

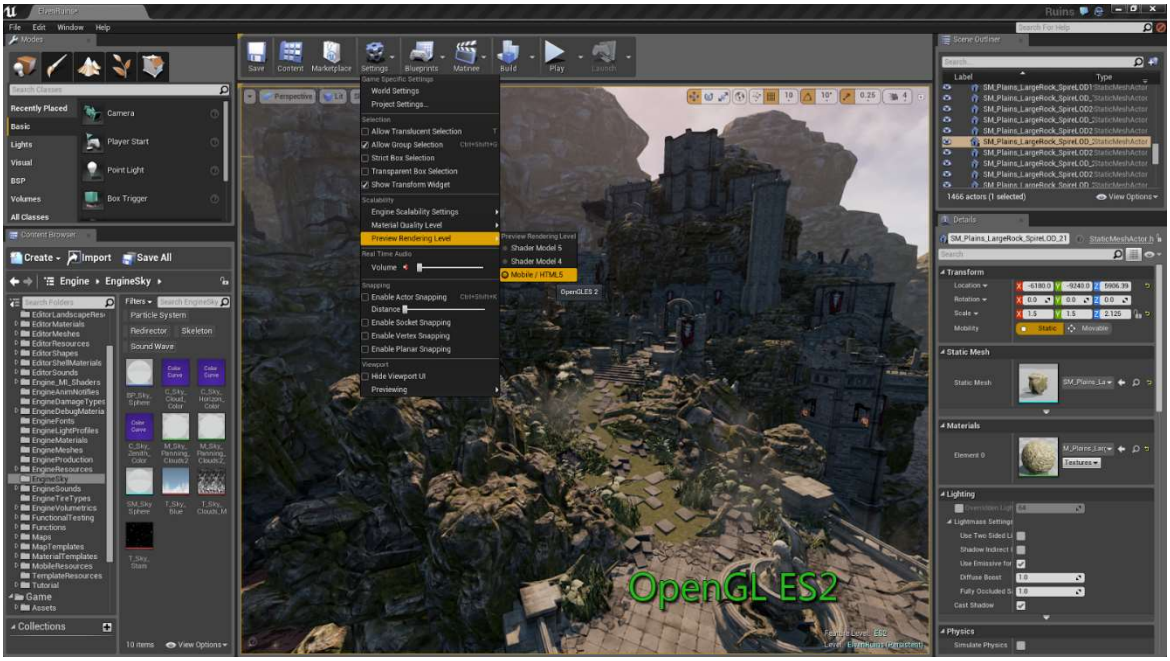

Figure 11: Unreal Editor

Unreal Engine is the main competitor of Unity when it comes to graphical fidelity and the amount of commercial console and triple-A game that are released. It is also a lot older than Unity (Unity was released in 2005 for Mac only and Unreal Engine was first released in 1998). A big advantage of Unreal to Unity is that it uses the industry standard language C++ instead of an older Mono C# version, which is still the standard, even for proprietary game engines. Besides this advantage, the whole engine has recently been published as open source. Which means you can change every single bit of the engine code to your liking, which is impossible on Unity since you are stuck with binaries and thus is less modifiable.

It also supports a visual coding interface called blueprint where you can drag behavior boxes and lines between boxes to do the coding for you. It also implements many of the features that Unity also has [9]. But also some additional features such as: changing code while the game is running, recoding gameplay and replaying it with different code, a VR editor, new Vulcan graphics API (application programing interface) and a node based material editor.

Both Unity and Unreal Engine have been used in the past to create serious games in various fields such as tactical training for soldiers, explaining fracking and medical training [10, 11, 12, 13].

Java LWJGL, jMonkey and other Java engines

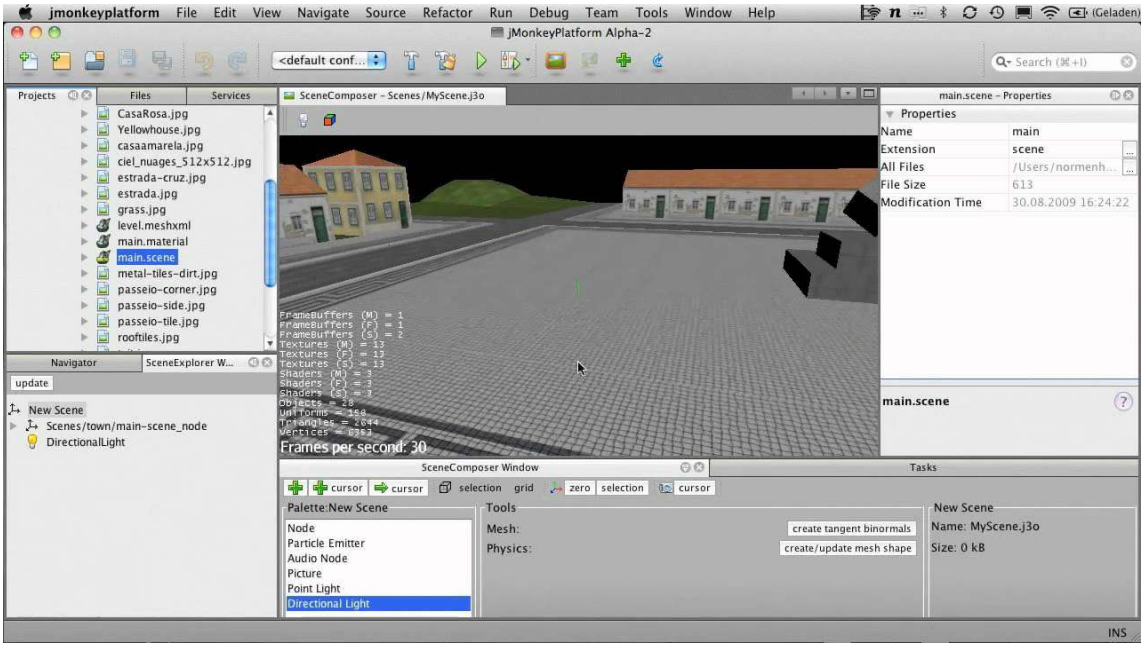

Figure 12: jMokey Editor

Java is quite an often used programming language in the computer science and business circles due to its many uses and object oriented nature. Many programmers in the field use Java and thus the pool of programmers is very large. It is also argued that Java is easier to learn because it does not allow you to directly modify memory and use pointers, such as C and C++, also called garbage collected. Therefore, there are not many triple-A games released due to the lack of control over the memory usage of the game and the fact that you can get very irregular frame rates when the garbage collector does interfere. However, this is only a problem when dealing with high fidelity games. Unless that is a requirement for the intended serious game, Java could, and has been used, in serious games. For example, the paving and roller simulator built by the University of Twente [4] is using jMonkey.

JMonkey is a game engine for Java with a scene, properties and navigator editor build in and looks familiar to the interface that Unity uses.

Serous Game Creation Methodologies

Catelano et al. [14] created guidelines for the creation of serious games trough pedagogically driven design based on five pedagogical theories, Bloom's Taxonomy, Kirkpatrick's Four-Level Training Evaluation Model, constructivism, experimental theory of learning and Nonaka's SECI model. They argued that simulation games are well suited to teach players the consequences of their decisions. Which is the goal of this project. They also argued that meaningful context helps learners to foster recall and knowledge application to real life easier. Facilitating the transferal of in-game skills to real life skills. Something that is noted as not trivial by Garris et al. [11] This contextualization is strengthened by including all relevant stakeholders in the design [14].

Catalano et al. [14] reinforces this idea that engagement can be increased by challenge through keeping the player on the flow curve. A concept introduced by Csíkszentmihályi [15]. They also state that player reflection on their choices is important to learning, this is reinforced by Harteveld et al. [16] and Catalano et al. [14] whom introduce the KWS method which describes three thought processes by the player: what I Know; what I want/Need to know; and what I have Solved. The serious game should provide these tools to make sure the player wants to learn new information in order to get better at the game and be able to progress. They also argue that choices in serious games should avoid ending in a success or fail state. They argue that this will hamper improvement in the player and promotes a guessing game instead of coming to meaningful conclusions.

Harteveld et al. [16] created a design philosophy for serious games that states that serious games design needs to balance three characteristics: play, meaning and reality. The effectiveness of the serious game relates strongly to how well these three pillars are in balance. It is the task of the game designer to make decisions based on the design dilemmas and trilemmas that are present in the design space. Play embodies the play and fun side of the game, including all the game design and technology behind the game. Meaning emphasizes the learning content in the game, the pedagogical theories and transferal of knowledge. Reality represents the closeness to the real life scenario's that the game is trying to prepare the player/learner for, such that he can make better decisions in real life scenarios based on in-game gained knowledge. This philosophy was later called triadic game design in Harteveld's book [17].

This game presents three options: One: training, to familiarize the player with the controls and procedures in the game. Two: the main mode, sequentially more challenging scenarios (like a story mode in an entertainment game). Three: a single scenario generator where weather, type of failures and types of responsibilities can be set up. They also believe that a workshop needs to be set up in order to facilitate effective transfer of the learning goals. They show failure types in levees and are given one scenario to train with this type of issue (mini scenarios). Reflection is a critical aspect of the learning process

Balancing Engagement and Education

Roger and Freiberg [18] stated that learning is difficult when learners are not motivated. Therefor it is paramount that engagement is maintained. This engagement can be achieved by removing cognitive load, user-friendly interfaces that don't wastes the users time, progressively introducing learning content, remove redundancy, preventing blocks in game flow, an appropriate interaction mode, snapshots of currently achieved progress, real-time feedback and self-evaluation, preventing repetitiveness and minimizing too deterministic actions [14].

Testing Serious Games

Testing the right level in Bloom's taxonomy can be done through asking the players to solve scenarios both before and after the serious game. In this way, improvement can be measured correctly. To minimize confirmation bias, the questions should be

intermingled with irrelevant questions with respect to learning goals in the pre- and post-questionnaires [14].

Control groups are useful at all levels [14].

Logs created by the game could be used to do quantitative analysis. In this way, behaviors can be tracked in real time [14].

Asphalt Quality

Therefore, you conduct background research to deepen your understanding of the research area. This research is not limited to a literature (papers, journals, periodicals, books etc.) study, but also includes other forms of information elicitation (e.g. client meetings, attending conferences, exhibition visits). You use the newly acquired knowledge and information to analyze the problem, which in turn leads to multiple research questions by asking “How?”, “What?” and “Why?” questions.

Pre-existing Asphalt Heat & Compaction Visualization Methods

HCQ (Hamm Compaction Quality) is a system for compaction specifically. It tracks position data for multiple connected compactors, interfaces with heat sensing cameras and GPS. Trimble creates a similar system (the cb460), however this system only tracks location and is not specifically designed for compaction. This system does do prescribed amount of passes.

Analysis of Executors and Construction Projects

An understanding of the roles, tasks and responsibilities that an asphalt construction executor has during the construction, has been gained through a wide variety of documents, interviews and meetings. The most clear depiction of the job is *a document created by Infra education center SBW and VBW-Asphalt called "De asfaltuitvoerder"*.

An Executors Project Description

At the start of a project, the executor is informed by their superior by the way of a folder. This folder contains all the project data such as the dimensions, location and amount of layers in the asphalt paving project; the amount, kind and types of machines available to the project; the time, date and duration of the project and information as such. Images and technical drawings for the project are also provided.

The executor then examines the construction site by himself to see if the drawings and the terrain match up. Calculations on the amount of asphalt, the surroundings and the foundation soil or pre-existing asphalt is then inspected by the executor.

The executor then makes a planning based on all the information he has gathered from the inspection and the supplied folder. The executor plans which machines are

positioned where, what asphalt is paved when and how. For example, when a junction is created and how the asphalt lanes should meet. The amount of asphalt is then recalculated by the executor and a series of calls is made between the asphalt factory and the executor to estimate the amounts. This estimate gets refined during the planning phase of the execution. In this planning phase, the speeds of the machines is also set, based on the weather predictions at that time.

The amount of asphalt delivery trucks is then calculated based on the speed an thickness of the paving machines, the capacity of the trucks, the round trip travel time to the asphalt factory from the construction site and the density of the asphalt mixture. The mixture is usually decided by the client or consulted based on client needs and requirements, and is therefor stated in the folder.

All of this information is updated as the project gets closer to starting. On the last evening the weather is checked once again to make sure the paver speed is correct and there is no waiting time on heavy rain. The asphalt factory is called on the day itself one last time before construction can start.

During the construction, the executor is responsible for the timely and quality of the construction process. He directs the crew who operates the machines, measures the asphalt compaction and supporting personal. The order of construction and time planning is managed by the executor and communicated to his subordinates. The way the road is paved, compacted and measured is decided by the crew itself, but is observed and understood by the executor. The executor also keeps the project manager up to date on the progress.

The executor is mostly on site to make sure that the entire project is progressing as planned, issues are dealt with and immediate solutions are found. As he is responsible for the delivery of a quality end product.

At the end of a project a short report is created to justify made decisions and document possible discrepancies in the final product.

Decisions Made by Executors

From the project description, decisions that are made by the executor during a project, can be identified. The executor can:

- Decide the order and way at which the road is paved.
- Decide the amount of asphalt that is needed during the project
- Decide at which temperature the asphalt is delivered
- Decide the speed of the paver
- Decide the initial setup and organization of machinery
- Decide the direction from which the trucks arrive

- Decide how junctions, lanes and additional special cases for asphalt construction are handled
- Decide when to stop paving, or continue at a lower speed, when the weather conditions change
- Decide the amount of asphalt trucks are used (this influences the amount of asphalt available per time frame)
- Decide on when and how often to measure the compaction of the asphalt
- Decide on when or weighter to call the project lead for an update on the construction process
- Decide to postpone the construction process for any reason
- Decide to hire additional trucks during the project, if needed

Possible Problems During Construction

As can be seen, the decisions that an executor can make are quite limited. To get a better understanding on where these decisions come into play, both parties were asked to come up with problems they have had to solve during their career.

The weather could change the speed at which the asphalt cools down, for example when it starts to rain, this means that the compaction process needs to speed up. → If the asphalt cools down more quickly the paving speed needs to decrease such that compactors have enough time to compact the entire road sufficiently. Or compactors need to compact faster.

Machines can break down. → In this case the solution depends on the machine that has a malfunction or the ability to fix the machine. In the case of a paver malfunction, if there are other pavers, these pavers could stop their work and focus on the main road, or the pavers finish this part of road after their own jobs are finished. In the case of a compactor breakdown, the remaining compactors have to do double duty, therefor the paver should slow down for the time compensation.

Long during heavy rain could pose a big problem such that the paving process can no longer continue at that particular day. → In such a case, construction often has to be stopped. Asphalt stays relatively warm over night if situated inside a delivery truck. These trucks then need to be hired for longer periods of time until the rain stops.

The effective travel time between the factory and the asphalt construction site could becomes longer, i.e. in the event of a traffic jam. → In this case the paver could slow down such that it uses less asphalt per unit of time, therefor not having to wait until the next delivery and paving against a cold slab of asphalt.

The wrong amount of asphalt is ordered. → In case of too little asphalt being ordered, and knowledge of this shortage is gained in time, a less thick layer of asphalt could be paved. This way the full length of the project is still maintained at the cost of long

term quality. In the case of too much asphalt, asphalt could be transported back to the factory, or transported to a nearby project.

The required compaction percentage is not reached. → If the asphalt is still hot enough, more passes could still work. Often this requires more compactors. Otherwise a fine is often payed to the contractor, or the entire layer needs to be stripped and redone.

Grading Systems for Decisions

To effectively grade the decisions made by experts and future experts alike, the goals of asphalt road construction need to be defined. As stated in the introduction, there is a trend towards a heightened focus on the quality of asphalt. The quality is dependent on the compaction of the asphalt after paving, but also on the uniformity of the compaction.

The compaction is important as the denser the asphalt is compacted, the longer the asphalt lasts over time. However at a certain point, the asphalt is too compact. After a compaction of 99%, the asphalt is so compact, that it can no longer stretch and shrink with outside temperatures, and cracks start to form during the summer and winter periods. Shortening the longevity of the asphalt significantly. Uniformity is important as differences in compaction lead to quicker wear and tear. The locations where the compaction differs a lot is the first area where wear starts to form as the strains, e.i. endured when cars drive or outside temperature changes, put on the asphalt cannot be uniformly distributed. Therefor it could be stated that the decision that results in the best and most uniform compaction is the best decision.

However, more factors come into play when grading on decisions. Time and money are also important factors in decision making. As time can be converted to money by calculating the labor costs and costs for hiring machinery, total amount of money spend or amount of budged used could be a measure for grading decisions. This measure could also be used to describe long term effects of decisions as paved roads may require reconstruction or costly repair if the guaranteed lifetime is not reached.

A final way of grading could be to pick a plethora of common issues and problems to solve grade decisions based on the universal truths in the field. For example, if a certain decision is always made in a particular situation by executors, that would be the only correct answer. As it is the industry standard.

Usage of (Real-time) Data in Asphalt Construction

As stated in the introduction, the serious game is part of an elective called innovations in asphalt road construction. The course teaches students, among other things, future and current innovations. These technologies are transforming the road construction industry from a trial and error culture, to a data driven culture. The most prominent technologies to grace the industry are Positional tracking, accurate temperature measuring, more accurate compaction sensors and real-time weather data. This section describes these new innovations and how they are used in the field of asphalt road construction.

Positional Tracking

Positional tracking methods are methods of tracking vehicles' positions over time. Usually this data is used to see how machines behave over the duration of a project. This data can then be analyzed and used by logistics optimizers to better the road construction process.

But location tracking can also be used in real-time ways. For example to automatically track the amount of passes a compactor has done over a particular section of paved asphalt. This could give the operator of the compactor a better understanding on how he has compacted and what spots might need additional passes. Another way this same idea could be implemented is to prescribe an amount of passes at the project planning stage. A compactor then knows if his job has been done to the specifications of the project.

Positional tracking could also be used to inform parties not present on the construction site on the progress and operations of the project. For example, if an executor is at the office, or present on an entirely different section of the project, he could still be able to see progress on all other sections that are not in his direct vision.

Temperature Measuring

Temperature has a huge importance in asphalt road construction as the amount of compaction possible depends on how hot the asphalt mixture is on paving. There is a range of temperature wherein compaction is effective. Is the temperature too hot, then the asphalt simply slides right under the compactors, effectively hoping up at the edges. The compactor could even sink so far into the asphalt that all asphalt is pushed to the sides of the compactor and the compactor becomes stuck. When the temperature is too cold, the amount of compaction will either be very little or none at all. But to know how hot the asphalt is at a certain point, heat camera's or heat sensors are required.

There are currently three main ways of measuring asphalt heat, by using heat sensitive camera's, by attaching sensors to the compactor itself that measure the temperature of the compaction cylinders or to embed sensors into the surface of the asphalt. The former two methods are only really useful for measuring surface temperatures as as they cannot measure heat from within the asphalt itself. But surface temperatures are sufficiently accurate to make good compaction strategy decisions.

The first method requires the sensors to be placed before paving and also need to survive the extreme paving temperatures and strains. These sensors are quite expensive and require extensive personal training in order to install and handle them correctly, therefor they are not used as often.

Compaction Sensors

Compaction sensors have been used for a while now to test asphalt compaction after paving and cooling down. These sensors often use sonar to measure the compaction of asphalt, but newer sensors use microwave. These measuring devices are useful for testing the asphalt after completion but cannot be used on asphalt that has a

temperature where it can still be compacted. It does however, in combination with positional tracking and temperature sensors, allow analyzers to draw conclusions and correlations between passes, temperature and compaction.

Real-time Weather Data

As the amount of compaction that can be achieved, is greatly dependent on the temperature, weather plays a major role in asphalt construction. Being able to accurately predict the weather and get live updates on changes is paramount in making decisions on paving speed, compaction strategies and planning. Rain for example, cools down the surface of the asphalt greatly as the water takes up the thermal energy to produce steam. But also factors as wind speed, overall temperature and snow change the cooling down speed of asphalt. The quicker the asphalt cools, the closer the compactors need to compact to the paver. Knowing when weather is going to change, allows the executor to change his schedule to accommodate for these changes.

Methods

Introduction & Sub Questions

In this section the methods for defining the learning goals; ideation; constructing and designing the serious game and testing the serious game are described. This includes justifications for the decisions made along the duration of the project.

The goal of the project is to answer the research questions posed in the introduction. An effective serious game is to be developed for students in the field of asphalt road constructions. To develop a serious game the following needs to be considered:

- What methodologies for serious game development would fit such a project?
- What learning methodologies are effective in serious games?
- What are the learning goals of the serious game?
- How can students be familiarized with new and emerging technologies in the field of asphalt road construction through a serious game?
- How do effective serious games balance learning and fun?
- What external factors are involved in the asphalt paving process?
- How could decision making be trained in serious games?

In the next chapter the learning goals are discussed. The chapter thereafter the other questions are answered in relation to the information to the state-of-the-art.

Defining Learning Goals

To define the learning goals for the serious game that developed, the research questions are to be considered. These state that the game needs to teach students to make better decisions in both phases of asphalt road construction. The decisions also need to incorporate real-time data and environmental factors. To list the learning goals, questions that are raised need to be answered.

- What kinds of decisions are made by asphalt road construction executors, in both of the phases?
- How can these decisions be graded?
- What real-time data is available during road construction, now and in the future?
- What environmental factors play a role in asphalt road construction?

The decisions that are made in asphalt road construction can be found in the state-of-the-art in the section Decisions Made by Executors. However not all decisions might benefit learning or the serious game. On the other hand, there are decisions which are not made by the executor, which do benefit the learning process.

A qualification dossier for the minor, drafted by *Janine Profijt* states the overall learning goals. As the serious game is part of the minor, some of these learning goals might overlap. The ones deemed relevant to the project are summed up (and translated) here:

- has specialized knowledge on processes in asphalt road construction
- has specialized knowledge on different compaction patterns and strategies for asphalt road construction
- has specialized knowledge on execution strategies in asphalt road construction
- has knowledge on the inner workings and operation of asphalt road construction machinery (such as compactors and pavers)
- has knowledge on emerging and current developments of new technologies in the asphalt road construction field
- can handle different compaction strategies
- can handle new technologies and innovations in the asphalt road construction field
- can analyze and map specific data on execution processes
- can defend an asphalt road construction plan
- Considers an implementation on the basis of data interpretation, and communicates on this
- systematically choses compaction strategies and executes those strategies

With the executor decisions, executor activities and minor learning goals, the learning goals for the serious game can be defined. These have been compiled by all the former. After playing the serious game the player / learner:

- has knowledge on the asphalt road construction process
- has knowledge on how temperature effects the compaction speed of asphalt
- has knowledge on how compaction speed influences paver speed
- can deduce the amount of pavers, compactors and trucks needed for a particular asphalt road construction job from the project data
- systematically choses compaction strategies and executes those strategies

- can read temperature maps produced by heat cameras
- can read compaction passes from a compaction map
- can deduce changes in hot mix asphalt as a result of changes in weather conditions
- has knowledge on the amount of asphalt needed for an asphalt road construction project based on the road dimensions
- can deduce the amount of asphalt needed per unit of time based on: distance between the construction site and the asphalt plant; paver speed; capacity of the asphalt delivery trucks
- has knowledge on the different types of compactors and their usage on different types of asphalt mixtures
- can identify and solve commonly occurring problems during asphalt road construction
- can identify, interpret and reflect upon asphalt road construction results and data
- can reason about the speeds at which pavers and compactors should move
- can reason about the distance at which compactors should compact in relation to the paver

In order to achieve these learning goals, player / learners need to be trained by the game and its mechanics. The next section will delve into the conceptualization and ideation on how the game could incorporate these learning goals into its design.

Ideation

Introduction

Asphalt road construction is a very broad topic and there is a lot that can be done by simulation and serious games that could teach many of it's aspects. This project will focus on the many roles that are played by the construction executor during a road construction project. To get a feel for what kind of simulation and serious games could be created to train aspiring executors, ideation techniques were used. This chapter will first explain the initial idea generation, then selection and evaluation of valuable ideas, and finally the presentation of a final concept.

First Ideation Phase

Before ideation can start, one must have a rough idea what a road construction job looks like and what roles the executor fulfills in this role. To get this initial feel for the job, an interview and visit to a real construction site was planned in the very beginning of this project. This was extremely valuable as it gives the ideator a much

needed look into the world of the executor. It also provides an opportunity to ask questions, observe and talk to the target demographic. This grounds the ideators pre-concieved ideas of what asphalt road construction means and provides mental context when reading additional literature on the topic.

Initial Idea Phase

To generate as many ideas as possible, a few methods were used. Method one was to compile a list of well known traditional game genres. This list was then used to come up with game ideas that would fit that particular genre. This list can be found in Appendix 1. The second ideation step was to create a mind map of asphalt construction concepts and relevant game mechanics, this can be found in Appendix 2. The third ideation step was to think of ways that the previous work could be incorporated and or improved to create a serious game.

3 Concepts

From all of the ideas generated, three concepts were generated. The previous ideas were filtered on the wishes and needs of the client (Janine Profijt) and feasibility. They were also filtered on how well they represented real asphalt paving and how well they would be able to teach the learning goals. These three ideas were then pitched during a conference presentation. All three ideas got feedback from industry veterans and teachers who will be using the serious game.

Concept 1 - Planning & Simulation

This concept came about when studying previous work in the state-of-the-art phase of the project. The idea was created from the simulations made previously by the University of Twente and is the most straight forward concept when considering the goals for the project. It is a simulation of an asphalt construction job with added gamification elements such as score, levels and difficulty progression. A planning phase was conceived and added as the initial research showed that decisions made in the planning phase of asphalt road construction greatly effect the end result. This was deemed an important addition.

The simulation would be able to be played multiple times to let the player/learner see what the effects of weather would be on the construction of asphalt. Dynamic weather changes and common (randomly generated, or pre-baked) construction difficulties could also be simulated to train the player/learner's problem solving skills in real time.

 Figure # - Concept 1 sketches

Concept 2 - Asphalt Road Construction Adventures

This idea was conceived during the ideation phase while exploring existing game genres. The *asphalt road construction pamphlet* emphasized the importance of communication during a project. Therefor a concept focusing on this aspect was created. Historically role playing games and adventure games have had a great emphasis on communication with other non player characters. However, due to time

constraints it is not feasible to create a traditional role playing game. Role playing games also rely on very deep combat mechanics and characterization to be enjoyable. Which are not as suitable for an asphalt road construction game.

Adventure games on the other hand have more flexibility as they focus on interaction with environments, characters and compelling narrative to keep the player interested. Which could be used to create an asphalt road construction job simulator / adventure serious game. Where the player needs to visit multiple locations to ensure the crew is happy, the right amounts of materials are ordered and the progress is inspected in real time. Real time events could trigger in pre-designed scenarios that would test the ability of the player to solve these issues.

 Figure # - Concept 2 sketches

Concept 3 - Asphalt Road Construction Tycoon

The third concept focuses on the management of an asphalt road construction firm. The player acquires project contracts from clients and needs to complete these project by assigning the correct amount of machines, resources and personal to the project. Each project is planned and executed by the player on a high level. The long term results are then shown to the player at a later state. The time is sped up greatly such that the player can see how long the constructed road's lifespan was. This way, the player/learner can see the effects of asphalt road construction parameters on the constructed roads quality over time.

The player/learner can manage multiple projects at the same time. This way, the player needs to divide resources between multiple projects. Successful play, results in more resources and rewards such that the virtual company can be contracted for more complex asphalt road construction projects. This is the core game loop for the game. The complexity of the projects increases the difficulty of the game.

The game shows the player a road construction companies complex where he/she can store their resources, machines and manage their crew. With buttons (in the form of buildings) and menus, the player/learner can acquire projects, send resources to those projects and visit the construction site. Statistics on the performance of the player/learner can be shown also. The resources are acquired by buying them from in-game currency which is gained by successfully completing projects. The game is finished when either all pre-designed construction projects are finished, or the company has gone bankrupt.

 Figure # - Concept 3 sketches

Concepts Summary

Concept 1 - Planning & Simulation	Concept 2 - Asphalt Adventures	Concept 3 - Asphalt Construction Tycoon

A serious game with two phases. Planning and Execution (simulation). In the planning phase the user reads the requirements for the road that he is going to build. After thorough reading, he/she inputs the amount of machines, kilos of asphalt. This phase is also used to make a time planning and select the type of asphalt which is going to be used. After the planning phase the user is presented with a fully interactive simulation of the asphalt road construction process. He is tasked with setting the speeds for the compactor(s) and paver(s). The compaction patterns are also directly input by the user. After the simulation phase is done and the entire road is paved and compacted the user is presented with feedback.	In this concept the difference between planning and execution is less defined. This concept focuses on the communication between the executor and his/her crew, the asphalt factory, trucks, lead project manager and 3rd parties. The player is graded based on his communication skills, ability to solve scripted issues and problems in a timely manner and the way the issues are solved. Both long term and short term decisions are presented. At the end of a construction day the player is graded on his performance that particular day.	The player is the owner of an asphalt construction company and is responsible for keeping the company afloat and profitable. The user is presented with jobs and contracts which he needs to fulfill by hiring crew, machines and buying resources such as asphalt. The user needs to assure quality on his delivered projects and needs to eventually manage multiple projects at the same time. The game focuses on the user to be able to plan, balance money, time and resources to get the best results. This concept has a big focus on long term decision making and shows long term effects that compaction has on roads.

Table 1: Concepts Table

Chosen Concept & Results Ideation Phase 1

At a meeting on the future of asphalt road construction education the three concepts were presented. Attendees came from a variety of construction companies (BAM, TWW and KWS among others) and school teachers from the asphalt paving field (SOMA and ROCTwente among others). Concepts 1 and 3 were very well received. Concept 2 was not well understood and it's potential did not match the other two concepts according to the field experts. The group was most excited about concept 1 for its simulation purposes and when done well, could be useful for not just students but existing employees as well.

The three concepts were also tested in how well they would be able to achieve the learning goals that were set out in the earlier section. This next table describes how well the concept would be able to achieve a learning goal on a ++ to -- scale.

	Concept 1	Concept 2	Concept 3
Client	++	-	+
Previous Work	++	n.a.	n.a.
Expert Opinion	++	-	++
1	--	++	-
2	++	+	--
3	++	+-	+-
4	++	+-	++
5	++	+	++
6	+	--	+
7	+	--	+
8	++	+	+
9	+	+	++
10	+	--	++
11	++	+	+
12	++	+	+
13	+	+-	++
14	++	+-	+
15	++	+-	+

Table 2: Evaluation Concepts

The grading is done on the following basis: ++ if the learning goals is integral to the game and is fully taught by the games mechanics; + if the game contains and or teaches the learning goal; +- if the game somewhat contains or could incorporate the learning goal; - if the game does not fully contain but hints at the learning goal; -- if the game could not reasonably incorporate or does not contain the learning goal at all.

When comparing the concepts to the learning goals, concept 1 has the most potential. It focuses on the paving and compaction process most, and is able to teach the differences in weather best. Concept 2 also would be able to do this, but is less flexible and realistic. Its focus was on communication, but this was not something that was needed most, according to the experts and the client. Concept 3 would also be able to show the results of weather changes and incorporate the planning phase, but that is not the focus of the concept and would require a lot more hours of work to realize, as it also has a whole other game to construct.

Ideation Phase 2 - Refinement of the Concept

The concept, at this stage, was very much based on the previous simulations made by the University of Twente as presented in the State of the Art. However, this concept still lacks many of the elements that differentiate a simulation from a simulation game or serious game. This chapter goes into the game mechanics, polish, and ideation of the simulation as a game and the two phase cycle.

To get a better understanding of what an executor does during and before a project, an interview was scheduled with two experts. One of the interviewees currently worked as an executor and the other had been and now lectured at the ROC of Twente. Which is an institute that teaches, among other things, asphalt road construction in theory and practice.

These interviews gave insight into the process that goes before and continues up to the completion of a project. Two major phases have been identified:

1. This is the planning phase of the project. During the planning phase, a binder with all information on the project is given to the executor. This binder contains information like a map with the dimensions of the road and surrounding terrain; multiple tables with asphalt mixture amounts, types and deliveries; a table with the amount and types of machines that are presumed to be needed for the project; the date and time, including the presumed project duration; and additional project details.

   During the planning phase the executor plans out the paving process. When multiple roads are to be paved, or lanes, or crossings; the executor plans the order at which the work is done. This is based on the time planning, amount of machines and requirements set by the contractor of the project. Sometimes certain techniques are prohibited as specified by the specification of the road. One of such specifications is the absence of cold welding, where one or more gaps between cold asphalt lanes are connected with hot asphalt.

   The executor also visits the construction site several days in advance to check the technical drawings, measure the surface and looks for anomalies. The

executor thereafter recalculates the presumed amount of asphalt he needs per hour and figures out how many trucks would be needed based on the distance to the factory. He then proceeds to call the asphalt factory for the precise amount of asphalt, usually one day in advance. This is also the day he looks at the weather report.

2. The second phase is the execution phase. During this phase the executor oversees the construction process, calls with the factory to talk them trough the details and manages the construction crew. He calls with his manager on the progress and directs the pavers, compactors and measuring crew. He is responsible for the delivery and quality of the asphalt.
3. After the execution phase, the executor is required to draft a report on the construction.

This idea of chopping up the game into phases was incorporated into the concept. Where the serious game would feature a planning, execution and feedback phase. Where the first two phases are graded in the final phase of the game. Planning would impact the execution phase based on choices made by the user and user inputed values.

To communicate this idea to stakeholders, a story board was created. This story board describes the idea of the game by showing fake screenshots. This method was chosen as it was quick to develop and change over time as needed. *Low-fidelity prototypes are useful for testing high level concepts as stated by L. Busche*. The image can be seen in figure 15.

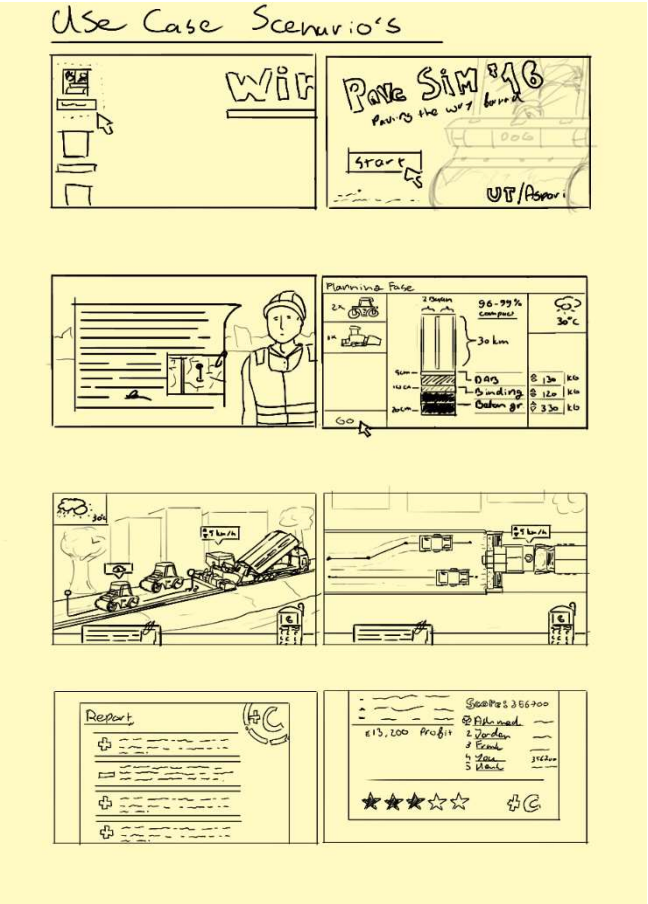

Figure 13: Low-fidelity prototype

The planning phase can clearly be seen in panel 3, the execution phase in panels 5, and 6; and the execution phase can be seen in panels 7 and 8. The feedback is shown in the form of a report, as this closely correlates with the report that has to be written by real executors. The report shows the good and the bad decisions, some statistics, a player leader board and a total amount of stars gained. The idea of using a simple final grading scheme was chosen to clearly communicate a final goal. This would inspire additional tries to get the perfect score and rewards the player based on performance. The good and bad points shows the user where to improve the next time around.

Game Loops

The core design for the serious game is that a player/learner plays multiple scenarios. These scenarios all follow the same structure: planning -> execution -> reflection. This aggregate of actions will be further described as a loop. This name is chosen as the player/learner after the reflection phase, will play a new scenario starting at planning again. As an analog, this loop can be viewed as a product development loop; analysis -> ideation -> concept selection -> prototyping -> reflection -> analysis -> etc.

This loop allows the player to learn from his/her past mistakes by re-playing a specific scenario and using their acquired knowledge. This loop structure also allows for showing differences of weather, temperature and other outside events to the player. As each scenario can be seen as it's own loop. This structure was ultimately chosen as

reflection and discovery based learning methods align well with such a game structure.

Requirements

In order to organize all the ideas for the serious game concept, a requirements list for the game was created. These requirements are split up into the following categories: Must have; Should have; Could have; Won't have, according to the MoSCoW model proposed by *Clegg and Baker in 1994*.

Must have

- Heat distribution simulation on asphalt surfaces over time
- Paver speed selection
- Compaction strategy input
- Asphalt compaction simulation
- Multiple levels / scenarios
- A score based on performance of the player/learner
- The ability to input ability for the width, height, thickness and type of top layer asphalt
- The ability to input the amount of pavers, compactors and trucks to hire

Should have

- Weather simulation
- Multiple compactor machine selection
- Road width selection
- Audio/graphics/gameplay settings for running the game
- A tutorial mode which explains the game to the user
- Asphalt delivery simulation between the factory and the construction site
- Varying levels of difficulty
- Statistics on the performance of the player/learner in the game for teachers
- A planner in which the player/learner can plan the order in which to pave roads
- Repercussions when selecting more machines than needed

Could have

- A scenario / level editor
- Gameplay analytics
- The ability to pave the support layers of the asphalt
- The ability to view the location where asphalt construction will happen
- Different simulated locations where asphalt can be paved

Won't have

- Accurate amount of passes for compaction strategies
- Fully accurate simulation of the entire asphalt road construction
- All machine types and brands for pavers, compactors and trucks
- A company simulator component
- First person control of machines in asphalt road construction
- Accurate heat transfer simulation throughout the asphalt
- Physically accurate heat and compaction simulation

Final Concept Description

Planning Phase

In the planning phase, the player/learner will select which and the amount of machines the player/learner wants to use. The player/learner then selects the parameters of the road that is to be constructed; the width, the height, the thickness and the type of asphalt. Finally, the player/learner can inspect the weather.

Execution Phase

The execution phase is where the simulation is started. In the simulation, the pavers and compactors can be controlled. The player/learner can only control the paver's speed, where as the compactors can be controlled freely. To see the progress of the compaction process, the user can select to see the heat and/or compaction of the asphalt. The compaction process can be seen through a free from camera which can be manipulated with a mouse and is rendered in 3D. The user can, himself, decide on when the simulation should end, and the compaction process as reached a satisfiable result.

Feedback Phase

In the feedback phase the user is shown the results of his/her performance. Here the compaction of the road is shown, including various statistics on the compaction and speed at which the road was constructed. The feedback phase will also give the player/learner tips and feedback based on that data. For example, suggesting to use more or less machines or compacting a certain section of the road. The feedback phase will assign points based on both the planning and execution phases of the game loop. At the end an aggregate score will be given.

Serious Game Development

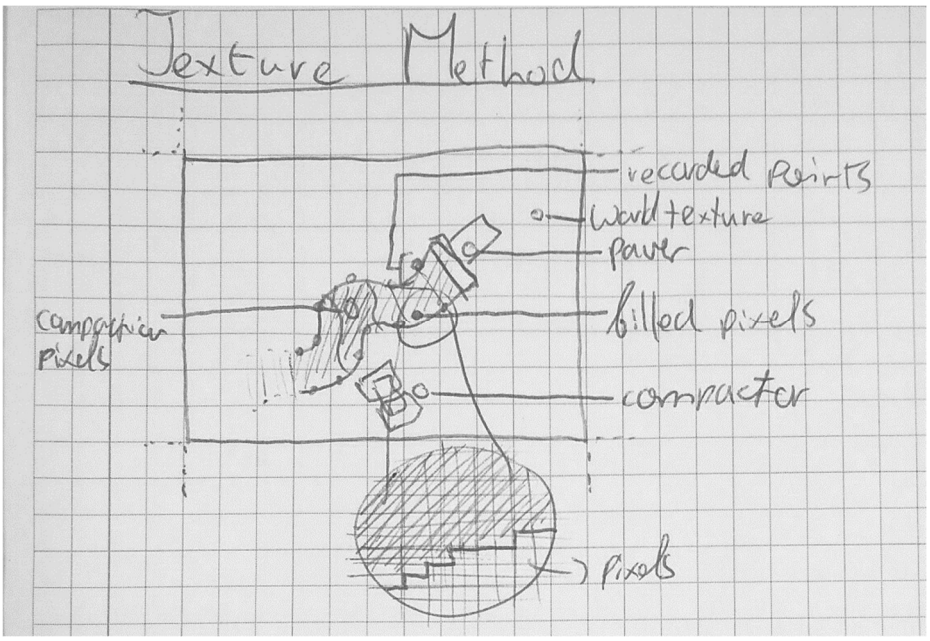

Figure 14: Diagram - Pixel Method

For the development of the serious game, Unity 3D was used. As it has a wide variety of platforms it can build to and has support for WebGL, which was deemed useful for environments where executable binary formats are not allowed. Unity was also a the most obvious choice as I had the most experience with the tool. Which allowed the focus to shift from the development of the serious game to the design more than with any other tool.

Data Representation Strategies

At the start of the development process, most of the focus was put on the simulation of the heat and compaction. To implement the way heat and compaction were represented in the game, a system needed to be developed for containing that data. A few strategies were considered for this to happen:

1. The first strategy was to create three gigantic textures, as big as the entire world where paving and compaction could happen. These textures were used to keep track of where paving had happened and where compaction had taken place. The paving texture would contain just one bit per pixel as it was only used to check if that particular position had been paved yet. The heat texture would store the heat level as a color only on the pixels where the paving

texture contained a 1 in that particular spot. The compaction texture would work in the same way. Where the compaction value would be stored as a color between fully transparent black up to fully opaque green.

Filling in the pixels works the same for both a compactor and a paver. Each of the vehicles kept a record of it's width at the points where paving or compaction would occur. It would record these global position every x amount of seconds. Once at least two records were created, a rectangle containing these four points would be filled by the program on the designated textures. This rectangle was filled with the use of the *Bresenham's line drawing algorithm* and the implementation was based on both *Sunshine2K's examples* and *Alexey Frunze's example on Stack Overflow*. This was done twice for both triangles of the given rectangle. The rectangle was drawn as triangles due to higher performance as trying to draw rotated quads.

This method has a big disadvantage, which is the size of the textures. As a big portion of the texture will be unused. This data bandwidth is basically thrown away. On systems with low video memory, resolution will be very low, thus the paving and compacting resolution will be very low.

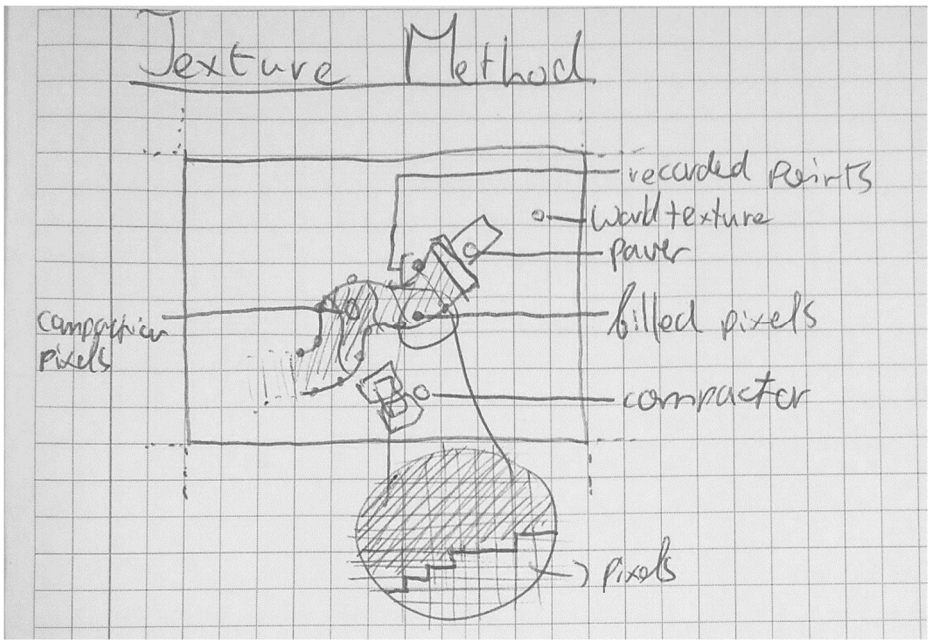

Figure 15: Diagram - Pixel Method

2. The second approach to the problem was to generate 3D meshes on the fly. A road generator would generate a mesh during the paving process where once every meter of paving two points at the edges of the paver's paving rods two points would be recorded. These points would then connect with two triangles to the existing mesh. Textures are generated for the mesh where the data for temperature and compaction are stored.

   The mesh is also used for checking collision between the compactor and the paved road. When the collision mesh of the compactor collides with the mesh, the compaction can start. The compaction is updated on the mesh with a similar polyfill algorithm as was used in method 1. Points are recorded, and

from those points, the texture is filled. However, this method has a major downside as the resolution of the pixels and compaction is different based on the turn radius of the mesh as can be seen in figure 17. The inner pixels of the curve are of higher resolution per meter as the outside pixels of the curve. Due to the texture being stretched by the distance of the vertexes of the mesh. This problem does not happen with the texture method.

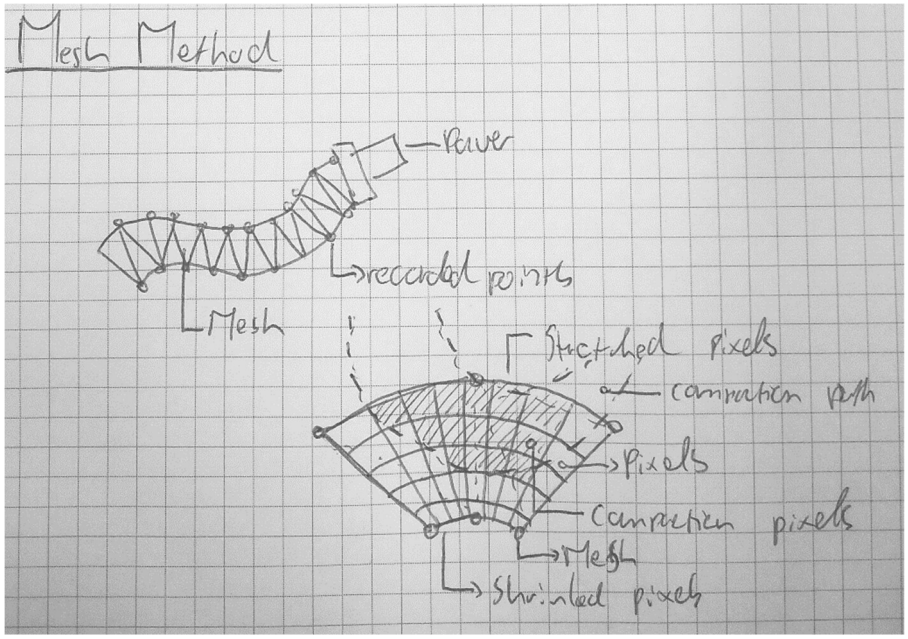

Figure 16: Diagram - Mesh Method

3. The third method for saving the heat and compaction data is to not use textures at all. Positions could be saved where the data type for the recording of position also contains heat and compaction data. The points would be generated by the moving compactor and paver. These points could either contain a width of a line, or two lines could be generated. Both approaches work similarly for the rest of the explanation. Volumes could then be calculated for where the compaction overlaps the paving lines. Line intersection points including with the lines generated by the compactor and paver describe the volume. This can be seen in figure 18.

   All compaction would then be represented as compaction volumes. The paving volumes and compaction volumes would then have to be rendered in some way. As they are only represented as these points. Temperature would be stored in the points where the values between points would be interpolated in distance and time at which the point was generated. The resolution of this representation would obviously depend on the frequency at which points are generated.

   Two downsides to this approach are that there is no way to store individual heat/compaction points, only volumes. The second is that the complexity of the code and the rendering is greatly increased. But the benefit is that potentially the same information is stored in non discrete points and uses less

space to store the data. It trades complexity in code and concept for data storage.

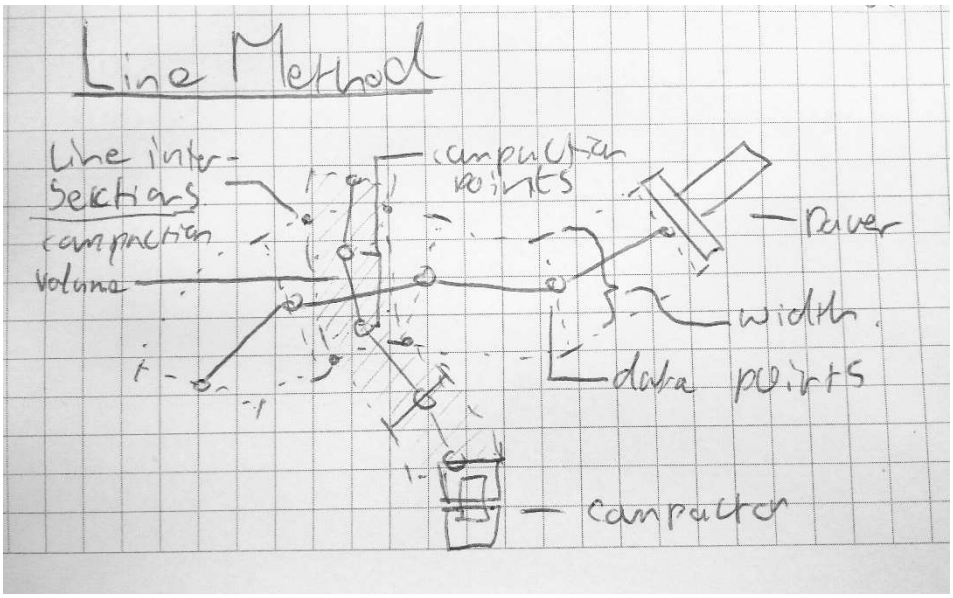

Figure 17: Diagram - Line Method

Decision & Conclusion

For the sake of keeping things simple and direct, method 2 was chosen with the restriction of allowing the paver to only move in straight lines. The ability to pave curved surfaces was deemed not necessary to convey the intended results of external influences on the quality of asphalt and the compaction process. This also got rid of the curve resolution problem. This solution is also a nice compromise between the easiest to implement and data abundant solution 1 and the complex but sparse solution 3. Therefor solution 2 was chosen for the representation of heat and compaction data.

Texture Filling Strategies

In the first iteration of the code, the texture pixel updating procedure was very slow and slowed down the performance of the game significantly. Taking seconds to do the work. In order to optimize the texture fill process, a few ideas were considered. One was to parallelize the work. Where once updating needed to be done for either the heat or the compaction, worker threads would be created to do rendering for a particular part of the road. These would then run until finished. Once the threads would finish they either would start again, with new data, or wait for new data to arrive. If data would be submitted twice to a stack of work, only the most recent directive would be executed. However Unity WebGL does not currently support threading.

Another option that was considered was to update the pixels on the GPU as shader code. This shader would then take the input bitmaps, temperature table and the delta time to calculate the heat dissension. However, this idea was not developed further as custom shader code is not supported on WebGL but also not supported on older video cards. Which the game was likely to encounter as most school PC's at schools that were visited had older hardware, and no dedicated GPU.

The last set of solutions that were considered and eventually implemented, was to only require a small amount of pixels for every column (width of the road), and let the texture mapping do the stretching over long distances. This approach reduced the compute time for the heat texture significantly. As the amount of pixels in one dimension was reduced significantly. The constraint that this optimization brought was that heat dispensation would no longer be able to be influenced by the compaction process. Therefor, thinner but denser asphalt would not change the rate at which the heat gets dissipated, nor does passing a compactor reduce the temperature.

Another optimization for the heat texture updating, was to only record time intervals on distance intervals. This time position would then look into a lookup table with heat dissipation values. These values would then be used to look up heat color values during rendering. These values would be replicated for a number of pixel columns based on the distance to the previous time point. These values are not interpolated, to reduce computation strain.

Heat Dissipation Data

Due to a lack of specialization in the field of thermodynamics, and the complexity of modeling heat dissipation over time in asphalt road construction, lookup tables were used. These tables were generated by *MultiCool*, a tool that calculates hot mix asphalt cooling time graphs for asphalt paving. Output for such a graph can be seen in figure 19.

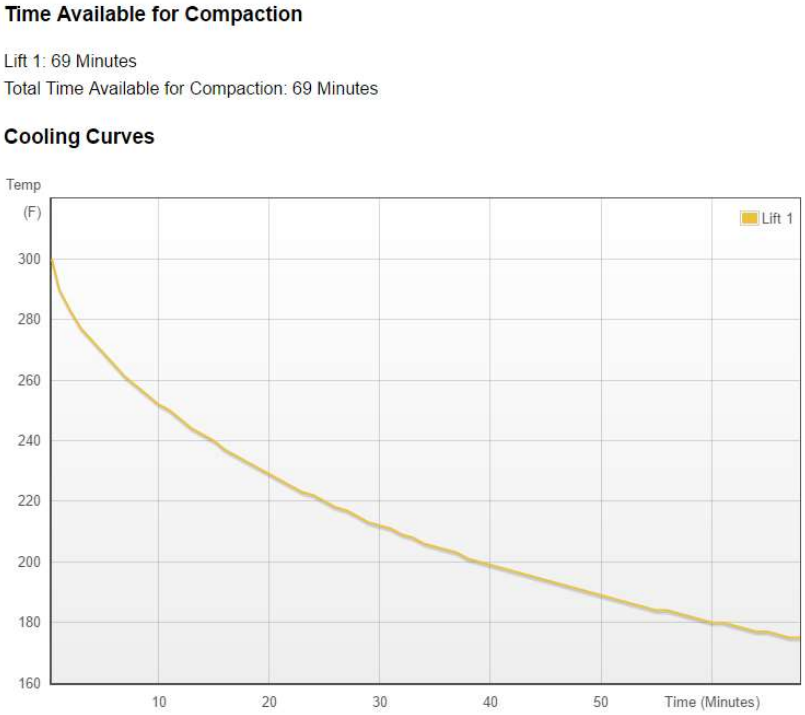

Figure 18: MultiCool Graph

With this tool many parameters are available for input, but as the serious game is only interested in showing the difference in temperature as it effects cooling, temperature was used as the only changing input. In the final version of the game four temperature data curves are used. For -5°C, 0°C, 10°C and 20°C. All values in between these data

sets, such as for example 18°C, are linearly interpolated. Sadly, no precipitation parameters are adjustable in the tool. No alternatives were found during the literature review, nor the execution stage of the project. These would have to be developed in the future to get a better approximation based on the thermodynamics between asphalt types and precipitation types like: water, snow and hail. The temperature data table can be seen in appendix 4

Visualization

All of this data needs to be communicated in some way or form to the user in such a way that makes sense and is clear. Such that the player/learner can make decisions based on the data, without looking at a set of incomprehensible numbers. To do this successfully, visualization techniques already used in the field, or just being introduced in the field of asphalt road construction were analyzed.

Denis Makarov at the University of Twente is developing a system for giving feedback to compactor operators. He develops ways to visualize temperature and compaction data in real time on a screen that will be placed in compactor cabins. Markov encodes temperature between red and blue. Where red encodes the hottest point and blue encodes the coldest temperature. He also uses discrete rectangles to encode the data with a relative low resolution per meter.

The Hamm compaction meter and the *Trimble CB460 Control Box* are both in cabin screens which display asphalt compaction through the amount of passes that a compactor has done on a particular location. Both systems use GPS to track the location of the compactor. Both systems use color to represent compaction percentage of pass counts. Both software show what color maps to what percentage of compaction, or what amount of passes. The main difference is that the HAMM device seems to also be able to measure the compaction, this Trimble unit was not. The HAMM device also seems to be able to track a higher resolution with its GPS module as the resolution on the render is way higher. However, this cannot be confirmed as the system was not seen in person.

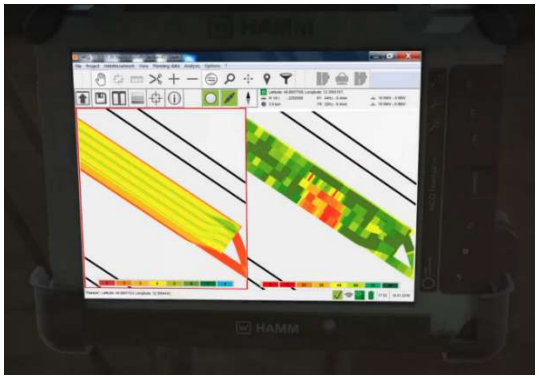

Figure 19: HAMM compaction meter render

Figure 20: Trimble visualizer

Both visualizations are quite similar to that what Makarov was pursuing. They use a two dimensional, top down view of the work and the asphalt. They both allow you to set the colors of the compaction and passes to the preferences of the user. They also allow for the representation of the compactor on the screen. Such that the user can see how his machine relates to the positioning on the screen. In the HAMM system, the colors for the compaction and the passes are the same in the render, which could be confusing in real life usage. As the two could be confused. The HAMM system also uses green for the right amount of passes and uses a color scheme from red to yellow to green to blue. The Trimble system uses a blue to light blue to blue to red to blue scheme as default. Which was very confusing due to the fact that three different values were encoded as the same color.

Usage in the Serious Game

As the color scheme convention for temperature maps in heat camera equipment also uses colors from white to yellow to red to blue to black, a similar was used in the serious game to represent heat. It was changed however, from red to yellow to green to blue. The color green was added as a way to tell the user when to start the compaction process. When the color starts to move to the blue part of the spectrum, the compaction process needs to be completed as compaction will slow down on colder asphalt. The created gradient can be seen in figure 22. The transitions from

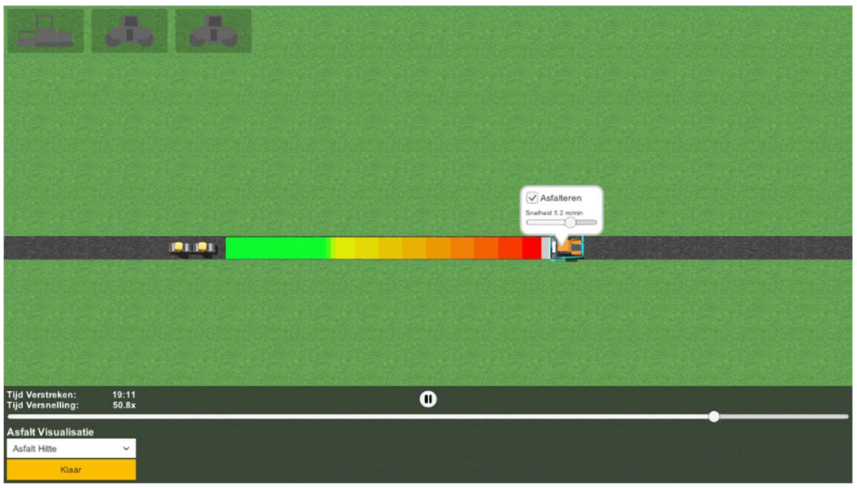

Figure 21: Heat Data Visualization

yellow to green and green to blue are quite harsh. This was chosen as it made it more clear when the asphalt would be 'green enough' to start compaction.

The compaction visualization was chosen to linearly interpolate between fully opaque and fully transparent green. This was chosen as it could be easily lain over the temperature. Such that both temperature and compaction percentage could be viewed at the same time by the user. Showing the relation between the two and for convenience. Another reason was to show the percentage of compaction as a percentage of green. More green meant more compaction. This was to be easily read by the user. If the compaction exceeds a 100%, or compaction is started too early (on too hot asphalt) the asphalt is essentially destroyed. Compaction then shown as a red color, indicating that something has irreversibly gone wrong. This value cannot be easily seen on the figure below, but is actually there.

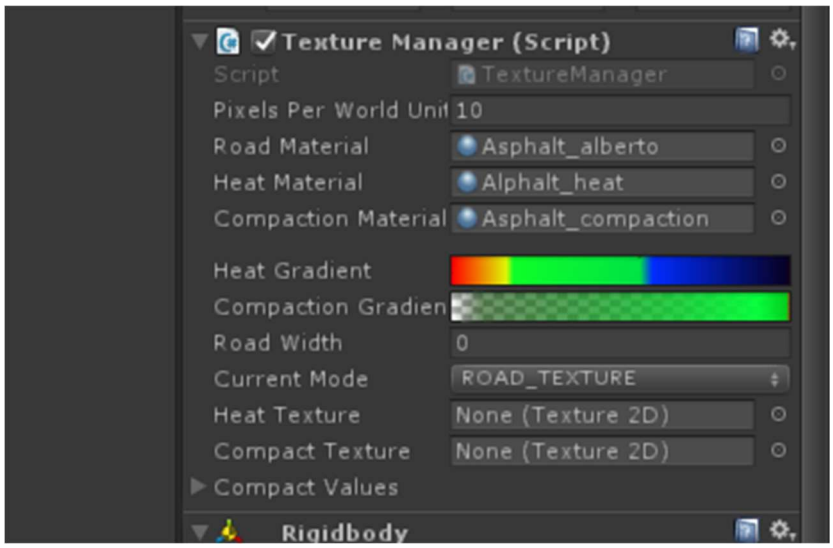

Figure 22: Unity Color Gradients

This feature was implemented as a custom shader at first. However the custom shader posed problems on older hardware thus was removed. In the final version of the serious game, this feature is no longer available and thus the motivation for choosing green is less sound. However, in the future this could be easily changed by another

developer as the gradients can be changed in a visual editor. This editor can be seen in figure 22.

Machines Control

The next challenge for finishing the simulation was deciding on how the user was to control and direct the pavers, compactors and trucks. Not only on how to control, but also how much control was required in order for the game to convey its meaning, be accurate to real life but also fun to interact with. This traility of relationships was also described by *Harteveld*.

Compactor Control

At first the easiest system was implemented. Where the user could control a waypoint on the screen where the compactor would move towards. This can be seen in figure 23. However, this system proved to be quite cumbersome to control as it requires you to change the point quite frequently. It also was not realistic as the execution manager does not control how the compactor moves, nor was it very fun to control the compactor in that way. It did give you a lot of control where the compaction would happen. But due to all the negatives the system was replaced with a ping pong equivalent quite early in development.

The second system that was tried out was to let the compactor move between two user definable points. This motion is often called ping pong, where an agent moves between a series of points by going back and forth along the chain. This solution can be seen in figure 24. This still requires control, but slightly less. Only when the lane of road has been compacted. This system allows the user to control in what direction or angle the compactor would compact the asphalt.

Figure 25 describes the third iteration of the compactor control. In this system, just the locations between which the compactor compacts, can be set. The compactor moves up and down over the with of the road by itself. Compacting the road evenly. This way, the user can focus their attention on figuring out when to compact, without knowing the specifics in the way compaction is done. Which is closer to realty as the executor usually does not interfere with the way the compactor compacts, but only where the compactor is too close or too far away from the paver.

The last solution was suggested during a conference on asphalt road construction, where the game was first demoed. It is to set the distance of the waypoints relative to the paver. Such that the waypoints travel with the paver as the paver moves forward. This way, even less attention needs to be given by the player/learner. Allowing the player/learner to focus on the other tasks that the player/learner has to do, such as inspecting the temperature, keeping eye on the weather changes.

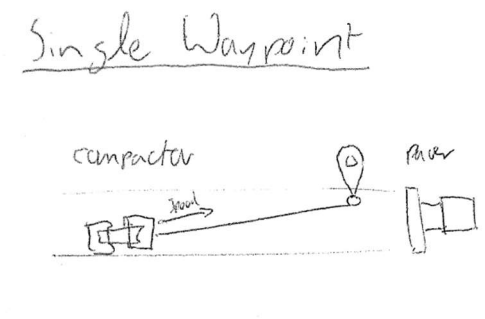

Figure 23: Compactor movement - single waypoint

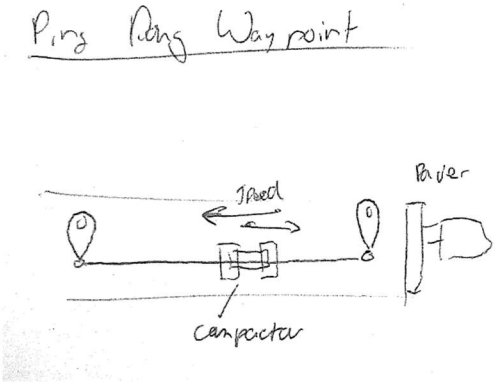

Figure 24: Compactor movement - Ping pong

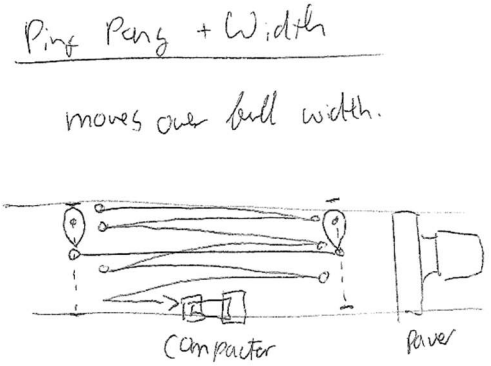

Figure 25: Compactor movement - Ping pong with width movement

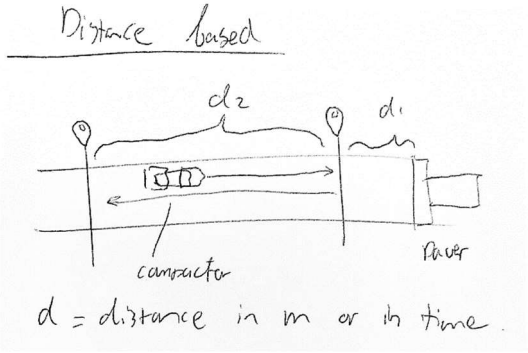

Figure 26: Compactor movement - Distance based

Paver Control

Paver control only had two implementations during the development of the game. At first, in the early stages of development, where all the systems and the mechanics of the road data representation was explored, the paver was moved by a single waypoint system. The same system that was used for controlling the paver. However, once the decision was made to only allow for straight roads, the decision was made that the only thing the user would be able to control was the speed of the paving process. This removed the need for needing waypoints and simplified the control for the user.

Usage of Proposed Systems

All but the fourth system were implemented into the games prototype at one point or another. The first was implemented to test out the way the game would feel to users and how fast the paving process would be. Soon it become obvious through usage that the one waypoint system was too demanding and required too much attention in order

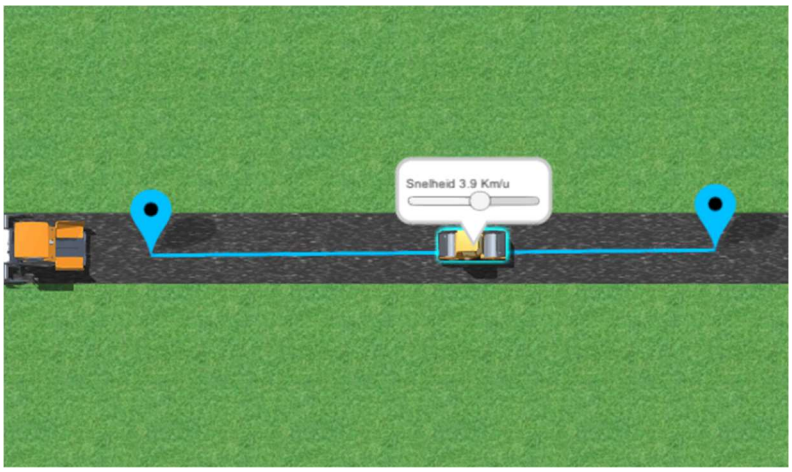

Figure 27: Waypoint System

for the rest of the simulation to work. It was quickly replace by the two waypoint system, which was used for the first initial test with users.

During the test it became clear that controlling a camera in a virtual space, in addition to controlling two compactors and a paver in this way was too difficult. Users struggled more with the camera and the paver waypoints than the actual heat and compaction strategies. Which is explained in more detail in the results section. After the first test the camera system was changed to only allow for horizontal movement, lessening the ways the user could fair away from the machines in the game field. The waypoint system was changed to the third iteration to reflect this change, allowing the user to focus more on the temperature and compaction instead of the compaction patterns used by the compactors.

The last system (distance based on paver) was again proposed by one of the students during the second test, but was not implemented due to time constraints.

Camera Control

The control of the camera almost had as much development as the control for the machines. At first the control of the camera was completely free form. The camera could be moved through 3D space via translation and rotation, modeled after 3D modeling applications. As the students would probably be most familiar with CAD(Computer Aided Design) like software. These applications use the middle mouse button for panning (moving side to side and up and down) with the middle mouse button, rotating the camera with the right mouse button and zooming with the scroll wheel.

However after discussion with D. Makarov, who tested 3D software systems with compactor operators, found that these users struggled a lot with 3D systems. Therefor the free form camera was removed and control was simplified to just panning the camera and zooming in. Removing the ability to rotate the camera. This decision was easily made as the third dimension was not presumed to add to the game's mechanics, nor the learning effects.

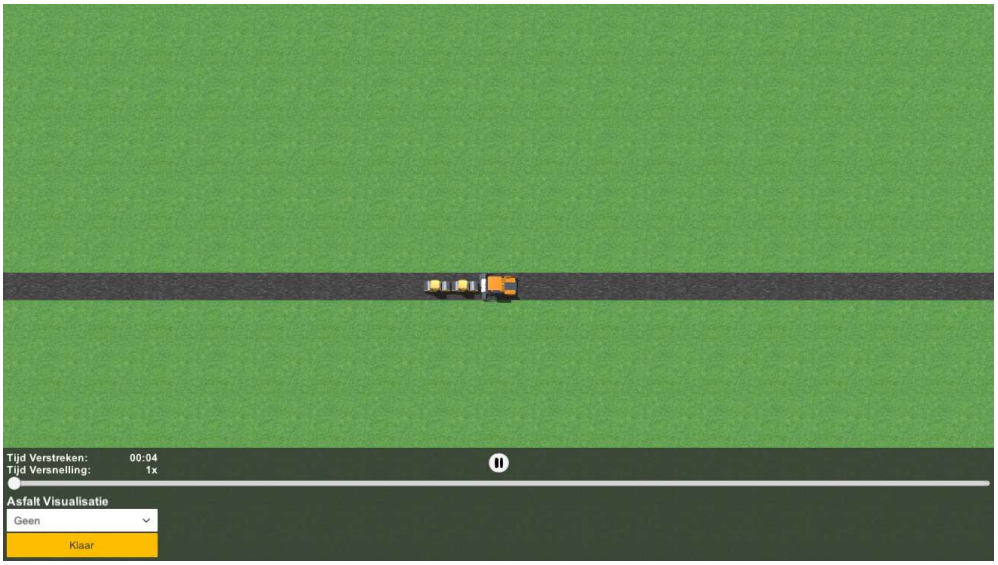

Figure 28: Birds Eye View

The camera was therefor simplified to a birds eye view of the compaction process. Where the camera could be moved froward, back, left and right, and also zoomed in and out. This view was chosen such that long stretches of road could be seen at the same time. This was important to convey as the changes in temperature are clearer to see over longer stretches of asphalt. Compaction can also be compared easily this way. However, at the expense of realism, as the executor usually has no way of seeing this much asphalt at the same time. This view could only be obtained with a plane or by the use of remote controlled drones.

2nd Iteration

As stated in the results section and before, it became clear that the control of the camera was, even with the simplification, still considered hard by users. This was also apparent through observation during the test. Uses frequently lost their machines by veering the camera too far away from the machines. Not allowing them to find their machines thereafter. It was also noted that the machines were difficult to control with such a camera. Due to this feedback the camera system was changed. The movement in the z direction (up and down) was removed. Where as the camera could now only be moved parallel to the road. Zooming was still possible as it was deemed important to both provide an overview and give the user control to zoom into specific areas to see more detail.

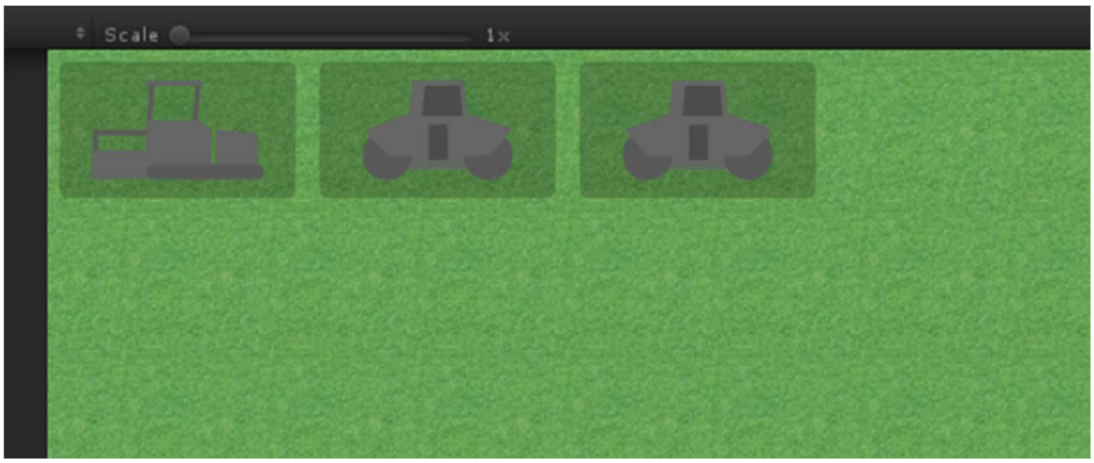

Figure 29: Camera Focus Buttons

Buttons were also added to the UI of the game, such that in the event of the user losing track of one of the machines, the camera could be focused on each of the machines with their dedicated button. This can be seen in figure 28.

Mouse control was also rebound from the middle mouse button to the left mouse button which is the more standard main way of interaction with applications.

Time Control

Once the temperature curves were implemented it became clear that the entire compaction process could take quite a while. The cooling down curve for weather temperatures of twenty degrees Celsius could take more than forty five minutes. This would take way too long for any play session and would not allow the player/learner to learn from their mistakes in a reasonable amount of time. Here, the realism was traded for fun and learning.

In order to keep the speed of heat dissipation and duration at which compaction could occur realistic, a time scale was implemented. This time scale allows the user to speed up the simulation, while keeping the simulation time accurate. For example, at two times speed, every simulation second would take half a second to play out in real time. In order to give the user enough time to react to the events, the user is also allowed to pause the simulation at any point. The simulation can not be slowed down, as the asphalt paving process is not a very fast process to begin with. Thus is was not deemed necessary to implement this. The pause button and the slider for time scaling can be seen in figure 27.

This feature was easy to implement as everything in the simulation is based on the delta time between simulation updates. Such as the speed of machines and the cooling down effect of asphalt. It was simply a matter of scaling the delta time based on the time scale selected by the user. However, updates of the temperature and compaction points did need to decrease if the time scale was too high, as the performance otherwise dropped too much. Unfortunately decreasing the accuracy of the simulation at extreme speeds. The maximum time scale was set at sixty times as the simulation would then simulate every minute as one second real time.

Feedback Phase

The feedback phase was developed after the first iteration of the simulation stage was finished. This order was taken as the feedback screen would need data to represent and the types of data that would be able to be generated from the simulation needed to be known. Some data was required, for example the average compaction, which is the quality measurement that is used by the game. But also parameters like collisions between machinery, the amount of select machinery, the duration of the compaction process and the compaction excess. These were used to give the player a particular grade. All planning phase parameters are also shown in the feedback phase.

What also can bee viewed in the feedback phase is a map of compaction. This map shows how well certain sections of asphalt are better compared, compared to other sections. The greener a section is, the better compacted it is. If the section is shown as red, the compaction has exceeded the maximum, or was compacted at a too hot temperature, which ruins the asphalt irreversibly.

However, during the first test it became clear that the goals for a particular play session or scenario, were not clear. It was also noted by J. Profijt, that the correlation between the statistics and the stars system was not clear. This lead to the development of the goal system that will be described in the game mechanics section.

Figure 30: Feedback Phase

The goals can be seen in figure 32, where the sections that are noted as green, are the goals that have been successfully achieved, and red sections (not seen in the figure) are goals which have not been achieved. In a goal section, the target is shown, and also if the goal has been reached. Completing all of the goals results in a full amount of stars, while reaching some of the goals would result in a fraction of the achievable stars.

Planning Phase

Once simulation and feedback had received enough work for initial testing, the planning phase was developed. During the planning phase, the player/learner was to chose which and how many machines they would need; the parameters of the top layer they were to pave; and given an overview of the predicted weather. The final design of which can be seen in figure 33.

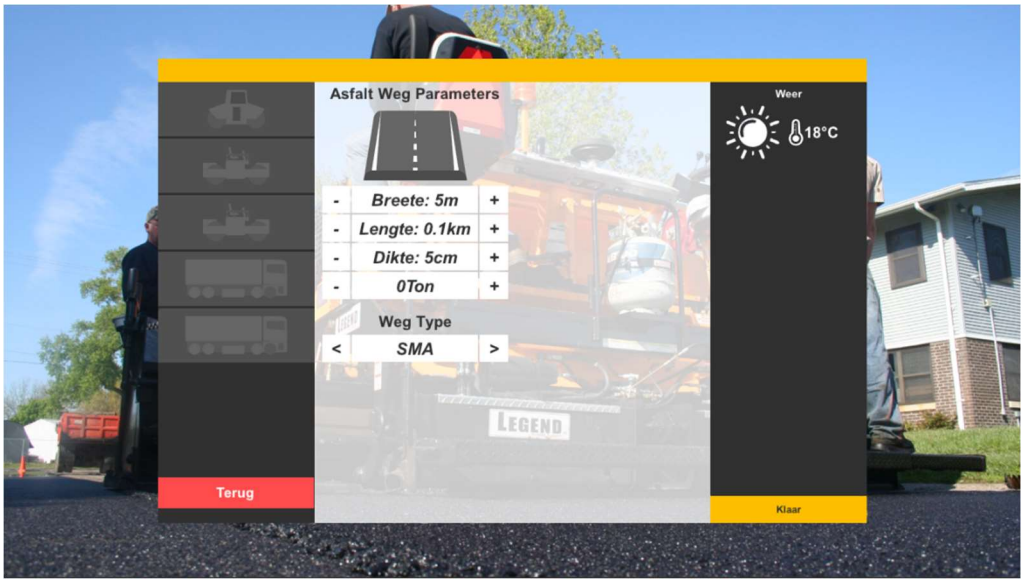

Figure 31: Planning Phase - Settings

However, in order to make every scenario and/or level distinct, a description of the work an eventually later, the goals of that particular level were added before the settings input screen. The final design for the planning phase description screen can be seen in figure 34. The goals can be found on the right side of the screen and are displayed in a list fashion. All of these goals are also shown in the feedback phase where can be viewed if the goals have been reached.

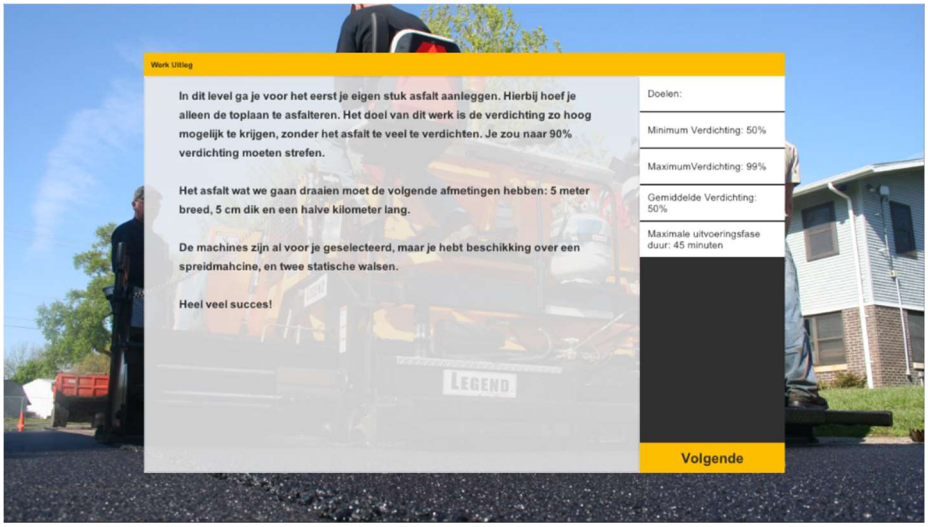

Figure 32: Planning Phase - Description

Game Mechanics

Besides the game learning the intended material, it should also be fun to play. To achieve this goal, in game rewards were added in addition to separate levels. The rewards were first conceived as in game currency. Where the player would spend money to buy new machines and tools, and gain money from completing scenario's successfully. However, peers who looked at the prototype thought that the amounts of money were not clear enough of an incentive to continue to play the game. In addition, it would be hard to give player/learners specific rewards for doing one particular phase well. This would, for example, not allow for rewarding just the planning phase.

To remedy this, inspiration from popular mobile games was drawn. For example, the widely popular game *Angry Birds* uses stars to note how well the level was played. These stars could be given to the player as a reward for particular phases or for achieving certain predefined goals.

Tutorial Design

In order to explain the base game mechanics and the control of the game, a tutorial was made that would show the player/learners what a player could do in each of the phases and how to achieve those objectives. The tutorial was created in the game and walked the player, step by step, through every motion or action that needed to be performed by the users. Finishing the tutorial was made a requirement before the player/learners could continue with playing the levels.

The tutorial explains the game by giving an introduction text to the phase and what needs to be achieved in that phase. Then the required steps for that phase are explained by text bubbles and text screens. For example, if the user needs to input a value into one of the fields in the planning phase, a text bubble would pop up above the input location telling the user what value to input. Once the user input the correct value, the tutorial would continue onto explaining the next step.

Interactive tutorials were made for the planning phase, execution phase and the feedback phase. During the planning phase, the goal of the planning pahse; how to select machines; how to input types and dimensions for the top layer; weather and goals are explained.

In the tutorial for the execution phase the control for the paver, compactor, the time scaler, the selection buttons, the camera and the asphalt viewer (the selector where the user selects weighter he/she wants to see the heat, asphalt or the compaction) are explained. The relation between heat and compaction is also shortly mentioned.

Finally the feedback screen is shown with the resulting data of the previous tutorial (execution phase). Here the results, and what they mean, are explained to the user such that the player/learner can read the results and understand the feedback when he/she starts playing the levels.

Level Design

Multiple levels were introduced to solve two problems. The first was to make the player feel as if they were progressing through the game, thus motivating them to play the serious game more often. The second problem was to solve the issue that many

mechanics need to be taught to the user. In order to not overwhelm the user with concepts they need to grasp, levels could be used to introduce, and test these mechanics one by one on their own, and then layer these mechanics on top of each other.

The following mechanics need to be taught to the user. They have been ordered on when they need to be taught, as some concepts need to be understood before other concepts can be learned.

- How the amount of passes of a compactor influences the compaction process
- How asphalt temperature impacts compaction
- How the paver speed impacts how compaction happens
- How weather temperature impacts asphalt head dissipation
- How the distance between the asphalt production factory and the asphalt construction site impacts the amount of tons asphalt per unit of time
- How the asphalt delivery truck count and truck capacity impact the tons of asphalt per time unit
- Which types of compactors can be used with which type of asphalt
- How types of asphalt have impact on how compaction is done
- How to plan such that no cold welds are needed at cross sections
- How precipitation influences asphalt heat dissipation
- How to react, as an executor, to sudden changes in the scenarios. Such as random events and sudden weather changes

The levels were structured such that all mechanics are first learned in solo and then are tested with the other mechanics. This is done such that the user doesn't get overwelmed during the learning process of new concepts. After that the user is tested on the gained knowledge on how these mechanics work together with the other previous learned systems. These are then repeted and hopefully become more and more of a second nature to the user. At the end of level 16, all mechanics are tested with random events. This could be continued at infinitum as the random events could be used to generate an endless amount of levels. This structure can be seen in figure #.

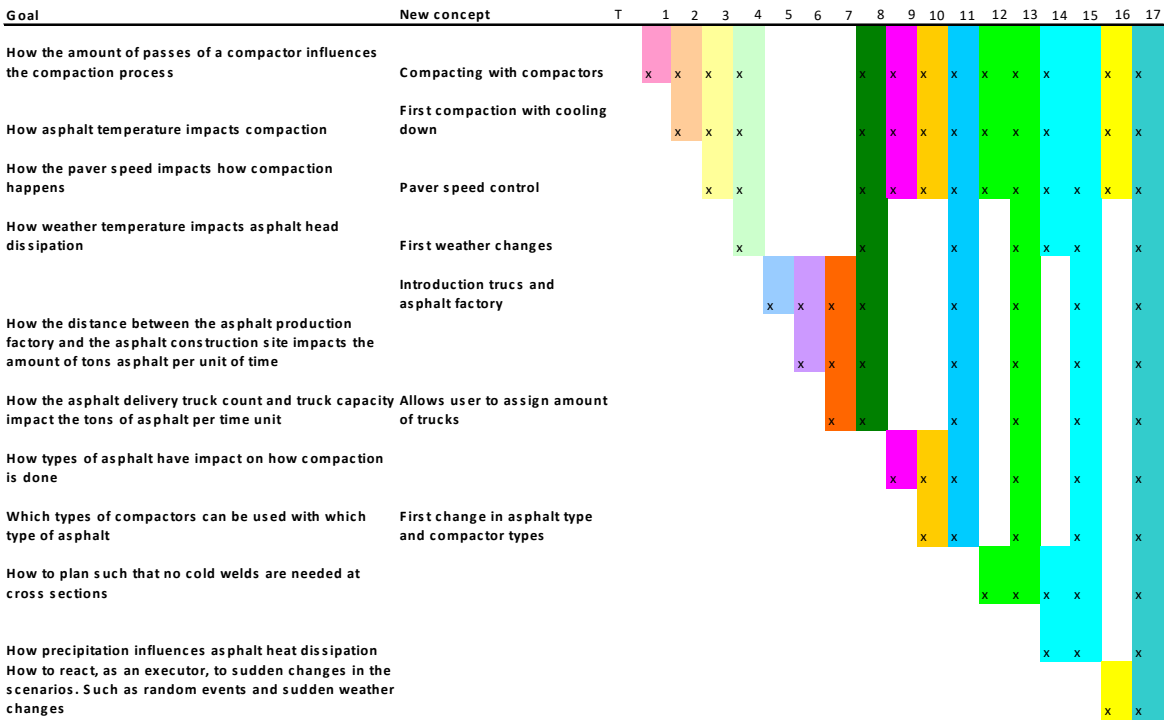

Table 3: Level Structure

Level Loading

In order to keep the game flexible to change and fast to iterate, a system was developed for loading in level files. These files consists of text parameters that describe the level. The game then loads all text files with the '*.level.txt' extension. Parameters like: flavor text, the amount of machines that are required, the temperature of the environment wherein the paving process will happen, weigher the user is allowed to set parameters himself and the goals that need to be reached during the level. An example of such a level can be seen in appendix 3

Text Loading System

To make it easier for lecturers and teachers to make changes to the written text in the game, a text loading system was developed. This could be done when terminology changes or the level of language used needs to be changed for the different levels of students who use the serious game. All buttons, text, labels and other text in the game is loaded through a text file which is the database for the game. The structure of this file is a key value pair where the key is bound to a place in the serious game where the text is used. The value is the actual text used for that particular location.

This feature should also be useful when the game needs to be localized for another country, field and or language. If the text for a particular key cannot be found, the text becomes ERROR, as this gives the users of the technology can clearly see where the text is not loaded correctly. This was chosen as seeing the wrong text is worse than seeing an error.

Analytics System

To test the game internally besides the user feedback and the test described in the section “Serious Game Testing”, a system was build which records statistics on the serious game. These statistics range from the amount of buttons pressed, the scenes loaded, the amount of time spend in each of the asphalt construction phases and also the results from that level. With this data, pain points in levels or the tutorial mode can be analyzed and potentially solved. For example, if a player/learner is spending way more time on the tutorial for the planning phase as the designer has anticipated; then that could be an indication that the tutorial is not really designed well.

These analytics could also be used to record a user’s progress through the game. This progress could then be shown to a teacher who could take action if one of the students lags behind schedule and needs more personal attention.

The analytics system is an API which can be called. The API is able to do the following things:

1. Record time with a text label by calling a StartTimer and EndTimer function with the same label
2. Increment values for specific labels
3. Attach values to labels
4. Attach strings to labels

These values are then uploaded to the Google Drive of a given user who is logged in. This prompt is done for every windows user. The login hash is saved in the registry of that user and can be reused, for example a teacher or a tester of the serious game. At the start of the serious game, the name of a user is used. The name is used as the file name for <name>.txt file that is stored on Google Drive. That way the user of the analytics system does not need to retrieve the file from every individual computer. The drawback of this system is that all computers do need to have an active connection to the internet. The analytics data is saved in JSON format, as this is easily read by many applications and is an often used standard.

Results

Test Design

The first test was done early in development when the game did not yet have all the features which were planned for the serious game. The game had no introduction for levels, did not contain many levels and the levels that were made were made for the purpose of testing the level loading system and were not designed according to the level progression diagram that can be found in figure #. The feedback was also still lacking polish and the analytics were not implemented at the time of testing.

The goals for this early test were to establish the current fun factor of the game, the game's stability on target machines (the game had up until this point only been run on the development machine) and to test the understandability and tutorial of the game.

Questionnaire

To reach these goals, two methods were used: observation and a questionnaire. The goal of the questionnaire was to ask the testers how well they understood each part of the game, how much they enjoyed the game and to ask them if they saw potential in the ability to learn from the game. The questionnaire was also used to ask for feedback that the testers might have. This method was chosen as it was not realistic to interview each of the players individually and asking the group as a whole would possibly not allow all participants to contribute equally. This way the opinions were also anonymous, which would allow the participant to be more honest when being critical of the game.

The questionnaire was divided into six parts. Each part, excluding the user questions and the final comments section, was created to reach one of the testing goals. The order of the sections was not chosen in any particular order. The questions were subdivided as follows:

- General questions on the previous experience of the testing user
- Questions on the playability of the game, including usability; how clear the game was to the user; the perceived difficulty on each of the sections of the game and the usability of the game's controls.
- Questions on progress in the game
- Questions on perceived learning
- Questions on the fun factor of the game
- And finally, an open section for users to fill in any comments or statements on the serious game

Observation

The goal of observation was to see how well the game would perform on the target hardware. Also mistakes or questions from players on how to progress would allow for a better understanding on what parts of the game needed more explanation.

Observation was also used to check if the findings of observing would correlate with the results of the questionnaire.

Test Planning

One hour was planned for the test. The plan was to explain about the project first, to explain what the goals of the project were and what would be expected of the students. Then the students would be allowed to play for 30 minutes, first following the tutorial (this was forced in the game, as the levels would only unlock if the tutorial would have been completed) and then time to try the sandbox levels where students could test the implemented systems. After the thirty minutes, they would have to fill in the questionnaire which is included in appendix 5.

First Testing Round (SOMA)

The first test of the serious game was done at a school for asphalt road construction SOMA located in Hardewijk, the Netherlands. Here the game was first tested with actual students. The test group consisted of eighteen (seventeen of which filled in the questionnaire) third year students who were following a course in asphalt road construction at MBO 3 level. The test was held in a computer room on the second of February 2017.

Technical & Design Issues

During the test a lot of issues became apparent. The game did not at all perform well on the target hardware and was so slow that most of the game was near to unplayable as the simulation showed at times less than five frames in a second. This made it really hard for the testers to control the machines and achieve the desired result. This was due to the fact that the target machines were very old and ran on old Pentium chips without any dedicated graphics acceleration hardware and very old versions of Direct X. The 3D models that the game was trying to render were too much for the machine to handle and get a decent frame rate.

Furthermore, the rendering of the game did not work correctly. The lines which should have appeared between the two waypoints flickered all around the screen and on some machines the rendering was flipped on the horizontal axis whilst the clicking was still normal. This made it impossible on some machines to click the machines to select them for moving.

Another problem which was more to do with how the control was defined was that users moved the camera too far away from the machines. This made the users lose track of where the machines and the road was. Requiring to hard reset the game re-load the level, as there was no menu implemented for this feature. However, this issue was just a problem after users had already completed the tutorial. When the camera worked, it was deemed very difficult and was commented on and asked about a lot during the test.

Through observation it also became clear that users had a lot of issues with the constant moving of the waypoints which became a problem as the game was not able

to pause at this point in development. The amount of control that this needed from the user was too high.

The goals of the levels were not clear to many of the students during testing. This was the case during the actual levels and not the tutorial. This was a valid criticism as the levels had not really been implemented but created just for the purpose of testing the systems. The differences between that were designed did not seem to change the game play that much in the eyes of the testers. Some students noted that the levels should feel more distinct in order for them to care for the levels at all. The levels were also deemed too easy as the results achieved in the execution phase did not lead yet to more or less stars in the feedback phase.

Questionnaire Results

According to the questionnaire results, most testers (82.3%) at least monthly play games, and for 47.1% of testers this was not the first serious game they played for education purposes.

Understandability of the Game

On a Likert-type scale for the clearness of the menus, the average was 5.06 with a median of 4 and a mode of 5. The testers noted that the difficulty of finding the levels However, the perceived amount of questions asked was only for 5.9% answered as never.

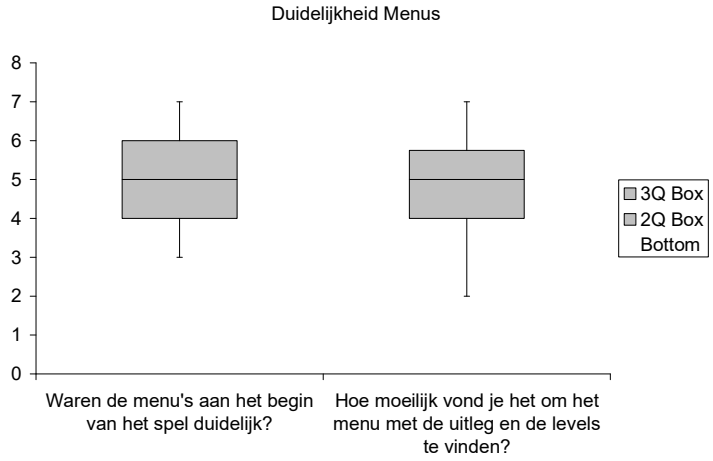

Figure 33: Box Plots - Levels

Most students (82.3%) answered to ask between ‘often’ and ‘once in a while’ questions during the planning phase. The average for the overall clearness of the planning phase was found as 4.29.

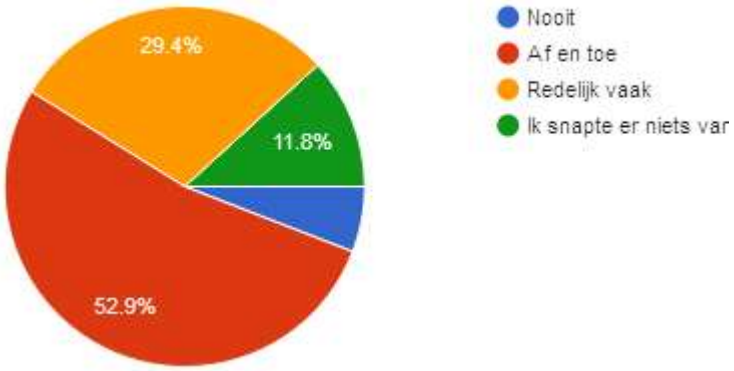

Figure 34: Pie Chart - Question Frequency Tutorial

In the explanation of the tested clearness of the execution phase was measured at an average of 4.12 with a median of four and mode of five. The perceived amount of questions asked during this phase was 70.6% for ‘often’ and ‘once in a while’ as can be seen in figure #.

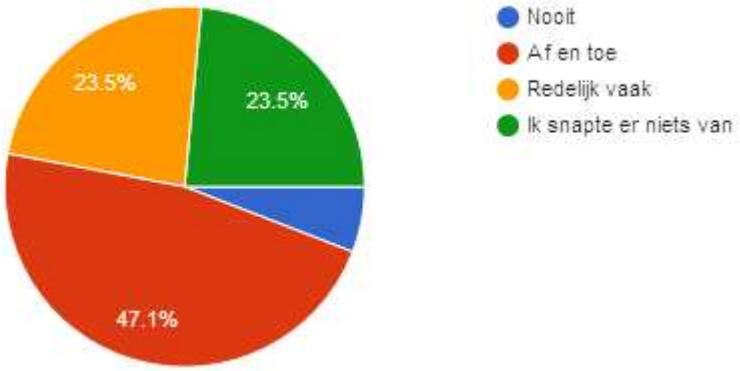

Figure 35: Pie Chart – Question Frequency Execution Phase

Game Input

The clearness of the camera was responded as 47.1% for yeas and 52.9% as no. Using the scroll wheel control was answered as 76.5% not pleasant and 23.5% as pleasant. The selecting of machines in the game was found as 4.23 mean, the compactor control as 3.00 and the time control (pausing and simulation speed) as 5.06. The box plots for these parameters can be seen in figure #.

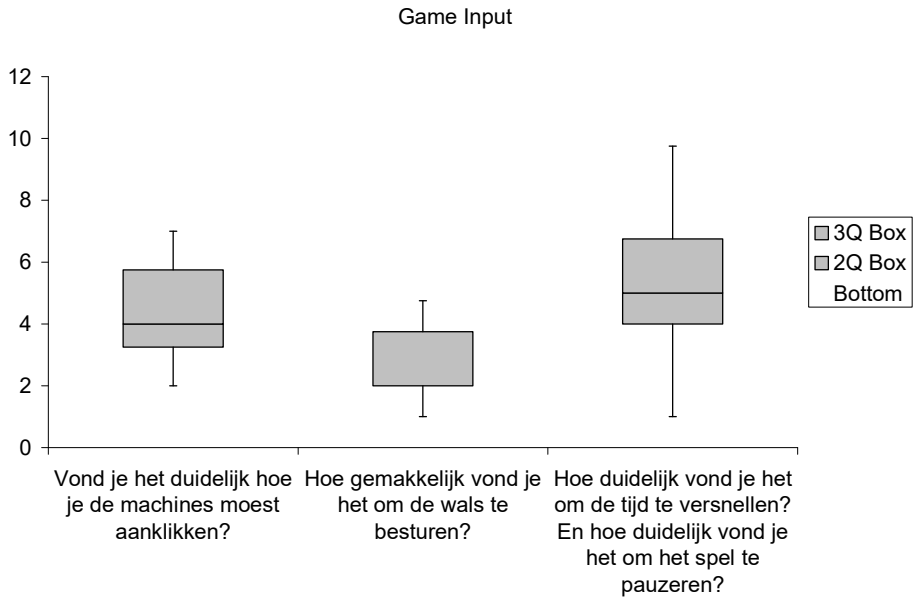

Figure 36: Box Plot - Clearness of Input

Level Progress

According to the survey all students have reached the end of the tutorial. Most students reached level 1 (76.5%) and the remainder reached level 2. Level 3 was not reached by any of the students. In figure # can be seen that most testers filled in that they perceived the levels hard to finish. The mean for difficulty is 4.82. The clearness of the goals of the level has an average of 4.82 between ‘very unclear’ and ‘very clear’.

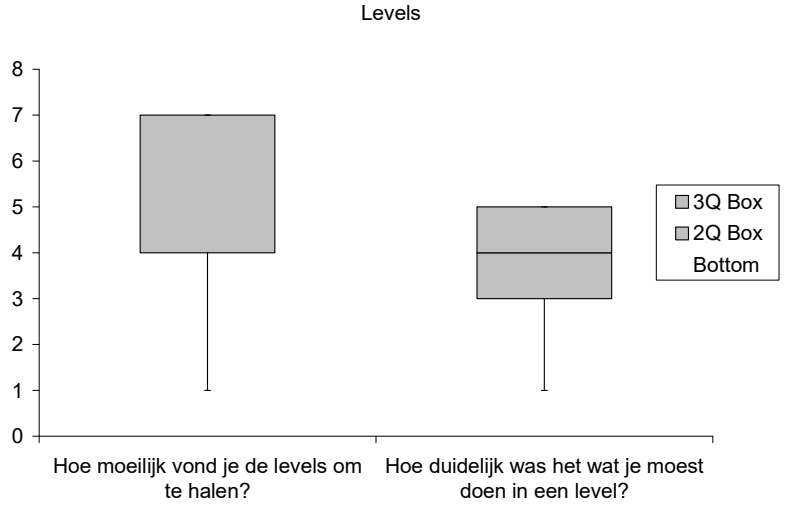

Figure 37: Box Plot - Difficulty

Learning Related Questions

First, a question was asked on the perceived level of the student. Here the student could answer how much knowledge the tester had on asphalt road construction. The

students could chose between: no knowledge, rudimentary knowledge, average knowledge and above average knowledge on the asphalt road construction processes. The average tester answered average knowledge (76.5%).

On the question on ‘to what extend the serious game gave the tester insight into the asphalt road construction processes’, the testers answered 2.65 mean on the Likert-scale. And 3.18 mean on how useful the game could be for understanding asphalt paving processes. The perceived knowledge on the effects of temperature on paving was stated as 4.88 mean. To the question on if they had gained information, by playing the game, on the cooling and paving behavior of asphalt in different temperature environments, the testers thought to have gained 3.59 which is between ‘no gain in knowledge’ and ‘a lot of gain in knowledge’. The box plots for these questions can be seen in figure #. The distribution on the last question can be seen in figure #.

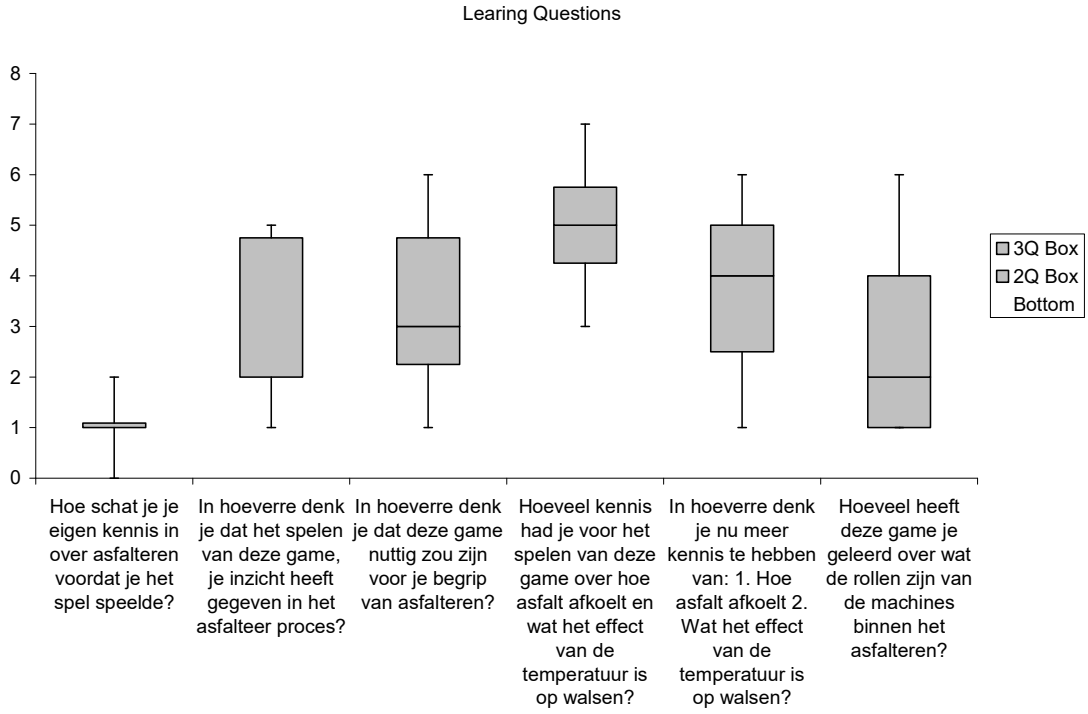

Figure 38: Box Plots - Learning

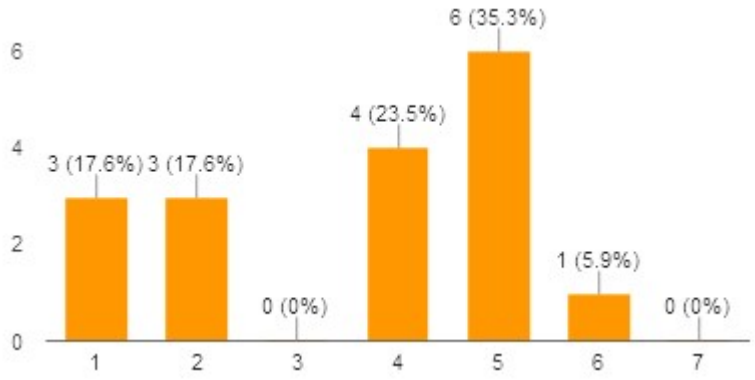

Figure 39: Likert Scale on Learning Effect on Roles of Machines

On the question of what the testers, thought was most/leas valuable, education wise, when playing the game, the testers gave these statements:

- To see how slowly the asphalt cools
- The temperature and the time speed
- When you should start compaction
- The least valuable was to set the speed of the pavers, as we have nothing to do with that
- To work with heat
- It is all so slow and did not move forward much
- Everything
- ?
- What you need during paving
- I thought it was completely unrealistic
- The whole game
- The paving of the asphalt itself
- How you should do it
- If it [the game] could be improved, then you could learn something from playing it. But we already know quite a bit on asphalt paving, therefore we learn less from this [game]

Questions on Fun Factor

The first question was on the perceived duration of the game. The students were asked on how long they thought they played the game. Possible answers were: less than 5 minutes, between 5 and 10 minutes, between 10 and 20 minutes, between 30 and 45 minutes and more than 45 minutes. An equal share of testers stated that they played between 5 and 10 and between 10 and 20 minutes (each 47.1%). The rest stated that they played between 30 and 45 minutes (5.9%).

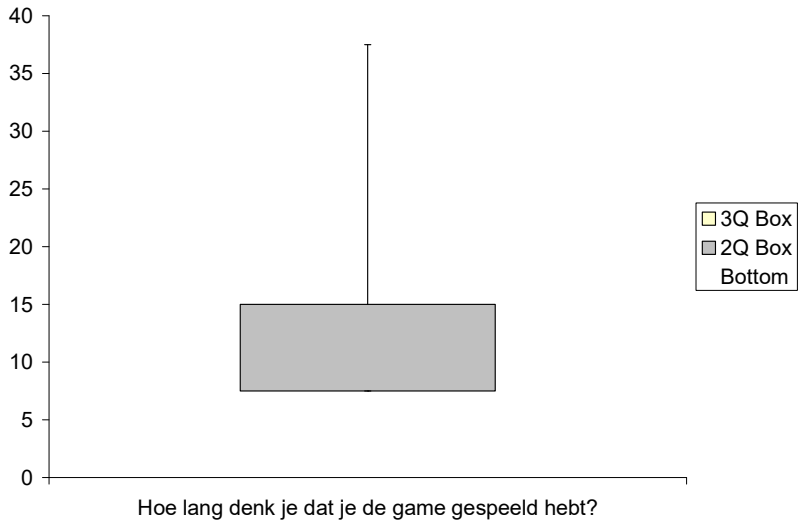

Figure 40: Box Plot - Percieved play time

The next question was on the motivation during the session. The testers could choose on a Likert-scale between ‘very unmotivated’ and ‘very motivated’. The graph for motivation can be seen in figure #. The mean motivation was calculated as 3.41. On the question, ‘how much fun they thought the game to be’, the mean between the extremes ‘not fun at all’ and ‘very fun’ was 3.06.

Out of seventeen students who filled in the questionnaire, only one would definitely replay the serious game, eleven would not and five students entered they might replay the game.

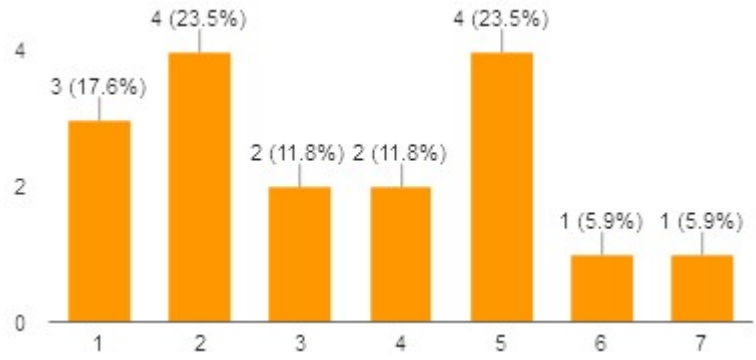

Figure 41: Likert Scale - Fun factor of the game

On the question if why the testers liked or disliked the game, the testers answered the following ways:

- The game didn’t work well
- It was hard and quite unrealistic, but you are on the right track
- Id did not understand how it [the game] worked
- It did not work properly

- Not very realistic yet
- Unrealistic and you cannot operate the machines themselves
- It was too hard to work with
- No
- It is very unrealistic and it doesn't work that well
- The paver kept driving in a straight line
- The game was hard to control and everything was going very fast
- It was not realistic
- I didn't understand any of it
- I did like it as it was a bit realistic
- I liked it, but it was not very educational
- For a beginner it was fun

Remarks

Students could leave remarks at the end of each section and one last time at the end of the survey. Some of the more interesting remarks are summed up here. The following remarks were made:

- The game should be more like farming simulator, as you can actually sit in the machines during construction
- It would be nice to set the sheering distance between the front and back roller during paving on dynamic compactors, as this is used in the real world two
- Use more field related language in the game
- More levels would be nice
- I could not select the asphalt paving machine and could not change the lines between the compactors
- Make the mouse less sensitive
- Controlling the compactors was very difficult and I could not operate the compactors in the way I wanted to
- The game is very choppy and the scroll to zoom and the moving of the camera was not a success

- I want to operate the machines myself
- Left mouse button instead of the right mouse button to select machines and show everything from the perspective of the operator
- If it works better I would like to play it [the game] again
- You should make it more clear at the start of the game what the goal is
- Try to make the game more realistic and les stressed, then it'll be fine
- [The game] Looked realistic
- Play the game on a screen before you let students play the game
- I would not put any more of your time into this game

Iteration on the Serious Game

Identified Problems during Testing

During the test it became very clear that the game was not yet ready for testing, as the game ran so poorly that lots of students struggled with controlling the game. This lead to a lot of questions and forced reboots of the game in order for students to make any kind of progress. The time acceleration feature made everything worse as the simulation takes longer to compute the results for longer delta times between frames. The camera control and mouse button assignments made it also hard for students to select and control the machines during the execution phase.

The lack of goals in levels was also a major problem. As can be seen from the results, answers and remarks made by students during the first phase. In order for students to feel like they progress, and have the motivation to play more than one level, the levels needed to have distinct goals and difficulty levels.

It also became clear that, because of the poor performance in combination with the complicated camera controls, lots of players lost their machines during game play. This issue had to be solved for the next test as well.

The state of the game might also have interfered with the results on fun, learning effect and understandability of the game. The tutorial and or the feedback during the game was obviously also not thorough enough to convey the systems of the game. As many students answered in their survey, not to understand the game. This was also apparent when observing, as many students asked questions during the tutorial. During the planning phase it also became clear that not all students read the tutorial texts. As some enthusiastically started to fill in amounts not prescribed in the input fields, and therefore getting stuck in the tutorial phase. As the tutorial requires the player to input the predefined amounts.

Solutions to Identified Problems

Performance Optimizations & Bug Fixes

The first problem that was worked on, was performance. As the game was hardly reaching more than five frames per second on the target machines, it became clear that a lot had to be removed and changed. This was also shown in the in engine profile that Unity 3D provides. Here you can see statistics on what functions in your code, and in the engines code use up the most time in a frame. An example of such a readout can be seen in figure #.

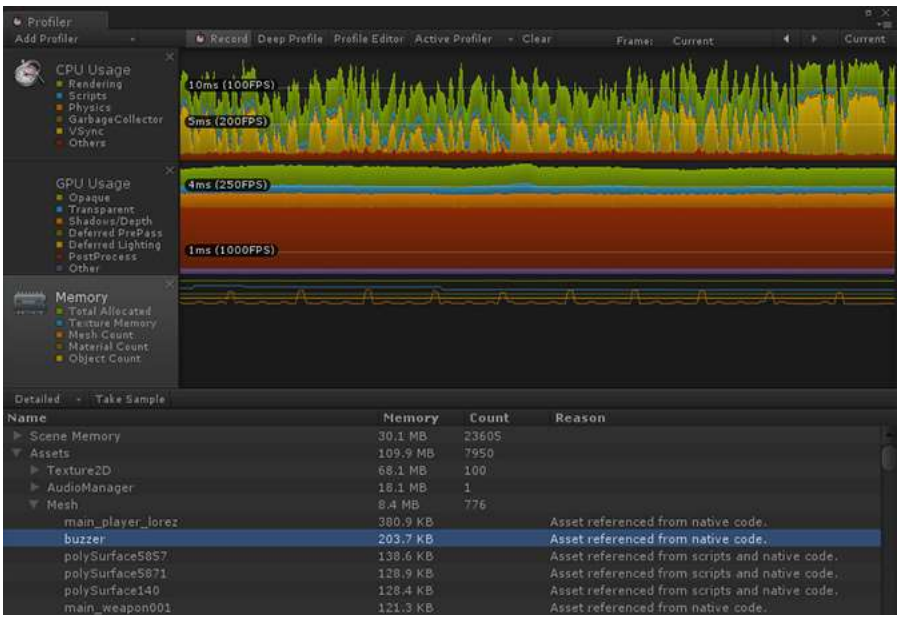

Figure 42: Unity analytics window

https://blogs.unity3d.com/wp-content/uploads/2013/03/memory-profiler.jpg

Rendering

The first optimization that was done, was on the way the game was rendered. Up until this point, all machines were rendered as 3D models, with fully dynamic lighting and shading. This was the first thing to be removed. All dynamic and static lights were removed and screenshots of the fuly rendered machines during game play were taken to be used as 2D images to be drawn on the screen instead. This method was chosen as it did not change the appearance of the game for the second test group, as the light didn't change anyway during game play, and would significantly increase performance.

Three dimensional terrain with grass and other vegetation was also removed. This was replaced by a flat grassy texture as a background to the road. This removed the shading of the hills but did not change the game by much. This also increased performance significantly.

However, there was still a problem where the blue lines between targets used by the compactors would glitch all over the screen. This issue was solved by removing the custom shader that was developed to be able to display the compaction and the temperature at the same time. This was justified as in real life, an executor is currently

also not able to see both at once on current technology. Users would still be able to switch between asphalt texture rendering, temperature map rendering and compaction rendering, but this was implemented as swapping materials instead of swapping the input to a shader. This solved that problem.

Another problem was the inverted rendering bug. Where on some systems, the rendering of the UI and post processing effects would be rendered normally, but the rest of the game inverted. This problem was not found and was not fixed up until during the second test.

Simulation Calculations

In order to optimize more, the calculations used for rendering the heat changing of the asphalt, concessions had to be made. For one, instead of calculating a heat difference for every width pixel of the road, only one heat value for the whole width is used. This cuts the amount of calculations done per length pixel significantly. For future development, this needs to be reverted as it does not allow for changing the temperature as e result of compaction. Nor does it allow calculation of heat dissipation differences at the edges of asphalt.

Another optimization is in when the calculations are done. Before, calculations would be done on a distance interval. Where, if the paver moved a certain distance, another render would take place. However, as the game allows for the speed increase of time, the amount of calculations per real second would increase also. This was changed to a minimum fixed time interval between calculations. Such that rendering, at worst, only happens between these intervals.

Collision

When the change was made to change the game from 3D to 2D, the collision was also changed in the hope that it would improve performance. Calculation of dynamic objects with each their own 3D collider could be costly. Therefore the system was changed to the in engine (Unity 3D) 2D collision system. This was done after analysis of the profiler showed that collision took up a sizable portion of the compute time per frame.

User Interface Improvements

In order to make the game easier to control, major improvements and restrictions were imposed. For example, the camera was fixed on the y-axis as the paver was also restricted to moving linearly in a straight line. As the player would never have to view anything other than the machines, which all move on the road, the camera was fixed on this axis. This was done to improve the understandability of camera control. Now the user only has to move the camera along the asphalt, and zoom in and out if need be.

Buttons were also added for every machine that participates in the asphalt road construction during the execution phase. These buttons can be pressed at any time and the camera will smoothly move towards this target. These buttons were added to remove the issue where users would not be able to relocate their paver and compactors when straying too far away from these machines.

The compactor movement was also changed to reflect the new camera movement restriction. Compactor waypoints were constraint to points on the center of the road. It was decided that compactor control was too demanding and too difficult to operate. The compactor would, after the change, move itself medially (opposite of laterally) on the asphalt between the two waypoints as described in figure #. This was justified as the user should not have to focus on how compaction is done, but when compaction should happen and where. This is closer to the goal of the serious game to teach these relations.

The speed acceleration of the game was also capped at sixty times normal speed. This was deemed sufficient as in the maximum case; the game would simulate itself at one minute per second. This tried to solve the issue where the compactors would move way past their waypoints when the delta time was too high. Another solution to this problem was also implemented. The delta position was changed to bounce back based on the target frames per seconds, such that the compactor would reflect the other way instead of overshooting the waypoint.

Tutorial Improvements

In order to make the tutorial clearer to the users and to reflect the implementation of the goal system some things were added. The goal system was explained in the tutorial text itself, and balloons were used to explain how the goal system would work during the reflection phase.

In the input section of the planning, all fields that did not have to be interacted with during the specific phase of the tutorial would be disabled. This was done to make it more clear what section of the interface should be interacted with, and what should not.

Some text balloons which were timer based, were extended in display time to make sure that even extremely slow readers would be able to read the entire text before the tutorial would continue.

Implementation of Goals

To give levels more meaning and the player a goal to work towards during play, goals were added to levels. These were added along side an introduction flavor text. These flavor texts would set up the scenario of the asphalt road construction job where the user would read what kind of job they were going to face and what kind of requirements would be for that particular job. On the side of the introduction text, a pane was placed where the goals are summarized. These goals would be tested in the feedback phase of the level where the player/learner would see if these goals would have been reached. These goals could then be defined in the level text file. The final version of these screens for both the feedback and the planning phase can be seen in figure # and figure # respectively.

Temperature Implementation

It was here that the temperature curve simulation was implemented. Before, all levels would start with the same temperature curve and there would be no way to make

levels with different heat dissipation rates. More curves were generate as stated in the designated section on this feature, and added to the game. The newly generated levels in preparation for the second test were using this feature.

Other Improvements and Changes

During this phase of development, the analytics system was developed along side all the previously mentioned changes. The levels that existed were removed and replaced by better designed levels, with each of the levels having their own goals, flavor texts and settings. Starting with a level where everything in the planning phase would be predefined and ending in a level where all parameters were left to the user to input.

After a talk with one of the teachers who taught at the ROC in Hengelo, The Netherlands, it became clear that the speeds for the paver and the compactors were not correctly interpreted. These were changed to reflect this inaccuracy. At first the pavers moved at five meters per second, but this was the maximum speed of the machine, and did not reflect the speeds of pavers in real world construction scenarios. These speeds would be more in the range of five meters per minute, an order of sixty slower. The speeds were changed to better represent reality.

Second Testing Round - Questionnaire

The second testing round of the serious game was done at a MBO school located in Hengelo, OV in The Netherlands. Contrary to the first testing location, this school did not specialize in (asphalt road) construction primary, but does have a small class of students who specialize in the field of execution and have had some classes in asphalt road construction specifically. The lecturer for these classes was an executor for many years before he became a teacher.

The sample size for this test was significantly smaller (eight students total) and the game was also tested by the teacher himself. However, due to circumstances, not all participants filled in the questionnaire. As with the former test location, this test was held at a computer room. The testing date was April 10, 2017.

Technical & Design Issues

Although most of the technical issues were solved in the iteration step, some of the issues persisted and some new problems were unwillingly introduced. The performance related issues were very much solved, although the game still ran significantly slower than expected, it was much better than the first test. The game ran more stably and the frequent hiccups during heat rendering were gone.

The game still rendered in revers at some points in time, but a solution was found during testing where, on higher settings, the game did not mirror its rendering. If a setting higher than and including ‘great graphics’ was selected, the rendering was not reversed. Therefor on all test machines, the game was started on the ‘great graphics’ setting.

A problem that was introduced and found during the test, was that on some systems the camera would be unintentionally move when dragging the markers for the

compactors. This made it hard to place the markers on the intended location as the camera moves to the side when trying this.

Questionnaire Results

The questionnaire for the second round was not changed from the first round of questions that was done at SOMA. The sample size for the second test was three responses out of eight testers. Some testers did not have the time to also fill in the questionnaire after playing the game. Some feedback was given orally when they left the testing area.

According to the survey the testers answered to play either, more than once per week games (33.3%) or hardly play games at all (66.7%). None of the testers answered to have played serious games before.

Understandability of the Game

On the question on how understandable the main menu was at the beginning of the game, two participants answered five and one tester six out of seven on a Likert-scale. The difficulty to find the menu with the levels and the tutorial was noted as (66.7%) three and 33.3% six on the scale from 'very difficult' to 'very easy' to understand.

Figure 43: Likert Scale – How hard it was to find the levels from the main menu

The understandability of the planning phase was noted as a 4.67 mean. All participants filled, for questions that were asked during the planning phase, as 'once in a while'. None of the participants had any remarks on the planning phase.

Figure 44: Likert Scale - How well the planning pase was explained

The understandability of the execution phase explanation was stated as 3.33 mean on the same Likert-scale. Two participants stated that they asked ‘a lot of questions’ and one stated that they asked questions ‘once in a while’.

Game Input

The camera control was answered as ‘not clear’ by 66.7% and ‘clear’ by 33.3%. All testers answered that they liked to control the zoom level by scrolling. The clearness for the selection of the machines the average was chosen as 3 out of 7 on the Likert-scale. The ease of operating the compactors was chosen as 1.33 mean. A graphic of the data can be seen in figure #. The control of the time scaling and pausing mechanic was chosen as 4.67 mean. This figure can be seen in figure #. The following remarks were made on the execution phase:

- Has yet to be optimized
- If you want to change the location of the compactors, the camera moves with the dragging of the waypoints

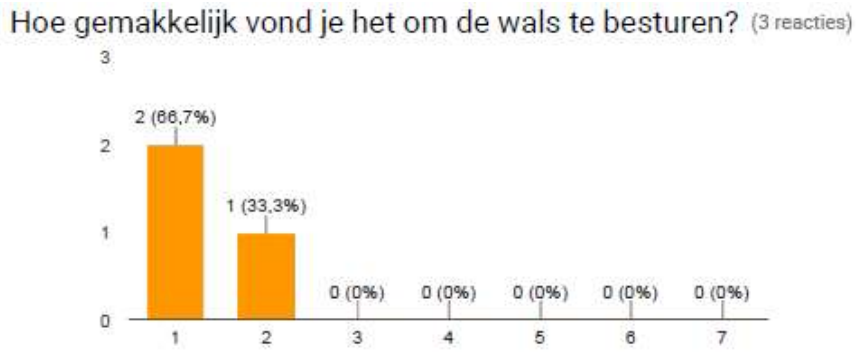

Figure 45: Likert Scale - How easy the compactor was to operate

Figure 46: Likert Scale - How clear the time controls were

Level Progress

All students noted that they reached at least the end of the tutorial and one player played level 3. The other two players played the tutorial and level 1. One of the players answered 1 while the other two answered 6 and 7 on how difficult they thought it was to finish the levels. On the clearness of the goals of the levels, the testers answered 2, 4, and 7 on a scale to 'very unclear' to 'very clear' with an average of 4.33.

Learning Related Questions

A 100% of the students answered that they had 'average knowledge' on asphalt compositions, machine differences and planning during asphalt road construction. On the question, in what capacity the game had given insight into the asphalt road construction processes the mean answer was 2.67 with the results represented in figure #. To the question, in what capacity the game has provided the testers with new insights into the asphalt paving process, the testers answered a mean of 3.67.

Figure 47: Likert Scale - Perceived amount of new insights into paving process

The question 'in what capacity did you know the effects of temperature on the compaction process' was answered as 4.33 mean, 5 mode and 5 median. In the question on how much more the game added to the knowledge on asphalt cooling and the effect of temperature of compaction, was 2.33 mean.

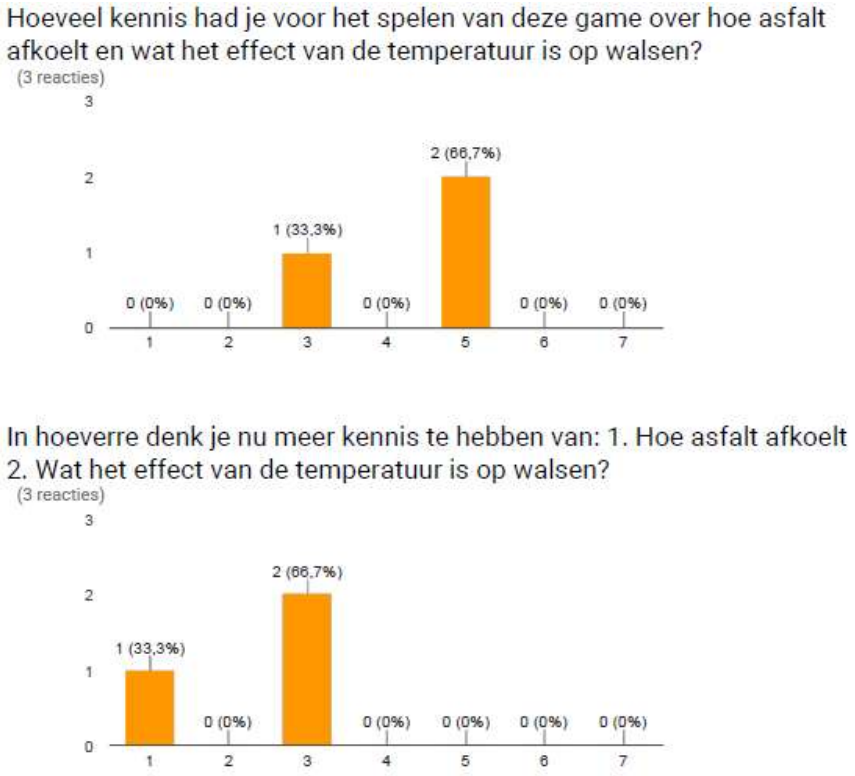

Figure 48: Likert Scale - Questions on knowledge before and gained knowledge on temperature differences on the compaction process

Testers were also asked on how much the game has thought the users on what roles specific machines have within the asphalt construction processed. This question was answered, the mean of these Likert-scales was 3. On the question on how realistic the game portrayed the asphalt construction process between ‘very unrealistic’ and ‘very realistic’ the mean of the choices was 4.67.

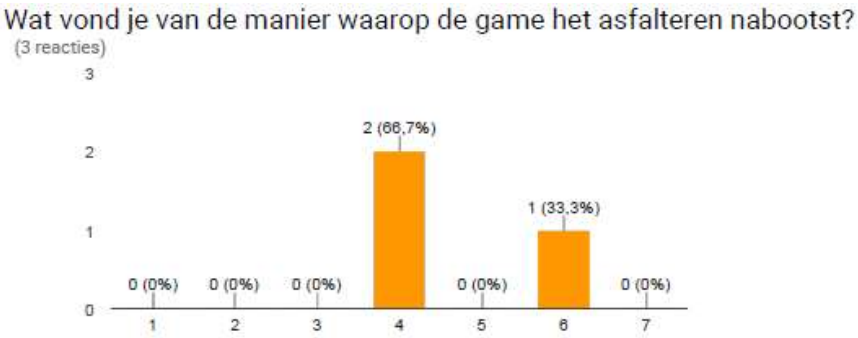

Figure 49: Likert Scale - Realism of the asphalt process simulation

Some remarks that were made at the end of the section on learning related questions where:

- It [the game] portrays a realistic view on how asphalt road construction works in real life
- The explanation was educational

- Some technicalities related to temperature have to be improved

Questions on Fun Factor

The first question in the questionnaire that was asked in this section asked testers to give an estimated play time they thought to have spent in the game. The testers could chose between ’5 and 10 minutes’, ’10 and 20 minutes’, ’30 and 45 minutes’ and ‘more than 45 minutes’. Two testers answered to have spent between 10 and 20 minutes, one tester said to have spent between 30 and 45 minutes. The average motivation that the testers said to have felt, was answered as 3.67, a figure can be seen in figure #. Between ‘not fun’ and ‘very fun’ the mean of the answers given was 2.67, the answers can be seen in figure #. Two players answered not wanting to replay the game, one answered that they might replay the game.

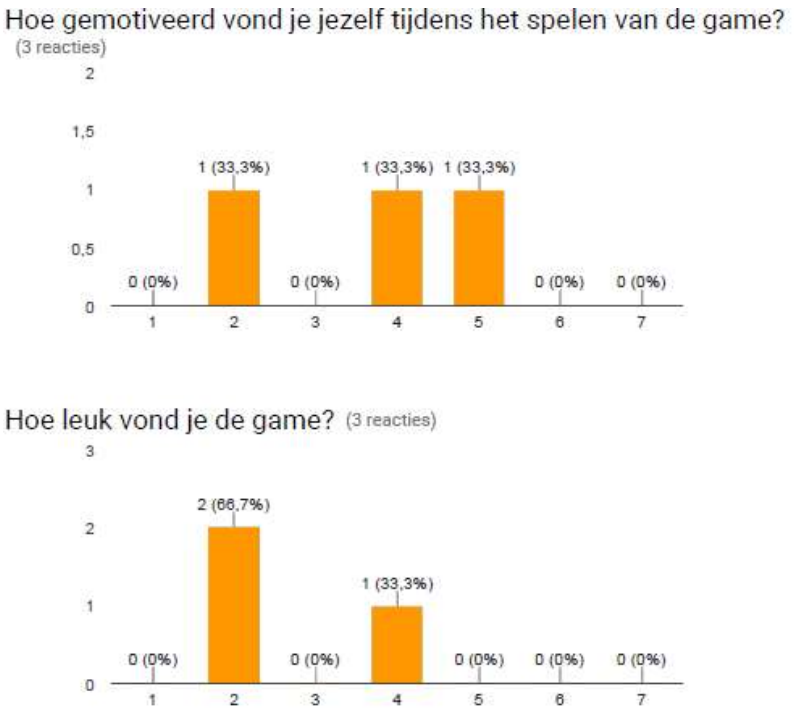

Figure 50: Likert Scale - Motivation and fun of the game

Remarks made by testers at the end of the questions on fun and motivation were:

- [The game does] Not [work] very optimal
- It [the game] didn’t work very well
- There are some problems [in the game] and you can’t do much yourself

Remarks

At the end of the survey, testers could leave any final remarks on topics that they didn’t think were handled yet, or remarks of which they did not feel there was a place for. The remarks were:

- Be more like farming simulator
- [The game] doesn't run very smoothly and the learning points do not come up as much

Besides the remarks left at the end of the questionnaires, some students made the remarks after the testing was done. These include, making the game more like farming simulator; the fact that the difference in temperature are not as clear as they could be; that the game should be played more than once; that a scenario creator could be useful teachers to teach them on specific scenarios; that this scenario editor can be used to train students daily for fifteen minutes over the stretch of a month.

Second Testing Round - Analytics Data

Besides the questionnaire, analytics data gathered by the game was also recorded. The analytics system was built to record the following data:

1. The date and time of the play session
2. Timers for all sections (scenes in Unity) of the game: main menu, planning phase, execution phase and feedback phase
3. Which buttons were pressed in the game and how often
4. Percentages for: average compaction, uncompacted asphalt and excess compaction, on each of the levels

The tabular data can be found in appendix 6. A summary of the results can be found in this section.

The time users spend in the main menu was an average of 3.75 minutes. This is as high as most users had their menu open during the explanation of the session. Most buttons in the game were only pressed once. User spent 3.05 minutes on average in the planning section of the tutorial. 13.30 minutes were spent on the execution tutorial on average and 0.48 minutes on the feedback tutorial.

Level one was played 1.07 times on average, where one student played level one twice. The average time spent in the planning phase of level one is 1.50 minutes, the execution time duration as 5.91 minutes. The average compaction percentage lay at 26.84%, with an uncompacted average of 65.37% and an excess compaction (or compaction on too hot asphalt) of 29.87%. Five testers played level one. One player did not play any of the levels.

Level two was played once at maximum by all players. Players who played level two spend an average of 1.36 minutes in the planning phase, 13.29 minutes in the execution phase and 0.15 minutes in the feedback phase. The average compaction for level two is 21.98%, 36.06% was not compacted and 59.23% was too much compacted (or too early). Three players played level two.

Level three was only played by one tester. A duration of 0.10 minutes was spent at the planning phase, 4.15 minutes in the execution phase and no data was recorded for the feedback phase (it was not reached by the only player who entered level 3). Average

compaction for level 3 is at 4.60%, where 95.40% was uncompacted and 4.60% was compacted too early or overcompacted.

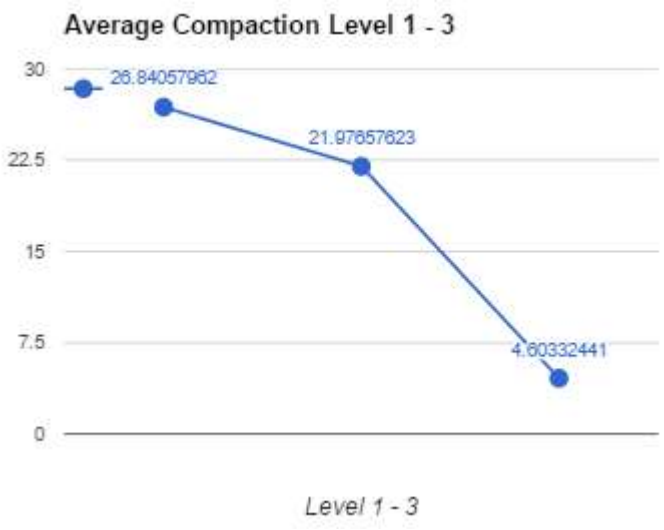

Figure 51: Average compaction per level

During the planning phase, testers returned to the goal/flavor text screen an average of 1.55 times. Most testers only returned once. Six times was the most often that was returned to the flavor/goal screen from the planning input screen, this was done just by one tester.

The average total amount of time spent in the game was 48.91 minutes. An average of 1.50 minutes was spent in the planning phase, 8.45 minutes in the execution phase and 0.32 minutes in the feedback phase. Figure # and # show the progression in average time taken for the execution and planning phases.

Figure 52: Average time spent in planning phase

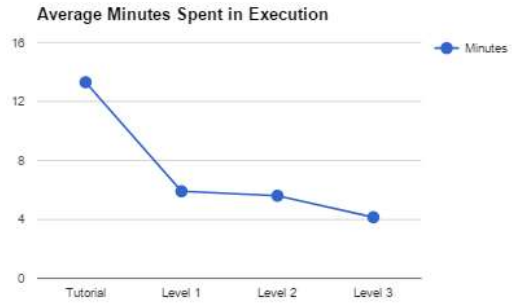

Figure 53: Average time spent in execution phase

Discussion

Important Notice

First, before all other points, it must be noted that the sample size for the second test is far too low to derive any meaningful conclusions from the data when comparing with the original test. Therefore, take this into account when reading the discussion hereafter.

Discussion Questionnaire Results

It is clear from the results that most of the tested students play games weekly and that the students attending at the ROC seem to play games less often than students at the SOMA, however this is hard to say. With this amount of experience on how to play digital audiovisual games, it can be comfortably assumed that this was not a deciding factor in the operation of the game.

It seemed that the user interface of game was quite clear to the users. As the majority of students answered to understand the interface as they answered 5 on a scale of 7 on totally unclear to totally clear. This is reinforced by the perceived amount of questions asked during the phases, where most students answered to ‘sometimes ask questions’. There was also no significant difference (5%) between the two testing locations and dates. The difference was not large enough to contribute these to the improvements made to the user interface. This difference was most likely due to differences in students sampled.

Through observation, the amount of questions asked by both groups was more often than ‘once in a while’. As many questions were asked by the students during the tutorial as some were stuck on a particular section of the tutorial. But once students completed the tutorial, they did not tend to ask questions during the level sections. It is also important to note that students from the ROC, consistently answered to find the execution phase less clear than the SOMA students with a difference of 14.3%. This might be due to the difference in experience playing games in genera. However not many students at the ROC asked themselves what they should do, this could be due to the added goal and flavor text screen featured in their version of the game.

The camera operation was noted as less clear by students at the ROC, this could also be due to the difference in video game experience. Another explanation could be that the waypoint control was not working correctly. However, it is more likely that the difference is due to the sample size. For scroll wheel operation the difference in perceived clearness by the students at SOMA and ROC might be significant, as the median changed from unclear to clear at the later test. This could be due to the fact that the camera no longer had to be moved by pressing the scroll wheel, or the improved performance of the game. What is quite strange is that the students at ROC, who got an improved version, filled in to like the selection of machines less than the students at SOMA, even though selection options were added and the inverse rendering and selection issues were solved. The most likely reason behind this difference is the fact that it was much harder to control the compactor waypoints as the camera kept moving when dragging the markers.

The time input control was noted, in both cases, as above average (5) and there was a negligible difference of 5.6% in favor of the SOMA students on clearness.

The average level that was reached by students was level 1. Some students reached level 2 and only one student reached level 3. This was due to the fact that only the second iteration of the game allowed students to go further than level 2. Most likely the motivation and lack of diversity in the levels in the first testing round at SOMA was the cause for the low percentage of students who reached level 2.The clearness of the goals in the levels was increased by 12% at the ROC, this is most likely due to the addition of flavor texts and goals in their version of the serious game. This might suggest that the goal system was a good idea to implement.

The most often stated playtime both at SOMA and at ROC, was between ten and twenty minutes. The mean perceived time spent at ROC was 10.7% higher than at the SOMA. The stated motivation of the students while playing the game increased just by 3.57% (negligible) from SOMA to ROC, but was still only 3.45 averages combined. This could mean that the improvements made to the game between test date 1 and test date 2 were not nearly enough to push the enjoyment of the game. The fun factor of the game even decreased at the ROC compared to SOMA by 5.57%, which isn't much, but an increase was expected as much of the games bugs had been removed and the frame rate and overall polish of the game was increased considerably. Besides the change in percentage, in both cases the rounded mean of fun perceived during play was noted as 3 out of seven on the Likert-scale. Due to this reason, testers probably didn't want to replay the game. This was the most selected answer.

Most students answered to have superficial knowledge on asphalt road construction, but felt that the serious game could not add to their insights into asphalt road construction in its current form (2.65 mean). They also noted that the game did not add to their understanding of asphalt road construction, although there was an increase between the ROC and SOMA of 7.14%, from 3.18 to 3.67. This increase is probably due to the sample size difference. The lack of increase after the game is probably due to the fact that students already feel that they have a firm grasp on the asphalt construction processes (4.8).

What is interesting to see is that students from SOMA said to have gained significantly more (17.9%) knowledge on the effects of temperature on compaction than the students at the ROC. However, the students at SOMA had an average of 3.59 from 'no additional knowledge' to 'a lot of additional knowledge' on a 1-7 Likert-scale. The game also didn't add any knowledge on the usage of different kinds of machines in asphalt road construction. Remarkable is that students at the SOMA thought that the game was way less realistic (2.65) than students from the ROC (4.67) (an increase of 28.9%). This might be due to the fact that the speed of the paver was changed to be more realistic and the effects of the temperature change might be more visible when the game ran significantly better. This might have also allowed students to have focused on the games systems instead of its flaws.

Four students noted that they thought that they said that they thought that seeing the temperature and the point when compaction should happen most educational. Three students said that the whole game was educational. Remarkably, one student made the remark that the technicalities regarding temperature would have to be improved. Two

students thought that the game was quite realistic at this point. It is interesting to see that, what the testers filled in on the Likert-scale, and what they say when you give them the opportunity to make remarks at the end of sections are completely different and almost contradictory.

Discussion Analytics Results

From the analytics results can be seen that clearly less time is spend on each consecutive iteration of the level flow (planning → execution → feedback). However, it does seem to mellow out when looking at the execution phase's graph in figure #. The reason why this can't be seen is that, for level 3, no flavor text was written, reducing the time considerably.

What is interesting to see is that players noted to have played between 10 and 20 minutes, while when looking at the data from the in game analytics, testers, on average, spend more than 45 minutes playing the game.

It can also be seen that the compaction percentage dropped of after each level played. The reason could be the falling interest level in the game, the difficulty of the levels or a combination of the two. The average compaction also isn't close to what it should be able to achieve in the game, with the maximum compaction reached during the testing was 63.35% where it should have been near the 90th percentile. Reasons why this percentile was not reached is probably due to two things: one being that it is hard to see how green the asphalt should be when it is sufficiently compacted, and two being that the compactors, when the game lags and/or is sped up, don't compact the edges too well. In hindsight it would have been nice to view the compaction textures as analytics as well. With this information, it would be easier to conclude why the compaction levels of students are so low.

Conclusion

The Research Questions

In review, the research question that was asked in the beginning of this report, namely:

"How can an effective serious game be developed for teaching students to make better decisions in both the planning phase as well as during the construction work, based on both real-time data and environmental factors?"

, and its two sub-questions:

"How could the cause effect relationships between external factors and the quality of the asphalt be best taught by means of this serious game?"

"What type of interactions with the simulation are best suited to train students to make better decisions based on real-time weather, temperature and geolocation data in the real word?"

can not be clearly answered by the resulting test data, as students described not to have learned much by playing the serious game. The main question, how effective serious games can be developed for teaching students cause effect relations in the asphalt road construction can also not be clearly answered, as this attempt was not as successful as was initially hoped. It is hard for students to focus on the educational value of a game when the systems which it tries to simulate are not very realistic due to technical problems. The same goes for the enjoyment of the game.

Creating an effective serious game that is both fun and simulates real life experiences is a difficult. Creating compelling audiovisual games is very hard and time consuming; creating them for educational purposes is even harder. Although the potential for using (even this) serious games in asphalt road construction is clearly visible and noted by industry experts and teachers alike, its potential was not fulfilled during the duration of this project. The game needs a lot of work and polish before it could be deployed and used by teachers and companies alike.

However, from the literature this potential is also supported. The experience and scenario based learning techniques described in previous work is implemented here. This suggests that this form of serious game / simulation could be useful when it is implemented fully.

It would be interesting to see, how the game would perform with students, if the technical difficulties are fixed and the asphalt and weather simulations are made more robust and finalized. As much of the opinions of students for liking the game and educational value were based on the game not properly functioning at the time of testing, biasing the findings as a result. However, the tutorial in the game was effective at explaining the game as students asked very little questions after they had all completed the tutorial.

A test when the game is finalized would also allow for comparing the analytics data. This could be used as another way to test improvements to the game and the levels that are played.

Reflection & Limitations

Developing a working and functional serious game that accurately simulates asphalt road construction mechanics and concepts was way more work than expected. I greatly underestimated the time that it would take to develop this concept generated in the idea phase of the project. Especially considering my extensive personal experience with creating traditional videogames. In hindsight I should have known, from past experiences, how much work it would be, and should have chosen either a greatly reduced concept or scaled back the game a lot.

For a team of one person the amount of work needed for creating a fully functional game with pleasant looking art direction, designing compelling and realistic gameplay, programming all the systems and simulations, creating tools for testing, debugging and making changes at the request of users and project stakeholders, was far to much work. In the planning for the project, three weeks were suggested for the implementation of the bachelor project, this period was not nearly enough and more than twice that amount of time was put into the game. I would suggest to, for (serious) game creation to assign at least two people to the same project, or change the projects scope to create a board game, or not allow for games to be created as bachelor assignments in general.

In regard to the testing phase of the project, the school chosen for the second test was not the best decision as the sample size at this school was too low to begin with. The class sizes at the ROC who study asphalt road construction are just too small for decent sample sizes. It would have probably be more useful to test with the same group or test at a school with more students such that the results of the questionnaire could have been better compared.

Compared to the rest of the game, the levels should have received more attention. As the differences in how the levels feel are not compelling enough for students to play. Maybe even a scenery change would have been enough for students, at least on a surface level, to be compelled more to be motivated to play the level as best as possible.

The feedback system that was used could also have used more thought and testing. The feedback given in percentages and numbers and how they relate to the amount of stars was not explained well in the interface, making it hard for students to see their progress. The earlier levels should also have been shorter (in the range of a few meters), getting the time between feedback and play shorter for students to see the effects of their play more quickly. Later on, levels could be made longer, as the player already knows what to do, and what not to do. The jump between teaching the students how to operate the machines and how to compact, and the time at which they learn how weather effects compaction, was way to short. In addition to this, the time speed up mechanic could have also been introduced later. This would have forced the

user to think more deliberately on the operation as the game would move at a slower phase.

Related to this is the readability of the game's mechanics. In the current version of the game, the visualization for the compaction of the asphalt is not very readable. The game uses a gradient from black to green to show how well compaction is done. Maybe more discrete values should have been used for the amount of passes, or the color should have become green only when the compaction is sufficient, while using another color to show the gradient of compaction values. Now it is quite hard to see why the compaction did not turn out as well as hoped. These changes would also allow the users to get better feedback during the feedback phase as the same texture is used there to show the final compaction.

This project has also taught me to always try to develop with a target machine. The amount of work that was spent into optimizing the game for the target platform was huge. Maybe this could have been prevented if a reference machine could have been obtained during development. Such that testing on the target hardware could have been done more often. This would most likely have saved a lot of time on optimizing as the problems would have been noticed earlier, allowing for changes in the game's design at an earlier stage.

The naming of the game was also not very wise, as this might have set unrealistic expectations with students as it might have prompted them to make comparisons to a commercially developed simulation game. Changing the look of the game might have also done this unintentionally. Maybe if the game's menus and title were chosen to look less 'finished' the interactions of the testers with the game might have been more focused on the systems instead of the technical issues. However, this is hard to say, as this effect is not documented and proven well.

Besides the testing and technical difficulties, there were also some theoretical roadblocks that were encountered during the development of the serious game. The lack of knowledge on thermal dynamics, hard data on the effects of weather on asphalt and the effects of compaction on the cooling of asphalt made it hard to accurately create models for these dynamic systems. This in turn made it very difficult to implement these systems into an accurate and real-time simulation. This could have been circumvented if I more actively searched for experts in these fields to help me figure these problems out. This would have resulted in a more accurate model and in my own grasp on the material I was tasked to teach. The same goes for generating well designed feedback and implementing the best ways of teaching the intended mechanics and reaching the learning goals, as my knowledge on education is also very limited.

Future Work

From the conclusion, it is clear that a lot needs to be done in order to effectively test this serious game. The simulation of the asphalt and the temperature characteristic changes with weather effects is not accurate enough to enable learning effects to be measured. Dynamic weather changes and accurate data sets of these changes are the first steps that need to be taken in order for the serious game to teach the intended

learning goals. Either, accurate models for thermodynamics on the effects of precipitation and environment temperatures on asphalt, or a very large and accurate dataset are required to achieve this goal.

Thereafter, a tool needs to be developed to allow teachers to develop tailored scenarios for students to play. It has been suggested that shorter, but more frequent sessions with the game might be more successful in achieving the learning goals and more desirable for teachers. A powerful tool which is easy to use by teachers would help achieve this goal. A tool like this is in development by another student at the University of Twente as of the writing of this document.

Another feature which could improve the decision making is the addition of asphalt delivery trucks. When these are simulated, the user needs to keep into account both the time in which the player is able to compact the asphalt, but also the speed of the paver as it needs sufficient asphalt in order to pave. This gives the player another challenge as he needs to base the speed of the paver on two parameters instead of one. Scenarios made with these two mechanics allow the designer of the levels to design for more interesting decisions that the player/learner needs to make.

- Implementation of distance to paver based compaction areas for compactors.
- Curved routes and multiple lanes of paving.
- Implement odd road construction scenarios for crossings, highways, roundabouts and highway insert lanes, etc.
- Creating optimized libraries in c/cpp that do the rendering
- Use (compute) shaders to calculate the heat dissipation
- Interaction between compaction and heat dissipation simulation
- Fix movement patterns for compactors as they currently do not compact evenly on the asphalt.
- Add multiple compaction strategy selection options

Table of Figures

References

[1] *Keuzedelen," Beroepsonderwijs bedrijfsleven,* 2016.

[2] J. P. Mahoney, G. M. Turkiyyah, S. Muench, T. N. (Organization) and U. S. D. of Transportation. Office of the Secretary, Transportation Infrastructure Design and Construction: Virtual Training Tools, Transportation Northwest, Department of Civil Engineering, University of Washington, 2003.

[3] J. P. Mahoney, *Pavement Related Internet Applications - PowerPoint PPT Presentation,* 2005.

[4] V. Center, "A Virtual Environment tool to review activities of construction machine's and demonstrate alternative working strategies," 2013. [Online]. Available: https://vimeo.com/85919112 . [Accessed 20 November 2016].

[5] C. International, "bottlenecks," CCT International, [Online]. Available: http://www.cctintl.com/solutions/modeling-and-simulation. [Accessed 20 November 2016].

[6] "Public Relations," Unity Technologies, [Online]. Available: https://unity3d.com/public-relations. [Accessed 20 November 2016].

[7] Unity, "multiplatform," Unity Technologies, [Online]. Available: https://unity3d.com/unity/multiplatform. [Accessed 20 November 2016].

[8] "editor," Unity Technologies, [Online]. Available: https://unity3d.com/unity/editor. [Accessed 20 November 2016].

[9] "Unreal Engine Features," Epic Games, [Online]. Available: https://www.unrealengine.com/unreal-engine-4. [Accessed 20 November 2016].

[10] "Unity Serious Games Showcase," Unity Technologies, 6 November 2013. [Online]. Available: https://unity3d.com/learn/resources/unity-serious-games-showcase. [Accessed 20 November 2016].

[11] R. Garris, R. Ahlers and J. E. Driskell, "Games, motivation, and learning: A research and practice model," *SIMULATION & GAMING,* vol. 33, pp. 441-467, 2002.

[12] D. Cowley, "SOARTECH AND EPIC GAMES ANNOUNCE OFFICIAL PARTNERSHIP FOR GOVERNMENT AND MILITARY APPLICATIONS," Epic Games, 1 December 2015. [Online]. Available: https://www.unrealengine.com/news/soartech-partnership-government-military-applications. [Accessed 20 November 2016].

[13] J. Gaudiosi, "HUMANSIM: VIRTUAL HEROES MAKES SERIOUS GAMES," Epic Games, 16 February 2011. [Online]. Available: https://www.unrealengine.com/showcase/humansim. [Accessed 20 November 2016].

[14] C. E. Catalano, A. M. Luccini and M. Mortara, "Best Practices for an Effective Design and Evaluation of Serious Games," *International Journal of Serious Games,* vol. 1, no. 1, 2014.

[15] M. Csikszentmihalyi, Flow: The Psychology of Optimal Experience, First Edition ed., Harper Perennial, 1991.

[16] C. Harteveld, R. Guimaraes, I. Mayer and R. Bidarra, "Balancing play, meaning and reality: the design philosophy of levee patroller," *Simulation and Gaming,* vol. 41, pp. 316-340, 2010.

[17] C. Harteveld, Triadic Game Design, 1 ed., Springer-Verlag GmbH, 2011.

[18] C. R. Rogers and H. J. Freiberg, Freedom to learn, 3 ed., Maxwell Macmillan International, 1994.

[19] J. Profijt, *Concept Uitwerking Kuezedeel Innovaties in de Asfaltwegenbouw,* 2016.

[20] S. B. W. opleindingscentrum Infra and VBW-Asfalt, *De asfaltuitvoerder,* 2002.

[21] B. Molkenthin, *Software Rasterization Algorithms for Filling Triangles,* 2012.

[22] I. Mayer, G. Bekebrede, C. Harteveld, H. Warmelink, Q. Zhou, T. van Ruijven, J. Lo, R. Kortmann and I. Wenzler, "The research and evaluation of serious games: Toward a comprehensive methodology," *British Journal of Educational Technology,* vol. 45, pp. 502-527, 2014.

[23] A. Frunze, *Answer to: "I need a pixel-perfect triangle fill algorithm to avoid aliasing artifacts",* 2014.

[24] C. Franzwa, Y. Tang and A. Johnson, "Serious Game Design: Motivating Students through a Balance of Fun and Learning," in *2013 5th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, 2013.

[25] D. Clegg and R. Baker, CASE Method Fast-track: A RAD Approach, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©1994, 1994.

[26] C. E. Catalano, A. M. Luccini and M. Mortara, "GUIDELINES FOR AN EFFECTIVE DESIGN OF SERIOUS GAMES," *International Journal of Serious Games,* vol. 1, 2014.

[27] L. Busche, *The Skeptic's Guide To Low-Fidelity Prototyping,* 2014.

[28] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal,* vol. 4, pp. 25-30, 1965.

Appendix 1 - Game Genre Based Ideas

- Sim City Like-Game
- Manage money and time resources
- Build factories and hire machines and employees
- Manage dynamic disasters that destroy the roads you build
- Maintain quality over time
- Text Adventure Like-Game
- Dialog options to keep up worker morale
- Enter values and estimations before execution
- Events pop up where you need to intervene, your adeptness to solve the problems bases your score
- Turn Based Strategy Like-Game
- Control all machines that are present on the build site.
- Plan routes for a fixed time frame, then see the results in real time
- Keep this pattern up until the production is finished
- Mediate feedback after one time frame
- Real-Time Strategy Like-Game
- Control all machines in real time
- Broad bird's-eye overview of the entire project
- React to changes in weather in real time and solve issues
- First Person Shooter Like-Game
- First person view of an executor in asphalt road construction
- Shout commands to your subordinates and direct their behavior
- No bird's-eye overview, walk around to see everything
- VR-ready?
- Graphic Adventure Like-Game
- Switch locations and talk to all your subordinates
- Manage your relationships and business associates
- Switch between a variety of locations to manage issues and unforeseen problems. i.e. Talk to the asphalt factory to produce more asphalt, hire more trucks and keep your employer up to date on the progress
- Walk in the scene towards objects and people to interact in a side scrolling section

- People move trough the scene dynamically as in games like the old LucasArts games of old
- Visual Novel Like-Game
- Much of the same mechanics like the graphic adventure game, but without the dynamically walking characters; scrolling backgrounds and other visual flair. More low budget friendly.
- Turn Based Strategy Like-Game
- A turn based game where the simulation is done in turns.
- Each turn you are able to plan the execution of the road construction where you can see the results of that turn.
- You are in control of the entire supply chain. From ordering machines, to controlling the machines on site.
- During a turn you are able to: order machines, change machine speeds, order asphalt per for the next few turns, and change the direction and todo's of the entire crew.
- ASPARi Simulator 2017 / Road Building Tycoon
- Manage your own road construction firm
- Acquire progressively more difficult jobs with increasingly higher requirements
- Let the company survive as long as possible
- See the effects of road compaction on the long term
- Share resources between simultaneous projects
- Hire or buy new machines, employees and equipment
- Buy asphalt mixtures at wholesale from catalogs and spread it out on your projects
- Be as productive and as profitable as possible
- Asphalt Construction Rhythm Like-Game
- Push the correct buttons at the correct time to pave the best road you can
- Answer questions about road construction on the beat and BEAT your opponents scores!
- Road Paving Racing Game
- Pave as much asphalt as possible at as quick of a rate
- Scores are based on the quality, quickness and correct quantity of asphalt used
- Switch between pavers and compactors to achieve the correct compaction
- Race against others in real time to see how the other players perform
- Different tracks have different weather conditions to let the players experience their difference in cooling time

Appendix 3 - Example Level

```
// Level parameters
naam: "Level 1"
planning_sterren: 2
uitvoering_sterren: 0
benodigde_sterren: 1

// Planning parameters
intro_text: "In dit level ga je voor het eerst je eigen stuk asfalt aanleggen. Hierbij hoef je alleen de toplaan te asfalteren. Het doel van dit werk is de verdichting zo hoog mogelijk te krijgen, zonder het asfalt te veel te verdichten. Je zou naar 90% verdichting moeten strefen.\n\nHet asfalt wat we gaan draaien moet de volgende afmetingen hebben: 5 meter breed, 5 cm dik en een halve kilometer lang.\n\nDe machines zijn al voor je geselecteerd, maar je hebt beschikking over een spreidmahcine, en twee statische walsen.\n\nHeel veel succes!"
kan_machines_kiezen: false
spreidmachines: 1
statische_walsen: 2
dynamische_walsen: 0
vrachtwagens: 2

// weg afmetingen
kan_afmetingen_kiezen: false
weg_breedte: 5
weg_dikte: 5
weg_lengte: 0.1
weg_type: SMA

[weather]
temperatuur: 18
bewolkt: false
regen: false
zonnig: true
sneeuw: false

[goals]
minimum_verdichting: 50.0
maximum_verdichting: 99.0
gemiddelde_verdichting: 50.0
maximale_duur_uitvoering: 2700.0
```

Appendix 4 - Temperature Table

-5 C	0 C	10 C	20 C
149	149	149	149
141.135	142.521	142.948	143.377
136.289	138.206	138.915	139.628
132.583	134.861	135.789	136.721
129.437	132.027	133.138	134.256
126.628	129.503	130.778	132.06
124.052	127.191	128.615	130.048
121.652	125.036	126.599	128.172
119.391	123.007	124.701	126.405
117.249	121.082	122.899	124.729
115.208	119.247	121.182	123.13
113.256	117.49	119.538	121.6
111.385	115.804	117.96	120.132
109.588	114.182	116.443	118.719
107.858	112.62	114.98	117.358
106.19	111.112	113.569	116.045
104.58	109.655	112.205	114.776
103.025	108.245	110.886	113.548
101.521	106.881	109.609	112.359

100.065	105.558	108.371	111.206
98.655	104.275	107.17	110.089
97.287	103.029	106.004	109.004
95.96	101.819	104.872	107.95
94.672	100.643	103.772	106.926
93.421	99.5	102.701	105.929
92.205	98.387	101.66	104.96
91.023	97.303	100.646	104.016
89.873	96.247	99.658	103.096
88.753	95.218	98.695	102.2
87.663	94.215	97.756	101.326
86.6	93.236	96.839	100.474
85.565	92.28	95.945	99.642
84.555	91.347	95.072	98.829
83.571	90.436	94.22	98.036
82.61	89.545	93.386	97.26
81.672	88.675	92.572	96.502
80.756	87.824	91.775	95.761
79.861	86.991	90.996	95.036
	86.176	90.234	94.326
	85.378	89.487	93.631

	84.596	88.756	92.951
	83.831	88.04	92.285
	83.081	87.338	91.632
	82.346	86.65	90.992
	81.625	85.976	90.364
	80.918	85.314	89.748
	80.224	84.665	89.145
	79.544	84.029	88.552
		83.404	87.97
		82.79	87.399
		82.187	86.838
		81.595	86.287
		81.013	85.746
		80.441	85.214
		79.879	84.691
		79.326	84.176
			83.67
			83.172
			82.683
			82.201
			81.726

			81.259
			80.799
			80.346
			79.899
			79.459
			79.026

Appendix 5 – Test Questionare (Dutch)

Vragenlijst Asfalt Simulator '17

Hier volgt een vragenlijst met 32 vragen over de serious game die je net hebt gespeeld. Neem de tijd en beantwoord de vragen zo goed en uitgebreid mogelijk. Aan het einde is er plaats voor algemene opmerkingen. Bij vragen, kun je me altijd aanspreken voor verdere uitleg mocht iets niet duidelijk zijn. De antwoorden zijn en blijven volledig anoniem dus wees vooral eerlijk.

Algemene vragen

Hoeveel ervaring heb je met games spelen? (Bijvoorbeeld op je telefoon of op een (spel)computer)

- Ik speel nooit games
- Ik speel bijna nooit games
- Ik speel elke maand wel eens een game
- Ik speel wekelijks games
- Ik speel veel games (meer dan een keer per week)
- Other…

Hoeveel ervaring heb je met games in het onderwijs?

- Dit was de eerste keer dat ik een game heb gespeeld voor het onderwijs
- Ik heb wel vaker games gespeeld voor het onderwijs
- Ohter…

Vragen over de speelbaarheid van de game

Waren de menu's aan het begin van het spel duidelijk?

	1	2	3	4	5	6	7	
Heel onduidelijk	O	O	O	O	O	O	O	Heel duidelijk

Hoe moeilijk vond je het om het menu met de uitleg en de levels te vinden?

	1	2	3	4	5	6	7	
Heel moeilijk	O	O	O	O	O	O	O	Heel makkelijk

Hoe duidelijk vond je de uitleg van de planningsfase?

	1	2	3	4	5	6	7	
Heel onduidelijk	O	O	O	O	O	O	O	Heel duidelijk

Hoeveel keer heb je je tijdens de uitleg van de planningsfase (af)gevraagd wat je moest doen?

1. Nooit
2. Af en toe
3. Redelijk vaak
4. Ik snapte er niets van

Heb je nog opmerkingen over de duidelijkheid van de uitleg van de planningsfase?

_____

Hoe duidelijk vond je de uitleg van de uitvoeringsfase?

	1	2	3	4	5	6	7	
Heel onduidelijk	O	O	O	O	O	O	O	Heel duidelijk

Hoeveel keer heb je je tijdens de uitleg van de uitvoeringsfase (af)gevraagd wat je moest doen?

1. Nooit
2. Af en toe
3. Redelijk vaak
4. Ik snapte er niets van

Vond je de besturing van de camera duidelijk?

- Ja
- Nee

Vond je het prettig om met het scrolwiel de camera te besturen?

- Ja
- Nee

Vond je het duidelijk hoe je de machines moest aanklikken?

	1	2	3	4	5	6	7	
Heel onduidelijk	O	O	O	O	O	O	O	Heel duidelijk

Hoe gemakkelijk vond je het om de walsen te besturen?

	1	2	3	4	5	6	7	
Heel lastig	O	O	O	O	O	O	O	Heel makkelijk

Hoe duidelijk vond je het om de tijd te versnellen? En hoe duidelijk vond je het om het spel te pauzeren?

	1	2	3	4	5	6	7	
Heel lastig	O	O	O	O	O	O	O	Heel makkelijk

Heb je nog opmerkingen over de duidelijkheid van de uitleg van de uitvoeringsfase?

_____

Heb je nog opmerkingen over de besturing van de uitvoeringsfase?

_____

Vragen over de speelbaarheid van de game

Tot welk level ben je gekomen?

1. Alleen de uitleg
2. Level 1
3. Level 2
4. Level 3

Hoe moeilijk vond je de levels om te halen?

	1	2	3	4	5	6	7	
Heel simpel	O	O	O	O	O	O	O	Heel moeilijk

Hoe duidelijk was het wat je moest doen in een level?

	1	2	3	4	5	6	7	
Heel onduidelijk	O	O	O	O	O	O	O	Heel duidelijk

Vragen over het leren

Hoe schat je je eigen kennis in over het asfalteren voordat je het spel speelde?

- Geen kennis van asfalteren
- Oppervlakkige kennis van het asfalteer proces (weten welke machines aanwezig zijn en wat ze grofweg doen.)
- Redelijke kennis (kennis van asfalt mengsels, verschillen tussen machines en planningsfase)
- Uitgebreide kennis (kennis van temperatuur invloeden op het asfalt, berekeningen die nodig zijn voor asfalteren en kennis van wals strategieen)

In hoeverre denk je dat het spelen van deze game, je inzicht heeft gegeven in het asfalteer proces?

	1	2	3	4	5	6	7	
Geen nieuwe inzichten	O	O	O	O	O	O	O	Veel nieuwe inzichten

In hoeverre denk je dat deze game nuttig zou zijn voor je begrip van asfalteren?

	1	2	3	4	5	6	7	
Helemaal niet nuttig	O	O	O	O	O	O	O	Heel nuttig

Hoeveel kennis had je voor het spelen van deze game over hoe asfalt afkoelt en wat het effect van de temperatuur is op walsen?

	1	2	3	4	5	6	7	
Geen kennis	O	O	O	O	O	O	O	Veel kennis

In hoeverre denk je nu meer kennis te hebben van: 1. Hoe asfalt afkoelt 2. Wat het effect van de temperatuur is op walsen?

	1	2	3	4	5	6	7	
Geen kennis	O	O	O	O	O	O	O	Veel kennis

Hoeveel heeft deze game je geleerd over wat de rollen zijn van de machines binnen het asfalteren?

	1	2	3	4	5	6	7	
Niets	O	O	O	O	O	O	O	Veel

Wat vond je van de manier waarop de game het asfalteren nabootst?

	1	2	3	4	5	6	7	
Onrealistisch	O	O	O	O	O	O	O	Zeer realistisch

Wat vond je het meest (of het minst) leerzaam aan de game en waarom?

Plezier van de game

Hoe lang denk je dat je de game gespeeld hebt?

- Minder dan 5 minuten
- Tussen de 5 en 10 minuten
- Tussen de 10 en 20 minuten
- Tussen een half uur en drie kwartier
- Meer dan drie kwartier

Hoe gemotiveerd vond je jezelf tijdens het spelen van de game?

	1	2	3	4	5	6	7	
Ongemotiveerd	O	O	O	O	O	O	O	Zeer gemotiveerd

Hoe leuk vond je de game?

	1	2	3	4	5	6	7	
Niet leuk	O	O	O	O	O	O	O	Heel leuk

Waarom vond je de game wel of niet leuk?

_____

Zou je de game in je eigen tijd nog een keer willen spelen?

- Ja
- Nee
- Misschien

Heb je nog opmerkingen over je ervaring met de game?

_____

Is er verder nog iets dat je wilt vertellen of kwijt wilt over de game?

_____

Bedankt voor het spelen van de serious game en het beantwoorden van de vragen. Ik wens je nog een prettige avond en veel succes met je opleiding!

Appendix 10 - Code

```
----------------------------------
CompactorPropertyManager.cs
----------------------------------

using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class CompactorPropertyManager : MonoBehaviour {

    public Text speedLabelText;
    public Slider speedSlider;
    public RoadCompactor compactor;

    // Use this for initialization
    void Start() {
        speedLabelText.text = "Snelheid " + compactor.speed.ToString("n1") + " Km/u";
        speedSlider.minValue = compactor.minSpeed;
        speedSlider.maxValue = compactor.maxSpeed;
        speedSlider.value = compactor.speed;
        speedSlider.onValueChanged.AddListener((value) => {
            speedLabelText.text = "Snelheid " + compactor.SetSpeed(value).ToString("n1") + "
Km/u";
        });
    }
}
----------------------------------
DataManager.cs
----------------------------------

using System;
using System.Collections.Generic;
using UnityEngine;

public class DataManager : MonoBehaviour {

    private DataSet[] datasets;
    private class DataSet {
        public float   temperature;
        public float[] heat_data;
    }

    private static DataManager _instance;
    public static DataManager Instance {
        get {
            if (_instance == null) {
                GameObject go = new GameObject("Data Manager");
                _instance = go.AddComponent<DataManager>();
                _instance.Initialize();
            }
            return _instance;
        }
    }

    private void Initialize() {
        string file_name       = "heat_data";
        TextAsset file         = Resources.Load<TextAsset>(file_name);
        string[] lines         = file.text.Split(new string[] { "\r\n", "\n" },
StringSplitOptions.None);

        // Get the first row, which contains the headers
        string   header       = lines[0];
        string[] header_parts = header.Split(';');
        datasets              = new DataSet[header_parts.Length];
        for (int column_index = 0; column_index < header_parts.Length; column_index++) {
            string column = header_parts[column_index];

            DataSet data_element = new DataSet();
            data_element.temperature = float.Parse(column.Replace(" C", ""));
```

```
            // First get the amount of data points (line count - empty string count)
            List<float> result_data = new List<float>();
            for (int row_index = 1; row_index < lines.Length; row_index++) {
                string   line = lines[row_index];
                if (string.IsNullOrEmpty(line)) break;
                string[] line_data = line.Split(';');
                if (string.IsNullOrEmpty(line_data[column_index])) {
                    break;
                }
                result_data.Add(float.Parse(line_data[column_index]));
            }
            data_element.heat_data = result_data.ToArray();

            datasets[column_index] = data_element;
        }
        Debug.LogFormat("Done parsing {0}. Parsed into {1} DataElements.", file_name,
header_parts.Length);
    }

    public float[] GetDataPoints(float temperature) {
        float min_found_temp = float.MinValue;
        DataSet min_dataset  = null;
        float max_found_temp = float.MaxValue;
        DataSet max_dataset  = null;

        for (int dataset_index = 0; dataset_index < datasets.Length; dataset_index++) {
            DataSet data_element = datasets[dataset_index];
            // If we found the temperature exactly
            if (data_element.temperature == temperature) {
                return data_element.heat_data;
            }
            if (data_element.temperature < temperature &&
                data_element.temperature > min_found_temp) {
                min_dataset    = data_element;
                min_found_temp = data_element.temperature;
            } else if (data_element.temperature > temperature &&
                        data_element.temperature < max_found_temp) {
                max_dataset = data_element;
                max_found_temp = data_element.temperature;
            }
        }
        // Caluclate intermediate temperatures
        int result_length = Mathf.Min(min_dataset.heat_data.Length,
max_dataset.heat_data.Length);
        float[] result    = new float[result_length];
        float fraction = (temperature - min_found_temp) / (max_found_temp - min_found_temp);
        for (int data_point_index = 0; data_point_index < result_length; data_point_index++) {
            // Lerp between the lowest and highest value
            result[data_point_index] = Mathf.Lerp(min_dataset.heat_data[data_point_index],
                                                  max_dataset.heat_data[data_point_index],
                                                  fraction);
        }
        return result;
    }
}
----------------------------------
ExecutionManager.cs
----------------------------------

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ExecutionManager : MonoBehaviour {

                              public static float simulationTimeScale = 1;
                              public static bool isPaused = false;
                              public static float simulationTimeSinceStart = 0;

                              public float startTime = 0;
                              public UnitSelector unitSelector;

                              [HeaderAttribute("Prefabs")]
```

```
                                public GameObject roadPrefab;
                                public GameObject topLayerPrefab;
                                public GameObject paverPrefab;
                                public GameObject compactorPrefab;
                                public GameObject shakerPrefab;
                                public GameObject truckPrefab;

                                [HeaderAttribute("Generation Parameters")]
                                public float roadDistance = 5f;
                                public float maxRoadWidth = 5f;

                                private GameState game_state;

                                GameObject[] roads;
                                TextureManager[] texture_managers;
                                Transform[] start_positions;

                                [HideInInspector] public GameObject[] pavers;
                                [HideInInspector] public GameObject[] compactors;
                                [HideInInspector] public GameObject[] shakers;
                                [HideInInspector] public GameObject[] trucks;

                                void Awake() {
                                    simulationTimeSinceStart = 0;
                                }

                                // Use this for initialization
                                void Start () {
                                    game_state = GameState.Instance;
                                    Level level = game_state.currentLevel;

#if UNITY_EDITOR
                                    if (level == null)
                                            level = game_state.currentLevel =
LevelLoader.LoadLevelFile("level1");
#endif
                                    // Generate roads
                                    float road_width = level.weg_breedte;
                                    int lane_count = (int)(road_width / maxRoadWidth);
                // TODO(Peter): Add the remainder of the roads to an additonal road strip, but change
the width based on that remainder.
                                    float width_remainder = road_width % maxRoadWidth;

                                    roads = new GameObject[lane_count];
                                    start_positions = new Transform[lane_count];
                                    texture_managers = new TextureManager[lane_count];

                                    for (int i = 0; i < lane_count; i++) {
                                            int offset = 0;
                                            if (i == 0) offset = 0;
                                            else if (i % 2 == 0) offset = (i / 2) * 5;
                                            else  offset = ((i / 2) + 1) * -5;
                                            GameObject road = roads[i] = Instantiate(roadPrefab,
new Vector3(0, 0.05f, offset),

                                                    Quaternion.identity);
                                            start_positions[i] = road.transform.Find("Start
Position");

                                            GameObject top_layer = Instantiate(topLayerPrefab,
new Vector3(0, 0, offset),

                                            Quaternion.identity);
                                            texture_managers[i] =
top_layer.GetComponent<TextureManager>();
                                    }

                                    // Generate others
                                    pavers = new GameObject[level.spreidmachines];
                                    for (int i = 0; i < level.spreidmachines; i++) {
                                            pavers[i] = Instantiate(paverPrefab,
start_positions[i].position,

                                Quaternion.Euler(0, 90, 0));
```

```
                                        PaverController paver =
pavers[i].GetComponent<PaverController>();

                                        paver.road_manager = texture_managers[i];

                                        paver.totalDistance = level.weg_lengte * 1000;
                                        paver.isPaving = true;

                                    }

                                    float vehichle_distance = 5;     // In meters
                                    Vector3 position = start_positions[0].position -
Vector3.right * vehichle_distance;

                                    compactors = new GameObject[level.statische_walsen];
                                    for (int i = 0; i < level.statische_walsen; i++) {
                                        compactors[i] = Instantiate(compactorPrefab,

                                position,

                                Quaternion.Euler(0, 90, 0));
                                        position -= Vector3.right * vehichle_distance;
                                    }
                                    shakers = new GameObject[level.dynamische_walsen];
                                    for (int i = 0; i < level.dynamische_walsen; i++) {
                                        Vector3 random_pos = Random.insideUnitSphere * 10;
                                        random_pos.y = 0;
                                        shakers[i] = Instantiate(shakerPrefab,

                                position,

                                Quaternion.Euler(0, 90, 0));
                                        position -= Vector3.right * vehichle_distance;

                                    }

                                    // Generate Unit selection UI
                                    unitSelector.GenerateUnitSelectionMenu(pavers, compactors,
shakers, trucks);

        // Snap camera to first paver
        FindObjectOfType<FollowTarget>().SetTarget(pavers[0].transform);

                                    startTime = Time.realtimeSinceStartup;
                                }

                                // Update is called once per frame
                                void Update () {
                                    if (!isPaused) {
                                        simulationTimeSinceStart += Time.deltaTime *
simulationTimeScale;
                                    }
                                }

                                public void SetRoadRenderMode(int mode_index) {
                                // 0: road only, 1: heat only, 2: compaction only, 3: heat &
compaction
                                    foreach (TextureManager tm in texture_managers) {
                                        tm.SetRenderMode(mode_index);
                                    }
                                }

                                public void CalculateExecutionResults() {
                                    GameState game_state = GameState.Instance;
                                    TextureManager texture_manager =
FindObjectOfType<TextureManager>();

                                    GameState.ExecutionResults results = game_state.results =
                                        new
GameState.ExecutionResults(texture_manager.heatTexture.width);
        results.compactionTexture = texture_manager.compactTexture;

                                    results.totalTime = Time.realtimeSinceStartup - startTime;
```

```
        AnalyticsManager.AddData(game_state.currentLevel.naam + "_" + "total_execution_time",
results.totalTime);

                                // Calculate average value
        float[] compaction_values = texture_manager.compactValues;
        float compaction_sum           = 0;
        int excessive_compaction_count = 0;
        int uncompaction_count         = 0;
        int compaction_point_count     = 0;
        float prev_sum                 = 0.0f;
        int lane_index                 = 0;
        for (int i = 0; i < compaction_values.Length; i++) {
            float compaction_value = compaction_values[i];
            if (compaction_value >= 100.0f) excessive_compaction_count++;
            else if (compaction_value <= 0) uncompaction_count++;
            compaction_sum += compaction_value;
            if (i > 0 && i % results.compactionTexture.height == 0) {
                results.averageCompactionPerLane[lane_index++] = (compaction_sum - prev_sum) /
results.compactionTexture.height;
                prev_sum = compaction_sum;
            }
            compaction_point_count++;
        }
                                results.averageCompaction =      (compaction_sum
                            / (float)compaction_point_count);
        results.averageExcess =          (excessive_compaction_count      /
(float)compaction_values.Length) * 100.0f;
        results.averageUncompacted =     (uncompaction_count              /
(float)compaction_values.Length) * 100.0f;

        AnalyticsManager.AddData(game_state.currentLevel.naam + "_" + "average_compaction",
results.averageCompaction);
        AnalyticsManager.AddData(game_state.currentLevel.naam + "_" + "excess_percentage",
results.averageExcess);
        AnalyticsManager.AddData(game_state.currentLevel.naam + "_" +
"uncompacted_percentage", results.averageUncompacted);
                            }

                            public void EndExecutionPhase() {
                                CalculateExecutionResults();

        // Load reflection scene.
        SceneManager.LoadScene("feedback");
                            }
}
----------------------------------
ExecutionTutorial.cs
----------------------------------

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class ExecutionTutorial : MonoBehaviour {

                            [SerializeField]
                            private int tutorial_state = 0;

                            public GameObject textBubble;
                            public GameObject tutorialTextPanel;
                            public TextBubblePositioner textBubbleManager;

                            [HeaderAttribute("Targets")]
                            public Transform paver;
                            public Transform compactor;
                            public RectTransform visualisationDropdown;
                            public RectTransform pauseButton;
                            public RectTransform speedSliderHandle;
                            public RectTransform timeElement;
                            public RectTransform doneButton;

                            // Private variables
```

```
                            Vector3 last_camera_position;
                            float last_orto_size;
                            UnitSelector unitSelector;
                            float last_time_scale = 0;
                            new Camera camera;

                            private string[] tutorial_texts = {
/*0*/                       "We zijn nu aangekomen bij de uitvoeringsfase. In deze fase wordt
het werk uitgevoerd en heb je controle over de spreidmachine en de wals.",
/*1*/                       "Eerst zullen we bekijken hoe we de virtuele camera kunnen
verplaatsen.",
/*2*/                       "Om de camera te verplaatsen moet je de linkermuisknop indrukken
en de muis naar rechts verschuiven. Probeer dit nu.",
/*3*/                       "En verplaats de camera vervolgens naar rechts door je muis naar
links te verplaatsen.",
/*4*/                       "Buiten de camera verplaatsen, kun je de camera ook in- en
uitzoomen door het scrollwiel te draaien.",
/*5*/                       "Draai het scrollwiel van je af om in te zoomen.",
/*6*/                       "Zoom nu weer uit door het skrollwiel naar je toe te draaien.",
/*7*/                       "Nu je weet hoe je de camera moet manipuleren gaan, kunnen we
beginnen met het besturen van de machines.",
/*8*/                       "Voor je zou je twee machines moeten zien, een
asfaltspreidmachine en een wals. Je kunt machines selecteren door op de linkermuisknop te
drukken.",
/*9*/                       "Selecteer de spreidmachine met de linkermuisknop.",
/*10*/                      "Zet nu de snelheid van de spreidmachine op 5 meter per minuut,
door de schuif langzaam naar rechts te schuiven.",
/*11*/                      "De camera volgt automatisch het laatst geselecteerde voertuig,
dit ontdoe je door handmatig de camera te verplaatsen.",
/*12*/                      "De wals bestuurt wat lastiger dan de spreidmachine. Als je hem
selecteerd dan zie je twee blauwe markers. De wals zal tussen deze twee markers blijven
bewegen zolang de snelheid hoger is dan 0 kilometer per uur.",
/*13*/                      "Selecteer nu de wals met de linkermuisknop.",
/*14*/                      "Verplaats beide blauwe markers zodat ze op de weg staan en zet
dan de snelheid naar 5 kilometer per uur.",
/*15*/                      "Om te kunnen zien hoe verdicht het asfalt is, moeten we de
visualisatie veranderen. Dat kan met de visualisatie drop-down.",
/*16*/                      "Druk op de visualisatie drop-down menu en klik op 'Asfalt
Verdichting'",
/*17*/                      "De verdichting is goed als hij groen is, en deste groener, deste
beter. Rood betekend dat je te vroeg, of te veel hebt verdicht.",
/*18*/                      "Je kunt nu duidelijk zien dat het helemaal mis gaat. Het asfalt
is veel te heet om nu al te gaan walsen. Om de temperatuur te zien, moet je de visualisatie
drop-down naar 'Asfalt Hitte' zetten.",
/*19*/                      "Druk op de visualisatie drop-down menu en klik op 'Asfalt
Hitte'",
/*20*/                      "Deze visualisatie laat de temperatuur van het asfalt zien. Het
verloopt van rood, naar groen naar blauw. Als het asfalt groen is, is het tijd om te walsen.",
/*21*/                      "Het duurt nog wel even voordat het asfalt groen is. Gelukkig
hoeven we niet te wachten en kunnen we de tijd versnellen.",
/*22*/                      "Als je de schuifknop onder 'Tijd versnelling' schuift kun je de
tijd versnellen en verlangzamen. Probeer dit nu.",
/*23*/                      "We kunnen ook de tijd stil zetten, dit doe je door op de pauze
knop te klikken. Doe dit nu.",
/*24*/                      "Dat was de tutorial, goed gedaan. Nu ben je klaar om te beginnen
als asfalt uitvoerder en ga je straks zelf je eerste weg aanleggen!",
                            };

                            // Use this for initialization
                            void Start () {
                                camera = Camera.main;

                                paver = GameObject.FindGameObjectWithTag("Paver").transform;
                                compactor =
GameObject.FindGameObjectWithTag("Compactor").transform;
                                unitSelector = FindObjectOfType<UnitSelector>();


                            visualisationDropdown.GetComponent<Dropdown>().onValueChanged.Add
Listener((option_index)=>{
                                        if (tutorial_state == 17) {
                                                if (option_index == 2) {
                                                        NextTutorialStep();
                                                } else {
```

```
                                                textBubble.SetActive(true);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        "Je hebt de verkeerde optie
gekozen, kies 'Asfalt Verdichting'.";
                                        }
                                } else if (tutorial_state == 20) {
                                        if (option_index == 1) {
                                                NextTutorialStep();
                                        } else {
                                                textBubble.SetActive(true);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        "Je hebt de verkeerde optie
gekozen, kies 'Asfalt Hitte'.";
                                        }
                                }
                            });

                            pauseButton.GetComponent<Button>().onClick.AddListener(()=>{
                                    StartCoroutine(WaitForNextTutorial(1));
                            });

                            NextTutorialStep();
                        }

                        // Update is called once per frame
                        void Update () {
                            switch (tutorial_state) {
                                    // Check if the camera has traveled to the left
                                    case 3: {
                                            if (camera.transform.position.x <
last_camera_position.x &&

                                Vector3.Distance(camera.transform.position, last_camera_position)
>= 75) {
                                                    NextTutorialStep();
                                            } else if (camera.transform.position.x >
last_camera_position.x &&

                                Vector3.Distance(camera.transform.position, last_camera_position)
>= 40) {
                                                    textBubble.SetActive(true);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        "Je gaat de verkeerde kant
op. Schuif de muis naar rechts om de camera naar links te verplaatsen.";
                                            }
                                    } break;

                                    // Check if the camera has traveled to the left
                                    case 4: {
                                            if (camera.transform.position.x >
last_camera_position.x &&

                                Vector3.Distance(camera.transform.position, last_camera_position)
>= 75) {
                                                    NextTutorialStep();
                                            } else if (camera.transform.position.x <
last_camera_position.x &&

                                Vector3.Distance(camera.transform.position, last_camera_position)
>= 40) {
                                                    textBubble.SetActive(true);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        "Je gaat de verkeerde kant
op. Schuif de muis naar links om de camera naar rechts te verplaatsen.";
                                            }
                                    } break;

                                    // Check if the user has scrolled in.
                                    case 6: {
```

```
                                                if (camera.orthographicSize < last_orto_size
&&
                                                        Mathf.Abs(camera.orthographicSize -
last_orto_size) >= 10) {
                                                        NextTutorialStep();
                                                }
                                        } break;

                                        // Check if the user has scrolled in.
                                        case 7: {
                                                if (camera.orthographicSize > last_orto_size
&&
                                                        Mathf.Abs(camera.orthographicSize -
last_orto_size) >= 10) {
                                                        NextTutorialStep();
                                                }
                                        } break;

                                        case 10: {
                                                if (unitSelector.targetUnit != null &&
                                                        unitSelector.targetUnit ==
paver.gameObject) {

                                StartCoroutine(WaitForNextTutorial(1));
                                                }
                                        } break;

                                        case 11: {
                        float speed = paver.GetComponent<PaverController>().speed;
                                                if (speed >= 0.05f) {

                                StartCoroutine(WaitForNextTutorial(1));
                                                }
                                        } break;

                                        case 14: {
                                                if (unitSelector.targetUnit != null &&
                                                        unitSelector.targetUnit ==
compactor.gameObject) {

                                StartCoroutine(WaitForNextTutorial(1));
                                                }
                                        } break;

                                        case 15: {
                                                if
(compactor.GetComponent<RoadCompactor>().speed >= 5.0f) {

                                StartCoroutine(WaitForNextTutorial(1));
                                                }
                                        } break;

                                        case 21: {
                                                StartCoroutine(WaitForNextTutorial(6));
                                        } break;

                                        case 23: {
                                                if (last_time_scale * 2 <
ExecutionManager.simulationTimeScale) {

                                StartCoroutine(WaitForNextTutorial(2));
                                                }
                                        } break;

                                        default: {} break;
                                    }
                                }

                                public void NextTutorialStep() {
                                    // Reset all
                                    tutorialTextPanel.SetActive(false);

                                tutorialTextPanel.transform.SetSiblingIndex(tutorialTextPanel.tra
nsform.parent.childCount - 1);
```

```
                textBubble.SetActive(false);

                switch(tutorial_state) {
                        // Tutorial text panel
                        case 0:
                        case 1:
                        case 4:
                        case 7:
                        case 11:
                        case 15:
                        case 17:
                        case 18:
                        case 21:
                        case 24: {
                                tutorialTextPanel.SetActive(true);

                        tutorialTextPanel.GetComponentInChildren<Text>().text =
                                        tutorial_texts[tutorial_state] + "
Klik op volgende om verder te gaan.";
                        } break;

                        // Tutorial bubble
                        case 2:
                        case 3:
                        case 5:
                        case 6:
                        case 13: {
                                textBubble.SetActive(true);

                        textBubble.GetComponentInChildren<Text>().text =
                                        tutorial_texts[tutorial_state];
                                last_camera_position =
camera.transform.position;
                                last_orto_size = camera.orthographicSize;
                        } break;

                        case 9: {
                                textBubble.SetActive(true);
                                textBubbleManager.SetUITargetPosition(paver,
0, 100);

                        textBubble.GetComponentInChildren<Text>().text =
                                        tutorial_texts[tutorial_state];
                        } break;

                        case 10: {
                                textBubble.SetActive(true);
                                textBubbleManager.SetUITargetPosition(paver,
0, 100);

                        textBubble.GetComponentInChildren<Text>().text =
                                        tutorial_texts[tutorial_state];
                        } break;

                        case 8: {

                        camera.gameObject.GetComponent<FollowTarget>().SetTarget(paver);
                                tutorialTextPanel.SetActive(true);

                        tutorialTextPanel.GetComponentInChildren<Text>().text =
                                        tutorial_texts[tutorial_state] + "
Klik op volgende om verder te gaan.";
                        } break;

                        case 12: {
                                tutorialTextPanel.SetActive(true);

                        tutorialTextPanel.GetComponentInChildren<Text>().text =
                                        tutorial_texts[tutorial_state] + "
Klik op volgende om verder te gaan.";

                        camera.gameObject.GetComponent<FollowTarget>().SetTarget(compacto
r);
                        } break;
```

```
                                        case 14:
                                        case 20: {
                                                textBubble.SetActive(true);

                                textBubbleManager.SetUITargetPosition(compactor, 0, 100);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        tutorial_texts[tutorial_state];
                                        } break;

                                        case 16:
                                        case 19: {
                                                textBubble.SetActive(true);

                                textBubbleManager.SetUITargetPosition(visualisationDropdown);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        tutorial_texts[tutorial_state];
                                        } break;

                                        case 22: {
                                                textBubble.SetActive(true);

                                textBubbleManager.SetUITargetPosition(timeElement);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        tutorial_texts[tutorial_state];
                                                last_time_scale =
ExecutionManager.simulationTimeScale;
                                        } break;

                                        case 23: {
                                                textBubble.SetActive(true);

                                textBubbleManager.SetUITargetPosition(pauseButton);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        tutorial_texts[tutorial_state];
                                        } break;

                                        case 25: {
                                                StartEvaluationPhase();
                                        } break;
                                }
                                tutorial_state++;
                        }

                        bool has_set_timer = false;
                        IEnumerator WaitForNextTutorial(float wait_time) {
                            if (!has_set_timer) {
                                    has_set_timer = true;
                                    yield return new WaitForSeconds(wait_time);
                                    NextTutorialStep();
                                    has_set_timer = false;
                            }
                        }

                        public void StartEvaluationPhase() {
                            // Do evaluation step

                        FindObjectOfType<ExecutionManager>().CalculateExecutionResults();

                            // Go to next scene
                            SceneManager.LoadSceneAsync("feedback_tutorial",
LoadSceneMode.Single);
                        }
}
```

FeedbackTutorialManager.cs

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class FeedbackTutorialManager : MonoBehaviour {

                            [SerializeField]
                            private int tutorial_state = 0;

                            public ScoreCanvasManager scoreManager;
                            public GameObject textBubble;
                            public GameObject tutorialTextPanel;
                            public TextBubblePositioner textBubbleManager;

                            [HeaderAttribute("Targets")]
                            public RectTransform scrollView;
                            public Button doneButton;

                            // Private variables
                            GameState game_state;
                            RectTransform star_container;
                            RectTransform bubble_target;
                            RectTransform compaction_element;

                            private string[] tutorial_texts = {
/*0*/                       "Aan het einde van zowel de planningsfase en de uitvoeringsfase
krijg je je resultaten te zien.",
/*1*/                       "Hierbij wordt je beoordeeld op je prestaties door middel van
tekst en sterren. Deze sterren heb je nodig om naar volgende levels te mogen.",
/*2*/                       "Ook krijg je een korte samenvatting van het aantal gebrukte
machines, het weer, het aantal botsingen, de kwaliteit van het wegdek en hoe lang je over het
level hebt gedaan. Deze tellen allemaal mee in de beoordeling.",
/*3*/                       "In dit venster kun je je resultaten zien. Er zijn nog meer
resultaten als je naar beneden scrolt. Scroll helemaal naar beneden.",
/*4*/                       "Hier kun je het totale aantal behaalde sterren te zien.",
/*5*/                       "Gefeliciteerd, je hebt je eerste ster gehaald! Kijk nog even
naar de rest van je resultaten.",
/*6*/                       "Je kunt bijvoorbeeld zien dat je gemiddelde dichtheid niet zo
hoog ligt en dat er een deel te veel verdicth is.",
/*7*/                       "Hier kun je zien hoe de verdichting is geweest voor het gehele
wegdek.",
/*8*/                       "Als je klaar bent met de resultaten kun je op klaar klikken.",
                            };

                            // Use this for initialization
                            void Start () {
                                if (scoreManager.starsContainers.Count > 0) {
                                        star_container = scoreManager.starsContainers[0];

                            star_container.GetComponent<UILevelElement>().SetStars(1, 1);
                                }

                                if (scoreManager.textureContainers.Count > 0) {
                                        compaction_element =
scoreManager.textureContainers[0];
                                }

                                doneButton.onClick.AddListener(()=>{
                                        // Done with the tutorial
                                        int gained_stars = PlayerPrefs.GetInt("stars", 0);
                                        if (gained_stars == 0) {
                                                PlayerPrefs.SetInt("stars", 1);
                                        }
                                        SceneManager.LoadScene("main_menu");
                                });

                                NextTutorialStep();
                            }

                            // Update is called once per frame
                            void Update () {
                                switch (tutorial_state) {
                                        case 4: {
```

```
                                        if (Mathf.Abs(scrollView.offsetMin.y) < 0.1f)
{
                                                NextTutorialStep();
                                        }
                                } break;

                                case 5: {
                                        StartCoroutine(WaitForNextTutorial(2));
                                } break;

                                case 8: {
                                        StartCoroutine(WaitForNextTutorial(3));
                                } break;
                        }
                    }

                    public void NextTutorialStep() {
                        tutorialTextPanel.SetActive(false);
                        textBubble.SetActive(false);

                        switch(tutorial_state) {
                                case 0:
                                case 1:
                                case 2:
                                case 5:
                                case 6: {
                                        tutorialTextPanel.SetActive(true);

                    tutorialTextPanel.GetComponentInChildren<Text>().text =
                                                tutorial_texts[tutorial_state] + "
Klik op volgende om verder te gaan.";
                                } break;

                                case 3: {
                                        textBubble.SetActive(true);

                    textBubble.GetComponentInChildren<Text>().text =
                                                tutorial_texts[tutorial_state];

                    textBubble.GetComponent<RectTransform>().anchoredPosition =
                                                new Vector2(0, Screen.height * 0.8f);
                                } break;

                                case 4: {
                                        textBubble.SetActive(true);

                    textBubble.GetComponentInChildren<Text>().text =
                                                tutorial_texts[tutorial_state];

                    textBubbleManager.SetUITargetPosition(star_container);
                                } break;

                                case 7: {
                                        textBubble.SetActive(true);

                    textBubble.GetComponentInChildren<Text>().text =
                                                tutorial_texts[tutorial_state];

                    textBubbleManager.SetUITargetPosition(compaction_element);
                                } break;

                                case 8: {
                                        textBubble.SetActive(true);

                    textBubble.GetComponentInChildren<Text>().text =
                                                tutorial_texts[tutorial_state];

                    textBubbleManager.SetUITargetPosition(doneButton.transform, 0,
                                                50);
                                } break;
                        }
                        tutorial_state++;
                    }
```

```
                            bool has_set_timer = false;
                            IEnumerator WaitForNextTutorial(float wait_time) {
                                if (!has_set_timer) {
                                        has_set_timer = true;
                                        yield return new WaitForSeconds(wait_time);
                                        NextTutorialStep();
                                        has_set_timer = false;
                                }
                            }
}
-----------------------------------
FollowTarget.cs
-----------------------------------

using UnityEngine;
using UnityEngine.EventSystems;
using System.Collections;

public class FollowTarget : MonoBehaviour {
    public static int LEFT_MOUSE_BTN = 0;

    public float   followSpeed = 2;
    public float   panSpeed    = 5;
    public float   zoomSpeed   = 5;
    public float   zoomDamping = 0.8f;
    public float   minDistance = 5;
    public float   maxDistance = 200;
    public Vector3 offset      = new Vector3(1, 90, 0);

    bool       has_mouse_gone_down;

    Transform  target;
    Vector3    pan_target;
    bool       pan_mode;

    Vector3    prev_mouse_pos;
    Vector3    ground_point;                   // Last point where we panned to

    float      zoom_velocity;

    new Camera camera;

                            // Use this for initialization
                            void Start () {
        camera = GetComponent<Camera>();
                            }

                            // Update is called once per frame
                            void Update () {
        float scroll_delta = -Input.GetAxis("Mouse ScrollWheel");

        if (!EventSystem.current.IsPointerOverGameObject() &&
            Input.GetMouseButtonDown(LEFT_MOUSE_BTN)) {
            has_mouse_gone_down = true;

            // Enter camera pan mode
            RaycastHit hitInfo;
            if (Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition),
                                out hitInfo, 1000f)) {
                if (hitInfo.collider.gameObject.layer == Layers.FLOOR_INT) {
                    print("Hit ground point");
                    ground_point = hitInfo.point;
                    pan_mode = true;
                    target = null;
                    prev_mouse_pos = Input.mousePosition;
                }
            }
        } else if (Input.GetMouseButtonUp(LEFT_MOUSE_BTN)) {
            has_mouse_gone_down = false;
        }

        if (target) {
            // Goto target
            transform.position = Vector3.Lerp(transform.position, target.position + offset,
```

```
                                            followSpeed * Time.deltaTime);
        } else if (pan_mode) {
            if (has_mouse_gone_down && Input.GetMouseButton(LEFT_MOUSE_BTN)) {
                // Panning
                Vector3 delta = Input.mousePosition - prev_mouse_pos;
                delta.y = 0;
                transform.Translate(-delta.x * panSpeed, -delta.y * panSpeed, 0);
                prev_mouse_pos = Input.mousePosition;
            }
        }

        if (target) {
            // Zooming
            UpdateZoomLevel(scroll_delta, target.transform.position);
        } else if (pan_mode) {
            UpdateZoomLevel(scroll_delta, ground_point);
        }
                            }

    public void UpdateZoomLevel(float scroll_delta, Vector3 target) {
        float zoom_acceleration = scroll_delta * zoomSpeed;
        // print("Zoom acceleration: " + zoom_acceleration);

        if (Mathf.Abs(zoom_acceleration) > 0.01f) {
            if (Mathf.Sign(zoom_acceleration) == Mathf.Sign(zoom_velocity)) {
                zoom_velocity += zoom_acceleration;
            } else {
                // Reverse zoom zoom_velocity
                zoom_velocity = 0;
                zoom_velocity += zoom_acceleration;
            }
        } else {
            zoom_velocity *= zoomDamping;
        }

        if (Mathf.Abs(zoom_velocity) < 0.01f) zoom_velocity = 0;

        // print("Zoom zoom_velocity: " + zoom_velocity);

        camera.orthographicSize += zoom_velocity;
        camera.orthographicSize = Mathf.Clamp(camera.orthographicSize, minDistance,
maxDistance);
    }

    public void SetTarget(Transform target) {
        this.target = target;
        pan_mode = false;
    }
}
-----------------------------------
GameSettings.cs
-----------------------------------

using UnityEngine;
using System.Collections;

public class GameSettings {

                            public bool fullscreen;                     // true / false
                            public int texture_quality;         // 0:low 2:medium 3:high
                            public int vsync;                           // 0 (no) 1 (every)
2 (every other)
                            public int antialiasing;            // none 2x 4x 8x
                            public int shadow_quality;          // 0:low 2:medium 3:high
4:very high
                            // TODO(Peter): Serialize the width, heihgt, and frequency
                            public int resolution_index;// Index of the setting
                            public float music_volume;          // 0.0f - 1.0f
                            public float sfx_volume;            // 0.0f - 1.0f
}
-----------------------------------
GameState.cs
-----------------------------------
```

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;

public class GameState : MonoBehaviour {

    public GameObject gameStateObject;

    [System.Serializable]
    public class ExecutionResults {
        [HeaderAttribute("Planning")]
        public float totalTime;

        [HeaderAttribute("Collisions")]
        public int paverCollisions;
        public int compactorCollisions;

        [HeaderAttribute("Compaction Results")]
        public Texture2D compactionTexture;
        public float averageCompaction;
        public float[] averageCompactionPerLane;
        public float averageExcess;
        public float averageUncompacted;

        public ExecutionResults(int compactionLanes) {
            averageCompactionPerLane = new float[compactionLanes];
        }
    }

    public Level currentLevel;

    public ExecutionResults results;

    // Singleton pattern
    private static GameState _instance;

    public static GameState Instance {
        get {
            if (_instance == null) {
                GameObject go = new GameObject("Game State");
                _instance = go.AddComponent<GameState>();
                GameTextManager gtm = go.AddComponent<GameTextManager>();
                gtm.filename = "tekst_database";
            }
            return _instance;
        }
    }

    void Awake () {
        if (_instance && _instance != this) {
            Destroy(this.gameObject);
        } else {
            _instance = this;
            DontDestroyOnLoad(this.gameObject);
        }
    }
}
-----------------------------------
Level.cs
-----------------------------------

[System.Serializable]
public enum RoadType {
                            SMA,
                            ZOAB,
                            AC16_SURF
}

[System.Serializable]
public class WeatherCondition {
                            public float temperatuur;
                            public bool bewolkt;
                            public bool regen;
                            public bool zonnig;
```

```
                            public bool sneeuw;
}

[System.Serializable]
public class Level {
                            // Level Parameters
                            public string naam;
                            public int planning_sterren;
                            public int uitvoering_sterren;
                            public int benodigde_sterren;

                            // Planning Parameters
                            public string intro_text;
                            public bool kan_machines_kiezen;
                            public int spreidmachines;
                            public int statische_walsen;
                            public int dynamische_walsen;
                            public int vrachtwagens;

                            // Road Parameters
                            public bool kan_afmetingen_kiezen;
                            public float weg_breedte;
                            public float weg_dikte;
                            public float weg_lengte;
                            public RoadType weg_type;
    public float yolo;

                            public WeatherCondition weather;

                            [System.Serializable]
                            public class Goals {
                                public float minimum_verdichting      = -1;
                                public float maximum_verdichting      = -1;
                                public float gemiddelde_verdichting   = -1;
                                public float maximale_duur_uitvoering = -1;
                            }
                            public Goals goals;

                            // Constructor
                            public Level() {
                                weather = new WeatherCondition();
                                goals   = new Goals();
                            }
}
-----------------------------------
LevelLoader.cs
-----------------------------------

using System;
using System.Text.RegularExpressions;
using System.Reflection;

using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LevelLoader : MonoBehaviour {

                            public Level[] loadedLevels;

                            public GameObject UiLevelPrefab;
                            private GameObject[] ui_levels;

                            public int gainedStars;

                            public Text starCounterText;

                            void Awake() {
                                 // Load all files with extention *.level.txt
                                TextAsset[] level_files =
Resources.LoadAll<TextAsset>("Levels");

                                int level_index = 0;
                                loadedLevels = new Level[level_files.Length];
```

```
                                print("Trying to load " + level_files.Length + " levels...");
                                foreach (TextAsset level_file in level_files) {
                                        loadedLevels[level_index++] =
LoadLevelString(level_file.text);
                                }

                                // Manage the stars counter
                                gainedStars = PlayerPrefs.GetInt("stars", 0);
                                starCounterText.text = "" + gainedStars;

                                // Create levels in UI
                                ui_levels = new GameObject[level_index];
                                Transform level_element_container =
transform.Find("Viewport/Content");
                                for (int i = 0; i < loadedLevels.Length; i++) {
                                        Level level = loadedLevels[i];

                                        GameObject ui_level = ui_levels[i] =
Instantiate(UiLevelPrefab);
                                        ui_level.transform.SetParent(level_element_container,
false);
                                        UILevelElement level_element_manager =
ui_level.GetComponent<UILevelElement>();
                                        level_element_manager.Name = level.naam;
                                        level_element_manager.SetStars(level.planning_sterren
+ level.uitvoering_sterren, 0);
                                        level_element_manager.requiredStars =
level.benodigde_sterren;
            // Lock all levels after
            //if (gainedStars <= 0) {
            //    level_element_manager.SetLock(gainedStars);
            //}

                                        Button level_select_btn =
ui_level.GetComponent<Button>();
                                        level_select_btn.interactable =
!level_element_manager.isLocked;
                                        level_select_btn.onClick.AddListener(()=>{
                                                LoadLevel(level);
                AnalyticsManager.IncrementKey(level.naam + "_playcount");
                                        });
                                }
                        }

                        // Use this for initialization
                        void Start () {
                        }

                        // Update is called once per frame
                        void Update () {
                        }

                        public void LoadLevel(Level level) {
                            GameState.Instance.currentLevel = level;
                            SceneManager.LoadScene("planning");
                        }

                        public void LoadTutorial() {
                            Level level = new Level();
                            level.kan_machines_kiezen = true;
                            GameState.Instance.currentLevel = level;
                            SceneManager.LoadScene("planning_tutorial");
                        }

                        public static Level LoadLevelFile(string levelName) {
        TextAsset level_file = Resources.Load<TextAsset>("Levels/" + levelName);
        Level result = LoadLevelString(level_file.text);
                            print("Succesfully loaded level: " + levelName);
        return result;
    }

                        public static Level LoadLevelString(string level_text) {
                            Level result = new Level();
```

```
                                string file_contents = level_text;
                                string[] lines = file_contents.Split(new string[] { "\r\n",
"\n" },

                                 StringSplitOptions.None);
                                for (int line_index = 0; line_index < lines.Length;
line_index++) {
                                        string line = lines[line_index].Trim();

                                        if (string.IsNullOrEmpty(line) || (line[0] == '/' &&
line[1] == '/')) {
                                                continue;
                                        }

                                        // Check for sub classes
                                        if (line[0] == '[') {
                                                // We know that we need to look for a sub
class
                                                line = line.Replace("[", "");
                                                line = line.Replace("]", "");

                                                FieldInfo field =
typeof(Level).GetField(line, BindingFlags.Instance |

                                                                                        BindingFlags.Public
|


                                BindingFlags.IgnoreCase);
                                                if (field != null) {
                                                        object sub_class_value =
field.GetValue(result);
                                                        line = lines[++line_index];
                                                        while (!string.IsNullOrEmpty(line) &&
line_index < lines.Length) {
                                                                ParseLine(line,
sub_class_value);
                                                                line_index++;
                                                                if (line_index >=
lines.Length) break;
                                                                line = lines[line_index];

                                                        }
                                                } else {
                                                        Debug.LogError("Error, could not
parse subclass: " + line + ".");
                                                }
                                        } else {
                                                ParseLine(line, result);
                                        }
                                }

                                return result;
                        }

                        private static void ParseLine(string line, object result) {
                            int colon_index = line.IndexOf(':');
                            string line_name  = line.Substring(0, colon_index);
                            string line_value = line.Substring(colon_index + 1).Trim();

                            MemberInfo[] members = result.GetType().GetMember(line_name);
                if (members.Length > 0) {
                                        MemberInfo member_info = members[0];
//                                      print(string.Format("Parsing: {0} -> {1}",
//                                              line_name, line_value));
                                        if (line_value.Contains("\"")) {
                                                line_value = line_value.Replace("\"", "");
                                                line_value = line_value.Replace("\\n", "\n");
                                                SetMemberValue(member_info, result,
line_value);
                                        } else if (line_value.Contains(".")) {
                                                SetMemberValue(member_info, result,
float.Parse(line_value));
```

```
                                        } else if (line_value.Contains("true") ||
line_value.Contains("false")) {
                                                SetMemberValue(member_info, result,
bool.Parse(line_value));
                                        } else if (line_name.Contains("weg_type")) {
                                                SetMemberValue(member_info, result,
Enum.Parse(typeof(RoadType), line_value));
                                        } else if (Regex.IsMatch(line_value, @"^\d+$")) {
                                                SetMemberValue(member_info, result,
int.Parse(line_value));
                                        } else {
                                                // We don't know how to parse this variable
yet.
                                                print(String.Format("Error: Unable to parse
{0} with value {1}.",
                                                line_name, line_value));
                                        }
        } else {
            Debug.LogError("Error, could not parse member " + line_name + ".");
        }
                            }

                            private static void SetMemberValue(MemberInfo member, object
target, object value) {
                                switch (member.MemberType) {
                                        case MemberTypes.Field: {
                                                ((FieldInfo)member).SetValue(target, value);
                                        } break;
                                        case MemberTypes.Property: {
                                                ((PropertyInfo)member).SetValue(target,
value, null);
                                        } break;
                                        default: { return; }
                                }
                            }
}
-----------------------------------
LoginManager.cs
-----------------------------------

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class LoginManager : MonoBehaviour {

    public MenuManager menuManager;
    private string username;
    private InputField input_field;

    public void AdvanceMenu() {
        if (string.IsNullOrEmpty(username)) {
            if (!input_field) {
                input_field = GetComponentInChildren<InputField>();
            }
            Text placeholder_text = (input_field.placeholder as Text);
            placeholder_text.supportRichText = true;
            placeholder_text.text = "<color=red>" + "Voer een gebruikersnaam in!" +
"</color>";
        } else {
            menuManager.PushMenu(menuManager.titleMenu);
        }
    }

    public void SetUser(string name) {
        AnalyticsManager.username = username = name;
    }
}
-----------------------------------
MainMenuController.cs
-----------------------------------

using UnityEngine;
```

```
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using System.Collections;

public class MainMenuController : MonoBehaviour {

    public Button storyModeButton;
    public Button sandboxModeButton;
    public Button scenarioModeButton;

    public string executionScene;
    public string planningScene;

                            // Use this for initialization
                            void Start () {

                            }

                            // Update is called once per frame
                            void Update () {

                            }

    public void LoadExecution() {
        SceneManager.LoadScene(executionScene);
    }

    public void LoadPlanning() {
        SceneManager.LoadScene(planningScene);
    }
}
-----------------------------------
MenuManager.cs
-----------------------------------

using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class MenuManager : MonoBehaviour {

    public GameObject loginScreen;
                            public GameObject titleMenu;
                            public GameObject mainMenu;
                            public GameObject levelMenu;
                            //public GameObject scenario_menu;
                            public GameObject settingsMenu;

                            public string url = "http://www.verzijl.com/peter/";

    public new Camera camera;
    public float moveSpeed = 20;
    public float rotationSpeed = 5;
    public Transform[] camera_positions = new Transform[3];
                            public Stack<GameObject> menu_stack = new Stack<GameObject>();

    private Transform target;

                            // Use this for initialization
                            void Start () {
        if (!camera) camera = Camera.main;

        loginScreen.SetActive(false);
                                titleMenu.SetActive (false);
                                mainMenu.SetActive (false);
                                levelMenu.SetActive (false);
                                //scenario_menu.SetActive (false);
                                settingsMenu.SetActive (false);

        if (string.IsNullOrEmpty(AnalyticsManager.username)) {
                                    PushMenu(loginScreen);
        } else {
            PushMenu(titleMenu);
        }
```

```
                        }

                        // Update is called once per frame
                        void Update () {
        GameObject cur_menu = menu_stack.Peek();
        if (cur_menu == titleMenu) {
            target = camera_positions[0];
        } else if (cur_menu == mainMenu) {
            target = camera_positions[1];
        } else if (cur_menu == levelMenu) {
            target = camera_positions[2];
        } else if (cur_menu == settingsMenu) {
            target = camera_positions[2];
        } else {    // Default
            target = camera_positions[0];
        }

        camera.transform.position = Vector3.Slerp(camera.transform.position,
            target.position, moveSpeed * Time.deltaTime);
        camera.transform.rotation = Quaternion.Slerp(camera.transform.rotation,
            target.rotation, rotationSpeed * Time.deltaTime);
                        }

                        public void PopMenu() {
                            if (menu_stack.Count > 1) {
                                    menu_stack.Pop ().SetActive (false);
                                    menu_stack.Peek ().SetActive (true);
                            }
                            // If there are less than 2 in the stack, do nothing.
                        }

                        public void PushMenu(GameObject menu) {
                            if (menu_stack.Count > 0) {
                                    menu_stack.Peek ().SetActive (false);
                            }
                            menu.SetActive (true);
                            menu_stack.Push (menu);
                        }

                        public void Quit() {
#if UNITY_EDITOR
                            UnityEditor.EditorApplication.isPlaying = false;
#elif UNITY_WEBPLAYER
                            Application.OpenURL(url);
#else
                            Application.Quit();
#endif
                        }
}
-----------------------------------
PauseButton.cs
-----------------------------------

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

//[RequireComponent(typeof(Button))]
public class PauseButton : MonoBehaviour {

                        public Sprite onSprite;
                        public Sprite offSprite;
                        public GameObject pauseScreen;

                        Button button;
                        Image toggableImage;

                        float prev_time_scaleTime;

                        /// <summary>
                        /// Awake is called when the script instance is being loaded.
                        /// </summary>
                        void Awake() {
```

```
                                button = GetComponent<Button>();
                                button.onClick.AddListener(TogglePause);
                                toggableImage =
transform.Find("Image").GetComponent<Image>();

                                pauseScreen.SetActive(ExecutionManager.isPaused);
                                toggableImage.sprite = (ExecutionManager.isPaused)? onSprite
: offSprite;
                            }

                            // Use this for initialization
                            void Start () {

                            }

                            // Update is called once per frame
                            void Update () {

                            }

                            public void TogglePause() {
                                ExecutionManager.isPaused = !ExecutionManager.isPaused;
                                pauseScreen.SetActive(ExecutionManager.isPaused);
                                toggableImage.sprite = (ExecutionManager.isPaused)? onSprite
: offSprite;
                            }
}
----------------------------------
PaverController.cs
----------------------------------

using UnityEngine;
using UnityEngine.UI;
using System;
using System.Collections;
using System.Collections.Generic;

public class PaverController : MonoBehaviour {

    public GameObject roadPrefab;

    public float minSpeed = 0;
    public float maxSpeed = 7;
    public float speed = 0;

    public Transform leftPaver;
    public Transform rightPaver;

    public GameObject current_road;
    public TextureManager road_manager;
    public bool isPaving = false;
    public float totalDistance = 50;
    private float distance_traveled = 0;

    private Vector3 prev_position;
    [Tooltip("In degrees celsius")]
    public float hmaTemperature = 300;

    // Use this for initialization
    void Start () {
        prev_position = transform.position;

        road_manager.StartRoad(leftPaver.position, rightPaver.position, totalDistance);
                            }

                            // Update is called once per frame
                            void Update () {
        if (ExecutionManager.isPaused) return;

        transform.position += transform.forward * speed * Time.deltaTime *
ExecutionManager.simulationTimeScale;

        if (isPaving) {
            // Update the road
```

```
            float delta_distance = Vector3.Distance(prev_position, transform.position);

            // Stop paving once done.
            if (distance_traveled + delta_distance >= totalDistance) {
                isPaving = false;
                SetSpeed(0);
                UpdateRoadLength(ref delta_distance);
            }

            if (delta_distance > 1f) {
                UpdateRoadLength(ref delta_distance);
            }
        }
                            }

    public float SetSpeed(float speed) {
        this.speed = speed;
        return this.speed;
    }

    private void UpdateRoadLength(ref float delta_distance) {
        prev_position = transform.position;
        distance_traveled += delta_distance;
        road_manager.UpdateRoadLength(leftPaver.position,
            rightPaver.position, delta_distance, hmaTemperature);
    }
}
----------------------------------
PavingPropertyManager.cs
----------------------------------

using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class PavingPropertyManager : MonoBehaviour {

    public Text speedLabelText;
    public Slider speedSlider;

    public Toggle pavingToggle;
    public PaverController paver;

                            // Use this for initialization
                            void Start () {
                                paver.isPaving = pavingToggle.isOn;
        pavingToggle.onValueChanged.AddListener((value) => {
            paver.isPaving = value;
        });

        float speed_mps = paver.speed;
        float speed_mpmin = speed_mps * 60f;
        speedLabelText.text = "Snelheid " + speed_mpmin.ToString("n1") + " m/min";
        speedSlider.minValue = paver.minSpeed;
        speedSlider.maxValue = paver.maxSpeed;
        speedSlider.value = paver.speed;
        speedSlider.onValueChanged.AddListener((value) => {
            speed_mps = paver.SetSpeed(value);
            speed_mpmin = speed_mps * 60f;
            speedLabelText.text = "Snelheid " + speed_mpmin.ToString("n1") + " m/min";
        });
                            }

                            // Update is called once per frame
                            void Update () {

                            }
}
----------------------------------
PlanningMananger.cs
----------------------------------

using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using System.Reflection;

public class PlanningMananger : MonoBehaviour {
                            public RectTransform introPanel;
                            public RectTransform planningPanel;

                            [Header("Info Pannel")]
                            public Text planningText;
                            public RectTransform goalContainer;
                            public GameObject goalPrefab;
                            public List<GameObject> goalElements = new List<GameObject>();

                            [Header("Planning Panel")]
                            public UIVehichleManager vehichleManager;
                            public UIWeatherManager weatherManager;

                            [Space]

                            public ButtonGroup widthButtonGroup;
                            public ButtonGroup lengthButtonGroup;
                            public ButtonGroup thicknessButtonGroup;
                            public ButtonGroup tonsButtonGroup;
                            public OptionsButtonGroup asphaltTypeButtonGroup;

                            [Space]

                            public GameObject loadingScreen;

                            [Header("Global Variables")]
                            public int numPavers = 0;
                            public int numCompactors = 0;
                            public int numShakers = 0;
                            public int numTrucks = 0;

                            // Use this for initialization
                            void Start () {
        PlanningTutorialManager tutorial_manager =
FindObjectOfType<PlanningTutorialManager>();
        if (tutorial_manager == null) {
            Level level = GameState.Instance.currentLevel;

    #if UNITY_EDITOR
                                    if (level == null) {
                                            level = LevelLoader.LoadLevelFile("level1");
                                    }
    #endif
            AnalyticsManager.StartCounter(level.naam + "_planning_duration");


                                    // Setup the planning
                                    planningText.text = level.intro_text;

                                    // Generate goals
                                    CreateGoalElement("Doelen:");
            Level.Goals goals = level.goals;
            if (goals.minimum_verdichting != -1)
                CreateGoalElement("Minimum Verdichting: {0}%",
                                goals.minimum_verdichting);
            if (goals.maximum_verdichting != -1)
                CreateGoalElement("MaximumVerdichting: {0}%",
                                goals.maximum_verdichting);
            if (goals.gemiddelde_verdichting != -1)
                CreateGoalElement("Gemiddelde Verdichting: {0}%",
                                goals.gemiddelde_verdichting);
            if (goals.maximale_duur_uitvoering != -1)
                CreateGoalElement("Maximale uitvoeringsfase duur: {0} minuten",
                                goals.maximale_duur_uitvoering / 60);

                                   // Setup the planning panel
                                   numPavers     = level.spreidmachines;
                                   numCompactors = level.statische_walsen;
```

```
                                    numShakers    = level.dynamische_walsen;
                                    numTrucks     = level.vrachtwagens;

                                    // Set all the intial values dictated by the level

vehichleManager.addButton.gameObject.SetActive(level.kan_machines_kiezen);
                                    for (int pavers = 0; pavers < numPavers; pavers++) {
                                            vehichleManager.AddVehichle(0);
                                    }
                                    for (int compactors = 0; compactors < numCompactors;
compactors++) {
                                            vehichleManager.AddVehichle(1);
                                    }
                                    for (int shakers = 0; shakers < numShakers; shakers++) {
                                            vehichleManager.AddVehichle(2);
                                    }
                                    for (int trucks = 0; trucks < numTrucks; trucks++) {
                                            vehichleManager.AddVehichle(3);
                                    }

                                    // Set the values for the buttongroups
                                    widthButtonGroup.Value     = level.weg_breedte;
                                    lengthButtonGroup.Value    = level.weg_lengte;
                                    thicknessButtonGroup.Value = level.weg_dikte;
                                    asphaltTypeButtonGroup.SetValue((int)level.weg_lengte);

                                    // Setup weather ui
                                    weatherManager.SetWeather(level.weather);
        } else {
            //
            // Tutorial:
            //
            tutorial_manager.planningText = planningText;

            CreateGoalElement("Doelen:");

            weatherManager.SetWeather(new WeatherCondition() {
                bewolkt     = false,
                temperatuur = 20,
                regen       = false,
                zonnig      = true,
                sneeuw      = false
            });
        }

                                    // Start with info
                                    ReturnToInfo();
                                }

                                // Update is called once per frame
                                void Update () {

                                }

    private void CreateGoalElement(string format_string, float value = 0) {
        GameObject goal_object = (GameObject)Instantiate(goalPrefab);
        goal_object.GetComponent<Transform>()
                                                .SetParent(goalContainer, false);
        goal_object.GetComponentInChildren<Text>().text =
            string.Format(format_string, value);
                                    goalElements.Add(goal_object);
    }

                                public void RecalculateVehichles() {
                                    // Reset old values
                                    numPavers = 0;
                                    numCompactors = 0;
                                    numShakers = 0;
                                    numTrucks = 0;
                                    // Count vehichles
                                    foreach (VehichleType type in vehichleManager.vehichles) {
                                            switch(type) {
                                                    case VehichleType.PAVER: {
                                numPavers++;                    } break;
```

```
                                    case VehichleType.COMPACTOR: {
                            numCompactors++;    } break;
                                    case VehichleType.SHAKER: {
                            numShakers++;               } break;
                                    case VehichleType.TRUCK: {
                            numTrucks++;                } break;
                                }
                        }
                        }

                        public void StartSimulation() {

                            // Load all the chanes in the planning phase into the level.
                            Level level = GameState.Instance.currentLevel;

        AnalyticsManager.StopCounter(level.naam + "_planning_duration");

                        RecalculateVehichles();
                            level.spreidmachines = numPavers;
                            level.statische_walsen = numCompactors;
                            level.dynamische_walsen = numShakers;
                            level.vrachtwagens = numTrucks;

                            level.weg_lengte = lengthButtonGroup.Value;
                            level.weg_breedte = widthButtonGroup.Value;
                            level.weg_dikte = thicknessButtonGroup.Value;
                            level.weg_type = (RoadType)asphaltTypeButtonGroup.Value;

                            // Go to next scene
                            loadingScreen.SetActive(true);
                            StartCoroutine(
                                    SwitchScenes(
                                            SceneManager.LoadSceneAsync("execution")
                            ));
                        }

                        public void GotoPlanning() {
                            planningPanel.gameObject.SetActive(true);
                            introPanel.gameObject.SetActive(false);
                        }

                        public void ReturnToInfo() {
        AnalyticsManager.IncrementKey("return_to_info");
                            planningPanel.gameObject.SetActive(false);
                            introPanel.gameObject.SetActive(true);
                        }

                        IEnumerator SwitchScenes(AsyncOperation sceneSwitch) {
                            yield return sceneSwitch.isDone;
                            loadingScreen.SetActive(false);
                        }
}
----------------------------------
PlanningTutorialManager.cs
----------------------------------

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class PlanningTutorialManager : MonoBehaviour {

                        [SerializeField]
                        private int tutorial_state = 0;
                        public GameObject textBubble;
                        public GameObject tutorialTextPanel;
                        public TextBubblePositioner textBubbleManager;

                        private string[] tutorial_texts = {
                            "Welkom bij de uitleg voor het asfalt constructie simulator
spel. In deze modus wordt de werking van het spel uitgelegd. Je gaat straks leren hoe je
machines toevoegd aan het werk, hoe je de parameters van de toplaag kunt instellen, hoe je het
```

```
asfalttype insteld en hoe je ziet wat voor weer het gaat worden. Druk op 'Volgende' om verder
te gaan.",
                                "Je bevind je nu in de planningsfase van het asfalteer
proces. In deze fase wordt het voorbereidend werk gedaan voor het asfalteren. We zullen alles
stap voor stap doorlopen zodat je straks zelf aan de slag kunt.",
                                "We gaan eerst kijken naar welke machines we nodig zullen
hebben bij het maken van een simpel wegdek. We hebben hiervoor een spreidmachine, een wals en
een vrachtwagen nodig.",
                                "We gaan nu een aantal machines toevoegen. Klik op de '+'
knop en selecteer een spreidmachine.",
                                "Voeg nu op dezelfde manier een statische wals en een
vrachtwagen toe.",
                                "Nu we weten welke machines we nodig hebben, gaan we kijken
naar het wegdek. We moeten een weg asfalteren van vijf meter breed, honderd meter lang en vijf
centimeter dik.",
                                "Je kunt de getallen in de vakjes aanpassen door op de '+' en
'-' knoppen te drukken.",
                                "Goed, dat zijn de dimensies van de weg. Nu moeten we
beslissen welk type asfalt we gaan gebruiken. Voor dit test project hebben we AC16 SURF
nodig.",
                                "Loop door de opties heen door op de '<' en '>' knoppen te
drukken.",
                                "Temperatuur en neerslag hebben veel invloed op de kwaliteit
van het asfalt. Het is dus belangrijk om te kijken naar het weerbericht. Dit weerbericht is te
zien in de rechterbovenhoek van het scherm.",
                                "Hier kun je zien wat de <b>verwachte</b> weersomstandigheden
zullen zijn tijdens het asfalteren. Echter, het weer kan altijd veranderen.",
                                "Druk op klaar om naar de volgende fase te gaan."
                            };

                            [Header("Targets")]
                            public RectTransform addVehichleButton;
                            public RectTransform widthButton;
                            public RectTransform roadTypeButton;
                            public RectTransform weatherElement;
                            public RectTransform doneButton;

                            // Private references
                            private PlanningMananger planner;
                            private float road_width;
                            private float road_length;
                            private float road_thickness;
                            private RoadType road_type;

                            public ButtonGroup[] asphalt_dimension_buttons;
                            public OptionsButtonGroup asphalt_type_button;
                            public Button done_button;

    public Text planningText;

    // Use this for initialization
    void Start () {
                                planner = FindObjectOfType<PlanningMananger>();
                                textBubbleManager =
textBubble.GetComponent<TextBubblePositioner>();
                                done_button = doneButton.GetComponent<Button>();
                                done_button.interactable = false;

                                // Find al button groups
                                road_width = asphalt_dimension_buttons[0].Value;
                                road_length = asphalt_dimension_buttons[1].Value;
                                road_thickness = asphalt_dimension_buttons[2].Value;
                                road_type = (RoadType)asphalt_type_button.Value;


                            asphalt_dimension_buttons[0].onValueChanged.AddListener((width)=>
{
                                        road_width = width;
                                        NextTutorialStep();
                                });

                            asphalt_dimension_buttons[1].onValueChanged.AddListener((length)=
>{
                                        road_length = length;
```

```
                                        NextTutorialStep();
                                    });

                                asphalt_dimension_buttons[2].onValueChanged.AddListener((thicknes
s)=>{
                                        road_thickness = thickness;
                                        NextTutorialStep();
                                    });

                                asphalt_type_button.onValueChanged.AddListener((option_index)=>{
                                        road_type = (RoadType)option_index;
                                        NextTutorialStep();
                                    });


                                    NextTutorialStep();
                                }

                                // Update is called once per frame
                                void Update () {

                                }

                                public void NextTutorialStep() {

                                    // Reset all
                                    tutorialTextPanel.SetActive(false);
                                    textBubble.SetActive(false);

                                    switch(tutorial_state) {
                                        case 0: {
                    planningText.text = tutorial_texts[tutorial_state];
                    tutorial_state++;
                } break;

                                        case 1:
                                        case 2:
                                        case 5:
                                        case 7:
                                        case 9: {
                                                tutorialTextPanel.SetActive(true);

                                tutorialTextPanel.GetComponentInChildren<Text>().text =
                                                        tutorial_texts[tutorial_state] + "
Klik op volgende om verder te gaan.";
                                                tutorial_state++;
                                        } break;

                                        case 3: {
                                                textBubble.SetActive(true);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        tutorial_texts[tutorial_state];

                                textBubbleManager.SetUITargetPosition(addVehichleButton);
                                                tutorial_state++;
                                        } break;

                                        case 4: {
                                                textBubble.SetActive(true);

                                textBubble.GetComponentInChildren<Text>().text =
                                                        tutorial_texts[tutorial_state];

                                textBubbleManager.SetUITargetPosition(addVehichleButton);
                                                planner.RecalculateVehichles();
                                                if (planner.numPavers == 1 &&
planner.numCompactors == 1 &&
                                                        planner.numTrucks == 1 &&
planner.numShakers == 0) {

                                addVehichleButton.gameObject.SetActive(false);      // Disable
adding more vehichles
                                                        tutorial_state++;
```

```
                            StartCoroutine(WaitForNextTutorial(0.6f));
                                    } else if (planner.numPavers > 1 ||
planner.numCompactors > 1 ||
                                                                    planner.numTrucks >
1 || planner.numShakers > 0) {

                            textBubble.GetComponentInChildren<Text>().text =
                                                                    "Je hebt van een van de
machines te veel geselecteerd. Je kunt ze verweideren door op '+' en dan op 'verweider
laatste' te klikken.";
                                    }
                                } break;

                                case 6: {
                                    textBubble.SetActive(true);

                            textBubble.GetComponentInChildren<Text>().text =
                                            tutorial_texts[tutorial_state];

                            textBubbleManager.SetUITargetPosition(widthButton);
                                    if (road_width == 5 && road_length == 0.1f &&
road_thickness == 5) {
                                            // Disable buttons
                                            foreach (ButtonGroup bg in
asphalt_dimension_buttons) {
                                                    bg.Interactable = false;
                                            }
                                            tutorial_state++;

                            StartCoroutine(WaitForNextTutorial(0.6f));
                                    } else if (road_width > 5 || road_length >
0.1f || road_thickness > 5) {

                            textBubble.GetComponentInChildren<Text>().text =
                                                                    "Een van de weg dimensies is
te hoog. De lengte moet 100 meter zijn, de breedte 5 meter en de dikte 5cm zijn.";
                                    }
                                } break;

                                case 8: {
                                    textBubble.SetActive(true);

                            textBubble.GetComponentInChildren<Text>().text =
                                            tutorial_texts[tutorial_state];

                            textBubbleManager.SetUITargetPosition(roadTypeButton);

                                    if (road_type == RoadType.AC16_SURF) {
                                            asphalt_type_button.Interactable =
false;
                                            tutorial_state++;

                            StartCoroutine(WaitForNextTutorial(0.6f));
                                    }
                                } break;

                                case 10: {
                                    textBubble.SetActive(true);

                            textBubble.GetComponentInChildren<Text>().text =
                                            tutorial_texts[tutorial_state];

                            textBubbleManager.SetUITargetPosition(weatherElement);
                                    tutorial_state++;
                                    StartCoroutine(WaitForNextTutorial(5));
                                } break;

                                case 11: {
                                    done_button.interactable = true;
                                    textBubble.SetActive(true);

                            textBubble.GetComponentInChildren<Text>().text =
                                            tutorial_texts[tutorial_state];
```

```
                                        tutorial_state++;

                            textBubbleManager.SetUITargetPosition(doneButton);
                                    } break;
                                }
                            }

                            IEnumerator WaitForNextTutorial(float wait_time) {
                                yield return new WaitForSeconds(wait_time);
                                NextTutorialStep();
                            }

                            public void StartSimulation() {
                                // Load all the chanes in the planning phase into the level.
                                Level level = GameState.Instance.currentLevel;
        if (level == null) {
            level = GameState.Instance.currentLevel = new Level();
        }
                                planner.RecalculateVehichles();
                                level.spreidmachines    = planner.numPavers;
                                level.statische_walsen  = planner.numCompactors;
                                level.dynamische_walsen = planner.numShakers;
                                level.vrachtwagens      = planner.numTrucks;

                                level.weg_lengte  = planner.lengthButtonGroup.Value;
                                level.weg_breedte = planner.widthButtonGroup.Value;
                                level.weg_dikte   = planner.thicknessButtonGroup.Value;
                                level.weg_type    =
(RoadType)planner.asphaltTypeButtonGroup.Value;

                                // Go to next scene
                                planner.loadingScreen.SetActive(true);
                                StartCoroutine(
                                        SwitchScenes(

                            SceneManager.LoadSceneAsync("execution_tutorial")
                                ));
                            }

                            IEnumerator SwitchScenes(AsyncOperation sceneSwitch) {
                                yield return sceneSwitch.isDone;
                                planner.loadingScreen.SetActive(false);
                            }
}
-----------------------------------
RoadCompactor.cs
-----------------------------------

using UnityEngine;
using System.Collections;
using System;

public class RoadCompactor : MonoBehaviour {

    public float minSpeed = 0;
    public float maxSpeed = 13;
    [Tooltip("In Km/h")]
    public float speed = 0;
    public float shiftSpeed = 0.1f;
    public float paverWidth;

    public float triangleDistance = 0.5f;
    public float compactionPerPass = 10;
    private Vector3 prev_pos;

    public Transform waypointPrefab;
    private Transform target;
    private Vector3 velocity = Vector3.zero;
    private float target_z = 0;
    public Transform startTarget;
    public Transform endTarget;

    Vector3[] prevPoints = new Vector3[2];
    public Transform[] compactPoints = new Transform[2];
```

```
    TextureManager road;

    // Use this for initialization
    void Start () {
        prev_pos = transform.position;
        prevPoints[0] = compactPoints[0].position;
        prevPoints[1] = compactPoints[1].position;

        startTarget = (Transform)Instantiate(waypointPrefab,
            transform.position + transform.forward * 4, Quaternion.identity);
        endTarget = (Transform)Instantiate(waypointPrefab,
            transform.position - transform.forward * 4, Quaternion.identity);
        endTarget.GetComponent<WaypointManager>().previous = startTarget;
        target = startTarget;

        endTarget.gameObject.SetActive(false);
        startTarget.gameObject.SetActive(false);

        paverWidth = Vector3.Distance(compactPoints[0].position, compactPoints[1].position);

        velocity = new Vector3(speed, 0, speed * shiftSpeed);
                            }

                            // Update is called once per frame
                            void Update () {
        if (ExecutionManager.isPaused) return;

        if (Mathf.Abs(transform.position.x - target.position.x) < 0.3f ||
            Mathf.Sign(velocity.x) == Mathf.Sign(transform.position.x - target.position.x)) {
            if (target == startTarget) {
                target = endTarget;
            } else {
                target = startTarget;
            }
            UpdateCompaction();
        }

        // Move ping pong along the width of the road.
        if (road && Mathf.Abs(target_z - transform.position.z) < 0.1f) {
            if (target_z < road.transform.position.z) {
                target_z = road.transform.position.z + (road.roadWidth / 2) - (paverWidth /
4);
            } else {
                target_z = road.transform.position.z - (road.roadWidth / 2) + (paverWidth /
4);
            }
        }

        // Move towards correct position

        velocity = new Vector3(speed * Mathf.Sign(target.position.x - transform.position.x),
0,
                               ((speed > 0)?shiftSpeed : 0) * Mathf.Sign(target_z -
transform.position.z));
        transform.position += velocity * Time.deltaTime *
ExecutionManager.simulationTimeScale;

        if (road && Vector3.Distance(prev_pos, transform.position) > triangleDistance) {
            UpdateCompaction();
        }
                            }

    void UpdateCompaction() {
        if (road == null) return;
        road.AddCompactTriangle(compactPoints[0].position,
                                compactPoints[1].position,
                                prevPoints[0], compactionPerPass);
        road.AddCompactTriangle(prevPoints[0],
                                prevPoints[1],
                                compactPoints[1].position, compactionPerPass);

        prevPoints[0] = compactPoints[0].position;
        prevPoints[1] = compactPoints[1].position;
```

```
        prev_pos = transform.position;
    }

    public float SetSpeed(float speed) {
        this.speed = speed;
        return speed;
    }

    public void SetActivateWaypoints(bool value) {
        startTarget.gameObject.SetActive(value);
        endTarget.gameObject.SetActive(value);
    }

    void OnTriggerEnter(Collider collider) {
        if (collider) {
            if (collider.tag == Tags.ROAD) {
                road = collider.GetComponent<TextureManager>();
                target_z = road.transform.position.z + (road.roadWidth / 2) - (paverWidth /
2);
                if (transform.position.z > road.transform.position.z) velocity.z *= -1;
            } else if (collider.tag == Tags.COMPACTOR) {
                GameState.Instance.results.compactorCollisions++;
            } else if (collider.tag == Tags.PAVER) {
                GameState.Instance.results.paverCollisions++;
            }
        }
    }
}
```

ScoreCanvasManager.cs

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class ScoreCanvasManager : MonoBehaviour {

    public Transform contentContainer;
    public GameObject statsContainerPrefab;
    public GameObject textureContainerPrefab;
    public GameObject starsContainerPrefab;
    public GameObject goalContainerPrefab;

    GameState game_state;
    public List<RectTransform> headers = new List<RectTransform>();
    public List<RectTransform> textureContainers = new List<RectTransform>();
    public List<RectTransform> statsContainers = new List<RectTransform>();
    public List<RectTransform> starsContainers = new List<RectTransform>();
    public List<RectTransform> goalContainers = new List<RectTransform>();

                            // Use this for initialization
                            void Start () {
        if (!game_state) {
            game_state = GameState.Instance;
        }

        if (game_state) {
            GameState.ExecutionResults results = game_state.results;
            Level level = game_state.currentLevel;
            Level.Goals goals = null;
            if (level != null) goals = level.goals;
#if UNITY_EDITOR
            if (level == null) {
                level = game_state.currentLevel = LevelLoader.LoadLevelFile("level1");
                goals = level.goals;
            }
            if (results == null) {
                results = new GameState.ExecutionResults(1);
            }
#endif
            CreateStatHeader("Planningsfase", true);
```

```
            CreateStatHeader("Machines", false);
            CreateStatsContainer(contentContainer, "Spreidmachines",      level.spreidmachines
+ "x");
            CreateStatsContainer(contentContainer, "Walsers",
level.statische_walsen + "x");
            CreateStatsContainer(contentContainer, "Trillende Walsers",
level.dynamische_walsen + "x");
            CreateStatsContainer(contentContainer, "Vrachtwagens",        level.vrachtwagens +
"x");

            CreateStatHeader("Weg", false);
            CreateStatsContainer(contentContainer, "Weg Breedte",         level.weg_breedte +
"m");
            CreateStatsContainer(contentContainer, "Weg Lengte",          level.weg_breedte +
"km");
            CreateStatsContainer(contentContainer, "Weg Dikte",           level.weg_breedte +
"cm");

            CreateStatHeader("Weer", false);
            CreateStatsContainer(contentContainer, "Temperatuur",
level.weather.temperatuur + "°C");
            if (level.weather.bewolkt)
                CreateStatsContainer(  contentContainer, "Bewolkt", "");
            if (level.weather.regen)
                CreateStatsContainer(  contentContainer, "Regen", "");
            if (level.weather.zonnig)
                CreateStatsContainer(  contentContainer, "Zonnig", "");
            if (level.weather.sneeuw)
                CreateStatsContainer(  contentContainer, "Sneeuw", "");

            int planning_stars = level.planning_sterren;
            if (planning_stars > 0) {
                CreateStarContainer("Behaalde Sterren Planningsfase", planning_stars, 1);
            }

            CreateStatHeader("Uitvoeringsfase", true);

            CreateStatsContainer(contentContainer,
                                 "Uitvoeringsfase duur",
                                 (results.totalTime * 0.0166667f) + " Minuten");
            CreateGoalContainer("Uitvoeringsfase duur",
                                goals.maximale_duur_uitvoering, results.totalTime);

            CreateStatsContainer(contentContainer,
                                 "Spreidmachine Botsingen", results.paverCollisions);
            CreateStatsContainer(contentContainer,
                                 "Wals Botsingen", results.compactorCollisions);

            CreateStatHeader("Verdichting", false);
            CreateTextureContainer(results.compactionTexture, 0, level.weg_lengte);

            CreateStatsContainer(contentContainer,
                                 "Gemiddelde Verdichting", results.averageCompaction + "%");
            CreateGoalContainer("Gemiddelde Verdichting",
                                results.averageCompaction, goals.gemiddelde_verdichting);

            CreateStatsContainer(contentContainer, "Percentage Teveel Verdicht",
results.averageExcess + "%");
            CreateStatsContainer(contentContainer, "Percentage Onverdicht",
results.averageUncompacted + "%");

            int execution_stars = level.uitvoering_sterren;
            if (execution_stars > 0) {
                CreateStarContainer("Behaalde Sterren Planningsfase", execution_stars, 1);
            }

            CreateStatHeader("Totaal Aantal Sterren", true);
            int total_stars = level.planning_sterren + level.uitvoering_sterren;
            CreateStarContainer("", total_stars, 3);
        }
                            }

    // Update is called once per frame
    void Update () {
```

```
                            }

    RectTransform CreateStatHeader(string property_label, bool big = true) {
        RectTransform stat_container =
Instantiate(statsContainerPrefab).GetComponent<RectTransform>();
        stat_container.SetParent(contentContainer, false);
        stat_container.GetComponent<LayoutElement>().preferredHeight = (big)? 40 : 30;
        Text label_text = stat_container.Find("Label").GetComponent<Text>();
        label_text.text = property_label;
        label_text.fontStyle = FontStyle.Bold;
        stat_container.Find("Value").GetComponent<Text>()
            .text = "";

        headers.Add(stat_container);
        return stat_container;
    }

    RectTransform CreateGoalContainer(string label, float amount, float goal_amount) {
        if (goal_amount != -1 && amount != -1) {
            bool achieved = (amount >= goal_amount);

            RectTransform goal_container =
Instantiate(goalContainerPrefab).GetComponent<RectTransform>();
            goal_container.SetParent(contentContainer, false);

            Transform stats_transform = goal_container.Find("Stats");
            stats_transform.GetComponent<Image>().color = (achieved)?Color.green : Color.red;
            RectTransform a = CreateStatsContainer(stats_transform, label, amount);
            RectTransform b = CreateStatsContainer(stats_transform, "Vereiste " + label,
goal_amount);
            RectTransform c = CreateStatsContainer(stats_transform, "Gehaald", (achieved)?
"Ja" : "Nee");

            a.GetComponent<Image>().color =
                b.GetComponent<Image>().color =
                c.GetComponent<Image>().color = (achieved)? Color.green : Color.red;

            goalContainers.Add(goal_container);
            return goal_container;
        }
        return null;
    }

    RectTransform CreateStatsContainer(Transform parent, string property_label,
                               object property_value) {
        RectTransform stat_container =
Instantiate(statsContainerPrefab).GetComponent<RectTransform>();
        stat_container.SetParent(parent, false);
        stat_container.Find("Label").GetComponent<Text>()
            .text = property_label;
        stat_container.Find("Value").GetComponent<Text>()
            .text = property_value.ToString();

        statsContainers.Add(stat_container);
        return stat_container;
    }

    RectTransform CreateTextureContainer(Texture2D compactionTexture, float start_length,
float end_length) {
        if (!compactionTexture) return null;

        RectTransform texture_container =
Instantiate(textureContainerPrefab).GetComponent<RectTransform>();
        texture_container.SetParent(contentContainer, false);
        texture_container.Find("Label/Text (Start)").GetComponent<Text>().text = start_length
+ "m";
        texture_container.Find("Label/Text (End)").GetComponent<Text>().text = end_length +
"m";

        // Transpose texture
        Texture2D transposed_texture;
        {
            int w = compactionTexture.width;
```

```
            int h = compactionTexture.height;
            transposed_texture = new Texture2D(h, w);
            Color[] source_pixels = compactionTexture.GetPixels();
            Color[] transposed_pixels = transposed_texture.GetPixels();
            for (int i = 0; i < w; i++) {
                for (int j = 0; j < h; j++) {
                    transposed_pixels[j + i * h] = source_pixels[i + j * w];
                }
            }
            transposed_texture.SetPixels(transposed_pixels);
            transposed_texture.Apply();
        }
        texture_container.GetComponentInChildren<RawImage>().texture = transposed_texture;

        textureContainers.Add(texture_container);
        return texture_container;
    }

    RectTransform CreateStarContainer(string label_text, int max_stars, int gained_stars) {
        RectTransform star_container =
Instantiate(starsContainerPrefab).GetComponent<RectTransform>();
        star_container.SetParent(contentContainer, false);

        // Generate stars
        UILevelElement star_manager = star_container.GetComponent<UILevelElement>();
        star_manager.Name = label_text;
        star_manager.SetStars(max_stars, gained_stars);
        starsContainers.Add(star_container);

        return star_container;
    }

    void GotoMainMenu() {
        SceneManager.LoadScene("main_menu");
    }
}
----------------------------------
SettingsManager.cs
----------------------------------

using UnityEngine;
using UnityEngine.UI;
using System.Collections.Generic;
using System.IO;

public class SettingsManager : MonoBehaviour {

                            public GameSettings settings;

                            public Toggle fullscreen_toggle;
                            public Dropdown resolution_dropdown;
                            public Dropdown texture_dropdown;
                            public Dropdown vsync_dropdown;
                            public Dropdown aa_dropdown;
                            public Dropdown shadow_dropdown;
                            public Slider music_slider;
                            public Slider sfx_slider;

                            Resolution[] resolutions;

                            void OnEnable () {
                                settings = new GameSettings ();

                                fullscreen_toggle.onValueChanged.AddListener (delegate {
OnFullscreenToggle(); });
        settings.fullscreen = Screen.fullScreen = fullscreen_toggle.isOn;

        resolution_dropdown.onValueChanged.AddListener (delegate { OnResolutionDropdown(); });
                                texture_dropdown.onValueChanged.AddListener (delegate {
OnTextureDropdown(); });
                                vsync_dropdown.onValueChanged.AddListener (delegate {
OnVSyncDropdown(); });
                                aa_dropdown.onValueChanged.AddListener (delegate {
OnAADropdown(); });
```

```
                                shadow_dropdown.onValueChanged.AddListener (delegate {
OnShadowDropdown(); });

                                music_slider.onValueChanged.AddListener(delegate {
OnMusicSlider(); });
                                sfx_slider.onValueChanged.AddListener(delegate {
OnSFXSlider(); });

                                resolutions = Screen.resolutions;
        // Init resolution dropdown
        resolution_dropdown.ClearOptions();
        List<Dropdown.OptionData> options = new List<Dropdown.OptionData>();

        settings.resolution_index = resolutions.Length - 1;

        for (int resolution_index = 0; resolution_index < resolutions.Length;
resolution_index++) {
            options.Add(new Dropdown.OptionData(resolutions[resolution_index].ToString()));
        }
        resolution_dropdown.AddOptions(options);

        LoadSettings();
                            }

                            // Use this for initialization
                            void Start () {

                            }

                            // Update is called once per frame
                            void Update () {

                            }

                            public void OnFullscreenToggle () {
        settings.fullscreen = Screen.fullScreen = fullscreen_toggle.isOn;
    }

    public void OnResolutionDropdown () {
        settings.resolution_index = resolution_dropdown.value;
        Resolution r = resolutions[settings.resolution_index];
        Screen.SetResolution(r.width, r.height, settings.fullscreen, r.refreshRate);
                            }

                            public void OnTextureDropdown () {
        settings.texture_quality = QualitySettings.masterTextureLimit =
texture_dropdown.value;
                            }

                            public void OnVSyncDropdown () {
        settings.vsync = QualitySettings.vSyncCount = vsync_dropdown.value;
                            }

                            public void OnAADropdown () {
        settings.antialiasing = QualitySettings.antiAliasing = (int)Mathf.Pow(2f,
aa_dropdown.value);
    }

                            public void OnShadowDropdown () {
        settings.shadow_quality = shadow_dropdown.value;
        QualitySettings.shadowResolution = (ShadowResolution)shadow_dropdown.value;
                            }

                            public void OnMusicSlider () {
        settings.music_volume = AudioListener.volume = music_slider.value;
                            }

                            public void OnSFXSlider () {
        settings.sfx_volume = AudioListener.volume = sfx_slider.value;
    }

    public void SaveSettings () {
        string json_data = JsonUtility.ToJson(settings, true);
        File.WriteAllText(Application.persistentDataPath + "/settings.txt", json_data);
```

```
    }

    public void LoadSettings() {
        string file_path = Application.persistentDataPath + "/settings.txt";
        print("Loading settings: " + file_path);
        if (File.Exists(file_path)) {
            settings = JsonUtility.FromJson<GameSettings>(File.ReadAllText(file_path));

            // Note: Changing the data also fires the events
            fullscreen_toggle.isOn = settings.fullscreen;
            resolution_dropdown.value = settings.resolution_index;
            texture_dropdown.value = settings.texture_quality;
            vsync_dropdown.value = settings.vsync;
            aa_dropdown.value = settings.antialiasing;
            shadow_dropdown.value = settings.shadow_quality;
            music_slider.value = settings.music_volume;
            sfx_slider.value = settings.sfx_volume;

            Screen.fullScreen = settings.fullscreen;
            OnResolutionDropdown();
            resolution_dropdown.RefreshShownValue();
        } else {
            SaveSettings();
        }
    }
}
----------------------------------
TextureManager.cs
----------------------------------

using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

public class TextureManager : MonoBehaviour {

    public int pixelsPerWorldUnit = 10;
    public Material roadMaterial;
    public Material heatMaterial;
    public Material compactionMaterial;

    [Space]
    public Gradient heatGradient;
    public Gradient compactionGradient;

    public float roadWidth;
    private float road_length;
    private float road_total_length;

    public enum RenderMode {
        ROAD_TEXTURE,
        HEAT_TEXTURE,
        COMPACT_TEXTURE,
        SIZE
    }
    public RenderMode currentMode = RenderMode.ROAD_TEXTURE;

    [HeaderAttribute("Render Attributes")]
    private int texture_height;
    private int texture_width;

    public Texture2D heatTexture;
    private Color[] heat_tex_pixels;
    private int[] temperature_map;

    public Texture2D compactTexture;
    public float[] compactValues;
    private Color[] compact_tex_pixels;

    // These points are used for updating the heat dispersion
    private List<float> time_points = new List<float>();
    private List<float> length_points = new List<float>();
```

```
    private MeshFilter filter;
    private new MeshRenderer renderer;
    private new BoxCollider collider;
    private Mesh mesh;

    public float minimumCompactionTemperature = 90;
    private float[] heat_data;

    // Use this for initialization
                            void Start () {
        StartCoroutine(DelayedUpdate(1.0f));

        float simulation_temperature = GameState.Instance.currentLevel.weather.temperatuur;
        heat_data = DataManager.Instance.GetDataPoints(simulation_temperature);
                            }

    /// <summary>
    /// Creates a road mesh, and a texture based on the paver's width and total distance.
    /// </summary>
    /// <param name="leftPos">The leftmost point of the road.</param>
    /// <param name="rightPos">The rightmost point of the road.</param>
    /// <param name="totalDistance">The length of the road.</param>
    public void StartRoad(Vector3 leftPos, Vector3 rightPos, float totalDistance) {
        filter   = GetComponent<MeshFilter>();
        collider = GetComponent<BoxCollider>();
        renderer = GetComponent<MeshRenderer>();

        // Set material & mesh
        renderer.material = roadMaterial;
        mesh = filter.mesh = CreateRoadMesh(leftPos, rightPos);

        // Clear Values
        time_points.Clear();
        length_points.Clear();

        road_length       = 0;
        road_total_length = totalDistance;
        roadWidth         = Vector3.Distance(leftPos, rightPos);

        // Create a texture
        texture_width  = (int)(roadWidth * pixelsPerWorldUnit);
        texture_height = (int)(totalDistance * pixelsPerWorldUnit);
        texture_width  = Mathf.Min(texture_width, 4096);
        texture_height = Mathf.Min(texture_height, 4096);

        heatTexture      = new Texture2D(texture_width, texture_height, TextureFormat.ARGB32,
false);
        heatTexture.name = "Temperature Map";
        heat_tex_pixels  = heatTexture.GetPixels();
        heatMaterial.SetTexture("_MainTex", heatTexture);

        // Heat map
        temperature_map  = new int[texture_width * texture_height];

        compactTexture            = new Texture2D(texture_width, texture_height,
TextureFormat.ARGB32, false);
        compactTexture.name       = "Compaction Texture";
        compactTexture.filterMode = FilterMode.Point;
        // init pixels
        compact_tex_pixels = compactTexture.GetPixels();
        for (int pixel_index = 0; pixel_index < compact_tex_pixels.Length; pixel_index++) {
            compact_tex_pixels[pixel_index] = new Color(0, 0, 0, 1);
        }
        compactTexture.SetPixels(compact_tex_pixels);
        compactTexture.Apply();
        compactionMaterial.SetTexture("_MainTex", compactTexture);

        ContourX = new int[texture_height, 2];
        compactValues = new float[texture_width * texture_height];

        UpdateThermalTexture();

        SetRenderMode(0);
    }
```

```
    /// <summary>
    /// Creates a mesh for the road.
    /// </summary>
    /// <param name="leftPos">Left start position of the road</param>
    /// <param name="rightPos">Right start position of the road</param>
    /// <returns></returns>
    private Mesh CreateRoadMesh(Vector3 leftPos, Vector3 rightPos){
        Mesh result = new Mesh();
        result.name = "Road Mesh";
        // Get the vertices and triangles from the mesh
        Vector3[] vertices = result.vertices;
        int[] triangles = result.triangles;

        // Add two vetices
        // *----x----*
        AddToArray(ref vertices, leftPos - transform.position + Vector3.up * 0.01f);
        AddToArray(ref vertices, rightPos - transform.position + Vector3.up * 0.01f);
        // Add two vetices
        // *----x----*
        AddToArray(ref vertices, leftPos - transform.position + Vector3.up * 0.01f);
        AddToArray(ref vertices, rightPos - transform.position + Vector3.up * 0.01f);

        // Here connect the vertices to the path with triangels.
        int i = triangles.Length;
        Array.Resize(ref triangles, triangles.Length + 6);
        // Make the two triangles
        triangles[i++] = vertices.Length - 3;  // Tri 1
        triangles[i++] = vertices.Length - 4;
        triangles[i++] = vertices.Length - 2;

        triangles[i++] = vertices.Length - 3;  // Tri 2
        triangles[i++] = vertices.Length - 2;
        triangles[i++] = vertices.Length - 1;

        // Update and optimize mesh
        result.SetVertices(new List<Vector3>(vertices));
        result.SetTriangles(triangles, 0);

        // Update the real mesh
        return result;
    }

    /// <summary>
    /// Converts world points to texture space points.
    /// </summary>
    /// <param name="point">The point to convert</param>
    /// <param name="texCoordinate">The converted point, it is only correct when the function
returns true.</param>
    /// <returns>Weither the point lies on the texture.</returns>
    private bool WorldToTextureSpace(Vector3 point, out Vector2 texCoordinate) {
        Vector3[] vertices = mesh.vertices;
        float mesh_width  = Mathf.Abs(vertices[1].z - vertices[0].z);
        float mesh_height = road_total_length;
        Vector3 vertex_world = transform.TransformPoint(vertices[0]);
        float tex_x = -(point.z - vertex_world.z) / mesh_width;
        float tex_y = (point.x - vertex_world.x) / mesh_height;
        texCoordinate = new Vector2(tex_x * texture_width,
                                    tex_y * texture_height);
        return !(tex_x < 0 || tex_x >= texture_width ||
                 tex_y < 0 || tex_y >= texture_height);
    }

    /// <summary>
    /// Adds a certain amount of compaction at a trinagular field.
    /// </summary>
    /// <param name="position1"></param>
    /// <param name="position2"></param>
    /// <param name="position3"></param>
    /// <param name="compactionPerPass"></param>
    public void AddCompactTriangle(Vector3 position1, Vector3 position2, Vector3 position3,
float compactionPerPass) {
        Vector2 tex_pos1, tex_pos2, tex_pos3;
        bool a = WorldToTextureSpace(position1, out tex_pos1);
```

```
        bool b = WorldToTextureSpace(position2, out tex_pos2);
        bool c = WorldToTextureSpace(position3, out tex_pos3);
        if (a || b || c) {
            // If all triangles are on the texture
            DrawTriangle(compact_tex_pixels, tex_pos1, tex_pos2, tex_pos3, compactionPerPass);
            compactTexture.SetPixels(compact_tex_pixels);
            compactTexture.Apply();
        }
    }

    /// <summary>
    /// Updates the road textures every duration.
    /// </summary>
    /// <param name="duration">The time between updates.</param>
    /// <returns></returns>
    IEnumerator DelayedUpdate(float duration) {
        UpdateThermalTexture();
        yield return new WaitForSeconds(duration);
        yield return StartCoroutine(DelayedUpdate(duration));
    }

    /// <summary>
    /// Updates the heat values of a thermal map texture
    /// </summary>
    /// <param name="texture"></param>
    private void UpdateThermalTexture() {
        // We don't have to update the road if the simulation is paused (duh..)
        if (time_points.Count > 0 && !ExecutionManager.isPaused) {
            int total_offset_in_pixels = 0;
            float pixels_per_meter = (texture_height / road_total_length);
            for (int time_point_index = 0; time_point_index < time_points.Count - 1;
time_point_index++) {
                // Update temperature points
                float delta_time = (ExecutionManager.simulationTimeSinceStart -
time_points[time_point_index]);
                float traveled_distance = length_points[time_point_index];
                int pixels_to_update = (int)(traveled_distance * pixels_per_meter);

                                                int duration_minutes = (int)(delta_time /
60.0f);
                float heat_value;
                if (duration_minutes < heat_data.Length)
                {
                                                    heat_value = heat_data[duration_minutes];
                }
                else
                {
                    // Take the linear difference of the last two points in the map and
linearly extrapolate
                    float last_delta = heat_data[heat_data.Length - 1] -
heat_data[heat_data.Length - 2];
                    heat_value = heat_data[heat_data.Length - 1] - last_delta *
(duration_minutes - heat_data.Length);
                    // Never get below outside temperature
                    heat_value = Mathf.Max(heat_value,
GameState.Instance.currentLevel.weather.temperatuur);
                }
                Color color = heatGradient.Evaluate(1 - (heat_value / (heat_data[0])));

                                                for (int y = 0; y < pixels_to_update; y++) {
                    for (int x = 0; x < texture_width; x++) {
                        if (x + total_offset_in_pixels * texture_width >=
temperature_map.Length) continue;
                        temperature_map[x + total_offset_in_pixels * texture_width] =
(int)heat_value;
                        heat_tex_pixels[x + total_offset_in_pixels * texture_width] = color;
                    }
                    total_offset_in_pixels++;
                }
            }
            heatTexture.SetPixels(heat_tex_pixels);
            heatTexture.Apply();
        }
    }
```

```
    /// <summary>
    /// Updates the road's verticies and uv based on the traveled length.
    /// </summary>
    /// <param name="leftPos">The leftmost point of the road.</param>
    /// <param name="rightPos">The rightmost point of the road.</param>
    /// <param name="added_road_length">Added travel length of the update.</param>
    public void UpdateRoadLength(Vector3 leftPos, Vector3 rightPos, float added_road_length,
float hmaTemperature) {
        road_length += added_road_length;
        collider.size = new Vector3(road_length, 1, Vector3.Distance(leftPos, rightPos));
        collider.center = new Vector3(mesh.vertices[0].x + (road_length / 2), 0, 0);

        time_points.Add(ExecutionManager.simulationTimeSinceStart);
        length_points.Add(added_road_length);

        Vector3[] vertices = mesh.vertices;
        Vector3 left = leftPos - transform.position;
        Vector3 right = rightPos - transform.position;
        left.y = 0;
        right.y = 0;
        vertices[vertices.Length - 2] = left + Vector3.up * 0.05f;
        vertices[vertices.Length - 1] = right + Vector3.up * 0.05f;
        mesh.SetVertices(new List<Vector3>(vertices));
        mesh.RecalculateBounds();

        // UV's: we just want to show the textue up to the percentage of which we traveled.
        float fraction_of_length = road_length / road_total_length;
        Vector2[] uvs = new Vector2[vertices.Length];
        uvs[0] = new Vector2(0, 0);
        uvs[1] = new Vector2(1, 0);
        uvs[2] = new Vector2(0, fraction_of_length);
        uvs[3] = new Vector2(1, fraction_of_length);
        mesh.uv = uvs;
    }

    // Sets the render mode
    // 0: Road
    // 1: Heat
    // 2: Compaction
    // 3: Heat & Compaction
    public void SetRenderMode(int value) {
        if (value >= 0 && value < (int)RenderMode.SIZE) {
            RenderMode mode = (RenderMode)value;
            if (mode != currentMode) {
                currentMode = mode;
                switch (currentMode) {
                    case RenderMode.ROAD_TEXTURE: {
                        renderer.material = roadMaterial;
                    } break;

                    case RenderMode.HEAT_TEXTURE: {
                        renderer.material = heatMaterial;
                    } break;

                    case RenderMode.COMPACT_TEXTURE: {
                        renderer.material = compactionMaterial;
                    } break;
                }
            }
        }
    }

    //
    // Utility functions
    //
    /// <summary>
    /// Adds one element to the end of the array.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="array"></param>
    /// <param name="element"></param>
    public static void AddToArray<T>(ref T[] array, T element) {
        Array.Resize(ref array, array.Length + 1);
```

```
        array[array.Length - 1] = element;
    }

    /// <summary>
    /// Sets a pixel at a given location
    /// </summary>
    /// <param name="x"></param>
    /// <param name="y"></param>
    /// <param name="c">color of the pixel</param>
    private void SetPixel(Color[] tex, int x, int y, int width, int height, Color c) {
        if ((x < 0) || (x >= width) ||
            (y < 0) || (y >= height)) {
            return;
        }
        tex[x + y * width] = c;
    }

    /// <summary>
    /// Waits for time amound of seconds untill returning. This function is uneffected by
Time.timeScale
    /// </summary>
    /// <param name="time">Amount of time to wait before returning</param>
    /// <returns></returns>
    public IEnumerator WaitForRealSeconds(float time) {
        float start = Time.realtimeSinceStartup;
        while (Time.realtimeSinceStartup < start + time) {
            yield return null;
        }
    }

    /// <summary>
    /// Draws a triangle on a given texture based on the three verticies.
    /// The triangle is drawn and filled with the bresenham line algorithm.
    /// </summary>
    /// <param name="texture">The destination texture</param>
    /// <param name="v1"></param>
    /// <param name="v2"></param>
    /// <param name="v3"></param>
    /// <param name="value"></param>
    void DrawTriangle(Color[] texture, Vector2 v1, Vector2 v2, Vector2 v3, float compaction) {
        int y;
        for (y = 0; y < texture_height; y++) {
            ContourX[y,0] = int.MaxValue;
            ContourX[y,1] = int.MinValue;
        }

        ScanLine((int)v1.x, (int)v1.y, (int)v2.x, (int)v2.y);
        ScanLine((int)v2.x, (int)v2.y, (int)v3.x, (int)v3.y);
        ScanLine((int)v3.x, (int)v3.y, (int)v1.x, (int)v1.y);

        // Get pixels
        for (y = 0; y < texture_height; y++) {
            if (ContourX[y,1] >= ContourX[y,0]) {
                int x = ContourX[y, 0];
                int len = 1 + ContourX[y, 1] - ContourX[y, 0];
                while (len-- != 0) {
                    if (x < 0 || y < 0 || x >= texture_width || y >= texture_height) {
                        continue;
                    } else {
                        float compaction_value = compactValues[x + y * texture_width];
                        // Check if the asphalt has cooled enough
                        int heat_value = temperature_map[x + y * texture_width];
                        if ((1 - (heat_value / heat_data[0])) < 0.165f) {
                            compaction_value = 100;     // <- This needs to be too high
                        } else if (heat_value <= minimumCompactionTemperature) {
                            compaction_value = 0;     // <- No compaction below 50 degrees
celsius
                        } else {
                            compaction_value += compaction;
                        }
                        compactValues[x + y * texture_width] = compaction_value;
                        Color color = compactionGradient.Evaluate(compaction_value / 100.0f);
                        SetPixel(texture, x++, y, texture_width, texture_height, color);
                    }
```

```
                }
            }
        }
    }

    int[,] ContourX;
    /// <summary>
    /// Scans a line from line 1 to line 2
    /// </summary>
    /// <param name="x1">x coordinate of line 1</param>
    /// <param name="y1">y coordinate of line 1</param>
    /// <param name="x2">x coordinate of line 2</param>
    /// <param name="y2">y coordinate of line 2</param>
    void ScanLine (int x1, int y1, int x2, int y2) {
        int sx, sy, dx1, dy1, dx2, dy2, x, y, m, n, k, cnt;
        sx = x2 - x1;
        sy = y2 - y1;
        // Swap coordinates
        if (sy < 0 || sy == 0 && sx < 0) {
            k = x1; x1 = x2; x2 = k;
            k = y1; y1 = y2; y2 = k;
            sx = -sx;
            sy = -sy;
        }
        // Swap direction of walking
        if (sx > 0) { dx1 = 1; }
        else if (sx < 0) { dx1 = -1; }
        else dx1 = 0;
        // Y
        if (sy > 0) { dy1 = 1; }
        else if (sy < 0) { dy1 = -1; }
        else dy1 = 0;

        m = Mathf.Abs(sx);
        n = Mathf.Abs(sy);
        dx2 = dx1;
        dy2 = 0;

        if (m < n) {
            m = Mathf.Abs(sy);
            n = Mathf.Abs(sx);
            dx2 = 0;
            dy2 = dy1;
        }

        x = x1; y = y1;
        cnt = m + 1;
        k = n / 2;

        while (cnt-- != 0) {
            if ((y >= 0) && (y < texture_height)) {
                if (x < ContourX[y, 0]) ContourX[y, 0] = x;
                if (x > ContourX[y, 1]) ContourX[y, 1] = x;
            }
            k += n;
            if (k < m) {
                x += dx2;
                y += dy2;
            } else {
                k -= m;
                x += dx1;
                y += dy1;
            }
        }
    }
}
----------------------------------
TimeManager.cs
----------------------------------

using UnityEngine;
using UnityEngine.UI;
using System.Collections;
```

```
public class TimeManager : MonoBehaviour {

    public Text passedTimeText;
    public Text timeSpeedText;

    private Slider timeSlider;
    private float start_time;

                            // Use this for initialization
                            void Start () {
        start_time = ExecutionManager.simulationTimeSinceStart;
                                timeSlider = GetComponentInChildren<Slider>();
        timeSlider.onValueChanged.AddListener((speed)=> {
            timeSpeedText.text = speed.ToString("0.0") + "x";
            ExecutionManager.simulationTimeScale = speed;
        });
                            }

                            // Update is called once per frame
                            void Update () {
        if (!ExecutionManager.isPaused) {
            float timer = ExecutionManager.simulationTimeSinceStart - start_time;
            string minutes = Mathf.Floor(timer / 60).ToString("00");
            string seconds = Mathf.Floor(timer % 60).ToString("00");
            passedTimeText.text = minutes + ":" + seconds;
        }
                            }
}
-----------------------------------
UIWeatherManager.cs
-----------------------------------

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class UIWeatherManager : MonoBehaviour {

                            public int temperature;
                            public Image weatherImage;
                            public Text temperatureText;

                            /* 0 = sun
                             * 1 = clouds
                             * 2 = sun + clouds
                             * 3 = clouds + rain
                             * 4 = sun + clouds + rain
                             * 5 = clouds + snow
                             * 6 = freezing
                            */
                            public Sprite[] weatherIcons;

                            // Use this for initialization
                            void Start () {

                            }

                            // Update is called once per frame
                            void Update () {

                            }

                            public void SetWeather(WeatherCondition weather) {
                                if (weather.temperatuur < 0) {
                                        if (weather.sneeuw) {
                                                weatherImage.sprite = weatherIcons[5];
                                        } else {
                                                weatherImage.sprite = weatherIcons[6];
                                        }
                                } else {
                                        if (weather.zonnig) {
                                                if (weather.regen) {
```

```
                                                        weatherImage.sprite =
weatherIcons[4];
                                                } else if (weather.bewolkt) {
                                                        weatherImage.sprite =
weatherIcons[2];
                                                } else {
                                                        weatherImage.sprite =
weatherIcons[0];
                                                }
                                        } else if (weather.regen) {
                                                weatherImage.sprite = weatherIcons[3];
                                        } else {
                                                weatherImage.sprite = weatherIcons[1];
                                        }
                                }
                                temperatureText.text = Mathf.Round(weather.temperatuur) +
"°C";
                        }
}
```

UnitSelector.cs

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using System.Collections;
using System;

public class UnitSelector : MonoBehaviour {

    public static int LEFT_MOUSE_BTN  = 0;
    public static int RIGHT_MOUSE_BTN = 1;

    public FollowTarget follower;

                            public GameObject targetUnit;
    public GameObject selected_waypoint;

    [Header("UI")]
    public Canvas canvas;
    private RectTransform unit_menu;
    public GameObject paverMenu;
    public GameObject compactorMenu;

    [Header("Unit Selector")]
    public RectTransform unitSelectContainer;
    public Transform[] units;
    public GameObject unitSelectPrefab;
    public Button[] unitSelectButtons;
    public Sprite[] unitSprites;

                            // Use this for initialization
                            void Start () {

                            }

                            // Update is called once per frame
                            void Update () {
        if (selected_waypoint && !Input.GetMouseButton(0)) {
            selected_waypoint = null;
        }
        if (targetUnit && unit_menu) {
             unit_menu.anchoredPosition =
Camera.main.WorldToScreenPoint(targetUnit.transform.position);
        }
                            }

    // Update is called once per frame
                            void FixedUpdate () {
        // If we raycast a waypoint, don't set another one
        if (Input.GetMouseButtonDown(LEFT_MOUSE_BTN)) {
            RaycastHit waypoint_hit_info;
            Ray mouse_ray = Camera.main.ScreenPointToRay(Input.mousePosition);
```

```
            Debug.DrawRay(mouse_ray.origin, mouse_ray.direction);
            if (Physics.Raycast(mouse_ray, out waypoint_hit_info, 10000, 1 <<
Layers.WAYPOINT_INT)) {
                selected_waypoint = waypoint_hit_info.collider.transform.parent.gameObject;
            }
                                }

        // If we have selected a waypoint, move the existing one
        if (selected_waypoint && Input.GetMouseButton(LEFT_MOUSE_BTN)) {
            RaycastHit hitInfo;
            Ray mouse_ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            Debug.DrawRay(mouse_ray.origin, mouse_ray.direction);
            if (Physics.Raycast(mouse_ray, out hitInfo, 10000, 1 << Layers.FLOOR_INT))
            {
                selected_waypoint.transform.position = hitInfo.point;
            }
        }

        // Select units
        if (Input.GetMouseButtonDown(LEFT_MOUSE_BTN)) {
            RaycastHit hitInfo;
            Ray mouse_ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            Debug.DrawRay(mouse_ray.origin, mouse_ray.direction);
            if (Physics.Raycast(mouse_ray, out hitInfo, 10000, 1 << Layers.UNIT_INT))
            {
                SetSelectedUnit(hitInfo.collider.gameObject);
            }
        }
    }

    public void GenerateUnitSelectionMenu(GameObject[] pavers, GameObject[] compactors,
GameObject[] shakers, GameObject[] trucks) {
        // Generate the unit selection menu
        foreach (GameObject paver in pavers) {
            GameObject go = Instantiate(unitSelectPrefab);
            go.transform.SetParent(unitSelectContainer, false);
            go.GetComponent<Button>().onClick.AddListener(()=>{
                SetSelectedUnit(paver);
            });
            go.transform.Find("Image").GetComponent<Image>().sprite = unitSprites[0];
        }
        foreach (GameObject compactor in compactors) {
            GameObject go = Instantiate(unitSelectPrefab);
            go.transform.SetParent(unitSelectContainer, false);
            go.GetComponent<Button>().onClick.AddListener(()=>{
                SetSelectedUnit(compactor);
            });
            go.transform.Find("Image").GetComponent<Image>().sprite = unitSprites[1];
        }
        foreach (GameObject shaker in shakers) {
            GameObject go = Instantiate(unitSelectPrefab);
            go.transform.SetParent(unitSelectContainer, false);
            go.GetComponent<Button>().onClick.AddListener(()=>{
                SetSelectedUnit(shaker);
            });
            go.transform.Find("Image").GetComponent<Image>().sprite = unitSprites[2];
        }
        foreach (GameObject truck in trucks) {
            GameObject go = Instantiate(unitSelectPrefab);
            go.transform.SetParent(unitSelectContainer, false);
            go.GetComponent<Button>().onClick.AddListener(()=>{
                SetSelectedUnit(truck);
            });
            go.transform.Find("Image").GetComponent<Image>().sprite = unitSprites[3];
        }
    }

    public void SetSelectedUnit(GameObject unit) {
        follower.SetTarget(unit.transform);
        if (targetUnit)
        {
            SetOutline(targetUnit, false);
        }
        targetUnit = unit;
```

```
        SetOutline(targetUnit, true);
        if (unit_menu) Destroy(unit_menu.gameObject); // Old menu if it existed
        switch (unit.tag)
        {
            case Tags.COMPACTOR: {
                unit_menu = Instantiate(compactorMenu).GetComponent<RectTransform>();
                unit_menu.SetParent(canvas.transform);
                Vector3 pos = Camera.main.WorldToScreenPoint(targetUnit.transform.position);
                unit_menu.GetComponent<RectTransform>().anchoredPosition = pos;
                unit_menu.GetComponent<CompactorPropertyManager>().compactor =
                    targetUnit.GetComponent<RoadCompactor>();
            } break;
            case Tags.PAVER: {
                unit_menu = Instantiate(paverMenu).GetComponent<RectTransform>();
                unit_menu.SetParent(canvas.transform);
                Vector3 pos = Camera.main.WorldToScreenPoint(targetUnit.transform.position);
                unit_menu.GetComponent<RectTransform>().anchoredPosition = pos;
                unit_menu.GetComponent<PavingPropertyManager>().paver =
                    targetUnit.GetComponent<PaverController>();
            } break;
        }
    }

    /// <summary>
    /// Removes previous outline on object and adds to the provided unit.
    /// </summary>
    /// <param name="unit">The unit to outline</param>
    void SetOutline(GameObject unit, bool value) {
        RoadCompactor compactor = unit.GetComponent<RoadCompactor>();
        if (compactor) {
            compactor.SetActivateWaypoints(value);
        }

        Renderer[] renderers = unit.GetComponentsInChildren<Renderer>();
        if (renderers.Length < 1) {
            renderers = unit.GetComponents<Renderer>();
        }
        for (int renderer_index = 0; renderer_index < renderers.Length; renderer_index++) {
            GameObject go = renderers[renderer_index].gameObject;
            if (go.name.Contains("Shadow")) {
                continue;
            }
            if (value) {
                go.layer = Layers.OUTLINE_INT;
            } else {
                go.layer = Layers.DEFAULT_INT;
            }
        }
    }
}
```