Technical University of Denmark

*Matthias Sebastian Alan Larsen, s103437*
*Joachim Vestmark Vith Jensen, s103430*

# Engineering of an Interactive Game for Teaching Elementary Mathematics

Bachelor's Thesis
B.Sc.-2013-27

**DTU Compute**
Department of Applied Mathematics and Computer Science

*Matthias Sebastian Alan Larsen, s103437*
*Joachim Vestmark Vith Jensen, s103430*

# Engineering of an Interactive Game for Teaching Elementary Mathematics

Bachelor's Thesis

Engineering of an Interactive Game for Teaching Elementary Mathematics,

**This report was prepared by**
Matthias Sebastian Alan Larsen, s103437
Joachim Vestmark Vith Jensen, s103430

**Supervisors**
Jeppe Revall Frisvad
Niels Jørgen Christensen

---

| | |
|---|---|
| Release date: | June, 2013 |
| Category: | 1 (public) |
| Edition: | First |
| Comments: | This report is part of the requirements to achieve the Bachelor of Science in Engineering (BSc) at the Technical University of Denmark. This report represents 20 ECTS points. |


Department of Applied Mathematics and Computer Science
Technical University of Denmark
Matematiktorvet
Building 303 B
DK-2800 Kgs. Lyngby
Denmark

www.compute.dtu.dk
Tel: (+45) 45 25 30 31
Fax: (+45) 45 88 26 73
E-mail: compute@compute.dtu.dk

# Abstract

When developing an interactive computer game for teaching elementary mathematics, it should be ensured that the game in fact can be used as teaching material and it should be sought that it also is motivational. The approach in this thesis has been to develop a game of high availability, with mathematical problems being presented differently and with levels of difficulty that both challenge and are not too hard for the players. The solution has been to engineer a browser based game that teaches multiplication and addition. In a city seen from above, a garbage truck with a given capacity should collect trash of given sizes and in given representations. The player needs to fill the truck completely by clicking on houses with trash, i.e. match the truck's capacity with the trash sizes. The level of difficulty adapts to the player's mathematical skills based on his or her performance in the game.

# Preface

As this project has been carried out by a group, we have been working together as a team with constructive criticism of each other, exchanging ideas, discussing designs and algorithms, correcting syntax errors etc. This means that everyone has more or less had a hand in every part of the game and thesis. However:

Matthias Larsen has had the main responsibility of the following parts of the game and thesis:

- Server-side

Joachim Jensen has had the main responsibility of the following parts of the game and thesis:

- Client-side

This thesis has a technical focus and only covers the psychology of teaching and learning superficially and briefly. The game concept presented and designed has been discussed with and approved by Erik Ottar Jensen, Teacher and Mathematics Teacher Advisor at Skolen ved Bülowsvej, Frederiksberg, Denmark

The target audience for this project is pupils between the age of 7 and 10, because this is the group that, in Danish schools, is taught the elementary mathematics this project is about. The game engineered in this project should be considered a prototype or proof of concept in the use of digital solutions for teaching.

It is a requirement that the game provides tracking of the player's performance so that the skill level of each player can be analysed and evaluated.

# Acknowledgements

We would like to express our gratitude to our supervisors Jeppe Revall Frisvad and Niels Jørgen Christensen for giving us the opportunity to work on this bachelor's thesis and for their feedback and guidance through the learning process. Furthermore we would like to thank Michael Rose for his introduction to game design.

Especially we would like to thank Erik Ottar Jensen for his invaluable comments and engagement during the development of the game concept and giving us the opportunity to test the game at Skolen ved Bülowsvej, Frederiksberg, Denmark. The participants of the test have provided us with important data about their user and learning experience, for which we are truly grateful.

Finally we would like to thank Tobias Løvgren Madsen for taking time to proofread the thesis.

# Contents

# List of Figures

# List of Tables

# Listings

# Nomenclature

**City** A road network with adjacent houses,

**Level** A world with content defined by the difficulty for a player.

**Modern browsers** Internet Explorer 9+, Google Chrome 15+, Mozilla Firefox 20+, Opera 12+ and Apple Safari 7+.

**Node.js server** The running Node.js instance.

**Player** See 'user'.

**Pupil** See 'user'.

**The application** The combined system of the PHP and Node.js implementation.

**User** A person playing the game or using the application.

**Viewport** The visible area of some content.

**World** A city generated by PHP.

# 1 Introduction

Teaching can be practised with a lot of different methods such as demonstration, recitation or memorization. They are all about passing on and applying patterns and models to the subjects of the teaching. One way of teaching is by making the pupil learn by playing computer games. The problem is that such computer game would have to teach correctly, and when dealing with mathematics, it should be ensured that the right patterns and models are the ones passed on, and that the pupils should be able to identify the mathematical problems in the game. At the same time, the game should be motivating.

This is an important topic because mathematics is widely used and helping teaching the pupils to solve different mathematical problems at a young age would improve their skills in innovation and problem identification, which are very important in fields like engineering. It is interesting because it will combine educational material with digital sources.

Each child is unique in their way of thinking and doing, which means that different teaching methods will work differently for each child. Some children are fine with traditional methods of teaching, while these methods have less impact on others. The game should be able to reach out to and teach as many pupils as possible, but assuming that every child will like the same game or teaching method is naive.

Combining play with teaching is not a new thing, but because computers are relatively new and because of the low availability of computers in many Danish schools, using computer games for education is not widely spread. Many existing digital solutions assume that every child learns the same way and use an instruction-like teaching method with little or no interactivity from the player.

This thesis describes the processes of engineering of a game which teaches multiplication and addition as well as the results. The key components of the game are that it is interactive and motivational, that it is able to present a mathematical problem in different ways, that instruction-like methods are avoided, that the availability is as big as possible, and finally that every child in the target audience should be able to play it regardless of his or her mathematical skills.

# 2 Analysis

This chapter describes the considerations and reflections in the earliest stages of the development process of the game. A game concept is presented and feature ideas are turned into possible algorithms, while technical requirements and solutions are evaluated.

## 2.1 Mathematical skill

Multiplication and addition are both important mathematical skills that are taught to pupils at a very young age in Denmark. They are important because they form one half of elementary arithmetic with negation and division forming the other half. While the study of mathematics covers many branches, elementary arithmetic is used often and is even required in order to learn and understand other areas of mathematics, such as algebra. This means that if one does not learn multiplication and addition, it will be very hard to learn other areas of mathematics, and for a pupil it means that as time goes by he or she might be left behind in the educational schedule and need special training.

When teaching any kind of mathematics, the teacher - being a computer or a human being - need not only to teach the algorithms on how to deal with the problems but also, with more emphasis, the theories describing the patterns and logical reasoning behind them.

A classic method is repetition, but it can be dangerous if a pupil successfully applies his own patterns that does not comply with the correct patterns and rules but work for some specific problems. This means that he has not fully understood the mathematics. Different patterns are great if they comply with the correct patterns because it shows individuality, but the teacher must be careful that the pupil does not get tunnel vision.

## 2.2 Gamification

Using digital solutions for teaching is a popular topic which is also being discussed among politicians. Recently, the current Danish government has released a proposal for a reform of the public school, which states:

> For at fremme anvendelsen af it i folkeskolen har regeringen afsat en pulje på i alt 500 mio. kr. i perioden 2012-2015. Puljen skal især bidrage til at øge anvendelsen af it og sikre bedre brug af digitale læremidler i folkeskolen. [25].

This is loosely translated to:

> In order to promote the use of digital solutions in schools, the government allocates a total of DKK 500 million in the period 2012-2015. The fund will mainly help to increase the use of digital solutions and ensure better use of digital learning materials in public schools.

This emphasizes that the current Danish government wants digital solutions to be more present and used in the teaching in Danish public schools.

Teaching multiplication and addition through a game is a tough job. It must be guaranteed that the mathematics comply with the correct patterns and that the game actually is educational and able to teach as intended. At the same time, because it is a game, it needs to be fun and motivational which is even harder to achieve because of the subjectivity of "fun".

In order to avoid tunnel visioning of the player and help him or her identifying the actual problems and applying the correct patterns, the problems met in the game must be represented differently in a way that is not very predictable. To make the player progress and further develop his mathematical skill in multiplication and addition, there have to be an increasing level of difficulty.

All this raises some questions: In what ways can multiplication and addition problems be represented? How can difficulty be quantified? And most of all, how can this be translated into a game?

### 2.2.1 Game concept proposal

The initial concept is a car with a specific capacity in its trunk whose purpose is to collect or deliver some sort of packages of a given size and quantity. It can be a postal truck or a garbage truck that drives around a city. The player then need to make sure that the numbers fit together when collecting or delivering packages, i.e. solve multiplication and addition problems correctly. The car and city are viewed from above.

Two questions remain and now even more have arisen: How should the player find the path to the destinations? How should the content be generated while ensuring that everything actually match up? And how should the player be rewarded when solving the problems correctly?

### 2.2.2 Path finding

The most obvious solution would be to let the player drive the car with the keyboard. This is very interactive and liberating because it gives the player complete

control over the travel from point A to B. However, it can easily shift focus from the actual educational purpose of the game, both because the player would spend time on driving around, but also because it can be a distraction to calculate the shortest paths when the player has to keep track of how to match the car capacity with packages optimally. Requiring too much interactivity from the player can give a confusing user experience and lead to worse learning [17].

One of the steps needed to increase the user experience is if such actions are automated. Letting the game control path finding means that the player himself does not have to bother finding the shortest path from point A to B. Automated path finding can be achieved in many ways, but one of the simpler ways is using graph theory. If the city is looked at as a graph, each house and road cell represent nodes.

There are several algorithms for path finding using graphs, each with advantages and disadvantages depending on the needs.

#### 2.2.2.1 Breadth-first search

Using breadth-first search (BFS) to search for a node in a graph means that, when beginning at the root node, all neighbouring nodes will be inspected. For each inspected node, their neighbouring nodes will be inspected afterwards in turn, and so forth. The algorithm uses a queue data structure to keep track of which nodes have been visited and which have not. When the node that is searched for is found, the operation is successful and inspection stops. Likewise, if all nodes have been inspected and the operation has not been stopped, it means that the node searched for does not exist in the graph.

The method can be used to find the shortest path between two nodes in an unweighed graph.

#### 2.2.2.2 Depth-first search

Using depth-first search (DFS) to search for a node in a graph means that, when beginning at the root node, the first neighbour node will be inspected. Then this node's first neighbour will be inspected, and so forth. When the node that is searched for is found, the operation is successful and inspection stops. If the algorithm reaches a node that does not have any successors, it will backtrack to the first node that have uninspected neighbours and continue the exploration. If all nodes have been inspected and the operation has not been stopped, it means that the node searched for does not exist in the graph.

DFS does not necessarily find the shortest path between two nodes, but it finds a path, if one exists.

#### 2.2.2.3 Best-first search

Best-first search is a strategy used to search in a graph with a given heuristic that depend on the graph and goal. By using a heuristic made for a specific purpose, the algorithm can on the way to the goal evaluate the onward search path with

information gathered up to the currently inspected node. This way, the path will contain nodes that are determined to be optimal in order to reach the goal.

The method can be used to find the shortest path between two nodes in a weighed graph.

#### 2.2.2.4 Method Comparison

While it is not crucial to find the shortest path from a car's current position to a house, it should be ensured that an actual path is found. To make the car movement more realistic, it should not be possible to drive through a house, even if this would be the shortest path to a destination. However, it should of course be possible to visit a house, i.e. inspect a house node. Both BFS and DFS algorithms fall short on this requirement, while a best-first algorithm such as A* can fulfil it.

A* finds a shortest, least-cost path between two nodes in a weighed graph. Because it evaluates the weighs during the search, it can find the path with the lowest expected total cost. Along the way, it keeps a sorted priority queue of alternative paths to take. This way, the cost of visiting a house can be set to be very high, making it inefficient to drive through it compared to taking an alternative path around it. Therefore, it can be ensured that a house should only be visited if it is the goal. Moreover, it can support different types of roads, e.g. highways and dirt tracks, where the former should be preferred over the latter.

### 2.2.3 Generators

The city would consist of a road network and an arbitrary number of houses. Houses should be able to contain packages of a given size in a given representation, and an arbitrary number of cars with a given capacity and representation should be able to either collect or deliver the mentioned packages to the houses. All these arbitrary numbers need to match up in the end.

#### 2.2.3.1 Roads

The road network would need to have at least one start cell and one exit cell for the cars to enter and leave the game when appropriate. Other than that the road network does not depend on the other content in the game. It can be generated completely at random, with a seed or a predefined template that matches the difficulty.

A completely random road network will make the game less repetitive if playing the same level of difficulty more than once. A seed would make it possible to get the same road network again. While a randomly generated road network and a road network from a template both can be said to be finite, the latter is the most because it requires a predefined, finite set of combinations that is most likely smaller than the set a random generator could create.

The system should guarantee that all the roads in the network are connected, and that a start cell and an exit cell can be reached.

One method to create the road network is to start by initializing a world of a given size X by Y and setting all cells to grass. A start and exit cell should then be selected, either randomly or statically. An A* algorithm could then be used to create a road network that connects the start and exit cells. If a more maze-like network is wanted, a DFS algorithm could be used.

#### 2.2.3.2 Houses, Packages and Cars

Houses, packages and cars cannot, like the road network, be generated completely at random because they depend on each other. There cannot be more packages than houses, because where should they then be put? In order to win the game, there should be enough packages for each car. Also, cars should be able to reach houses, therefore the houses have to be placed only adjacent to roads.

The following statements must be ensured when generating content:

- There has to be at least one car
- The number of houses must be at least the number of cars
- The number of packages must be at least the number of houses
- The accumulated size of packages must at least correspond to the accumulated car capacities
- The car capacity must have a perfect fit with one or more packages

The first statement might seem obvious, but that does not make it less important. If an algorithm does not have this base rule, all the other statements become flawed because the constraints are then inadequate to ensure that the game logic is as intended, and thus the game will be left with errors. The list guarantees that no matter how many cars, houses and packages that are generated, and no matter how big the various car capacities and the packages' sizes are, it is possible to win the game, because the content match up.

There are two ways to follow the list, either top-down or bottom-up. Top-down means that initially some content is generated and thereafter adjusted to fit, and bottom-up means that content is generated for an accurate fit. It is much like a box, where a top-down method tries to stuff some content in it to fit the size, and a bottom-up method will build the box in the size needed. While the latter might be the easiest to configure to actually follow the list, it often comes to down to performance when choosing the algorithm.

A top-down method is to create houses and packages first, and then use a network flow graph to do the initial car generation.

A number range for the current level starts from n and ends in m. The flow graph's sink should at maximum accept m, forcing all combinations to stay within the range of the game.

An example of a graph with the packages of sizes: $[5, 6, 9, 10]$, given $n = 4$, $m = 11$ can be seen on figure 2.1

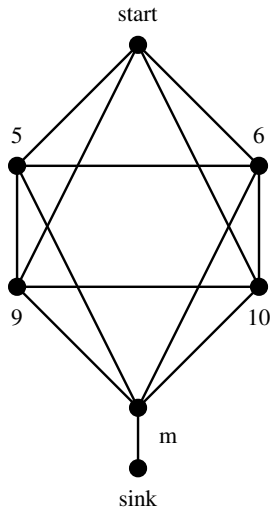



**Figure 2.1:** Initial graph

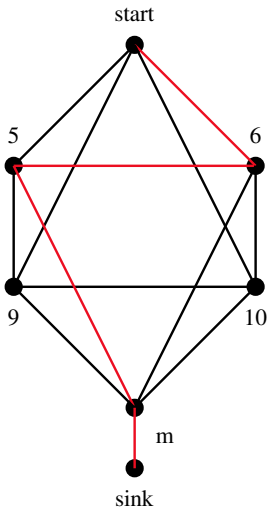



**Figure 2.2:** The path

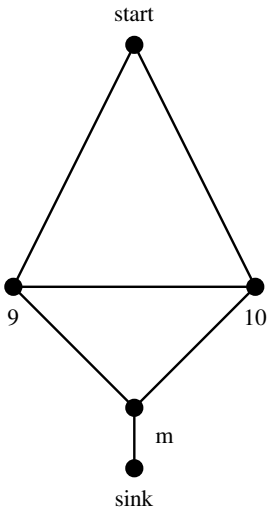



**Figure 2.3:** Removed path

As one can see, all package sizes are represented by a node, and visiting a node costs the node's value. When a path is found between start and sink, for instance like the one shown in figure 2.2, these nodes would be 'used' and the nodes and their corresponding edges would be removed from the graph as shown in figure 2.3.

Given $m = 11$, this will eventually result in the following set for the cars: $[[5, 6], 9, 10]$, thus resulting in three cars, of which two can contain one package and one car can contain two packages. If the number of sets would come too close to the number of packages, i.e. each car would only have one package, the algorithm should merge some of the cars resulting in fewer cars that could contain multiple packages.

The merging would be done in the following manner: Take the smallest and largest car in the set, merge the two, and remove them from the set. Keep doing this until there is either no more cars left in the set, or the number of cars is satisfying.

A bottom-up method would be to determine the number of cars initially and then calculate their capacities. For each car capacity, some trash should be generated in different package sizes, and finally houses should be generated for each package.

As seen, the first method is much more complex than the second method and in a worst case scenario, all cars could be removed if no safety was created. On the other hand, the second method might be slower performance wise.

Instead of inserting all packages in the game from the start, they could pop up over a time period – and perhaps disappear again. This could increase the number of packages for each game as they could then spawn in the same house. It rises a problems for the player though: how can he be certain that the present packages are the right packages for the car? Should the player really wait for the right packages to spawn?

### 2.2.4 Representations

Each car and package should have a representation of their value. This representation could be a table, figures, a simple equation or just the actual number.

It is important that it varies, so that the pupil will be taught that values are different from and do not rely on numbers, and that a number in fact just is a representation of a value. Other representations of the same value are equally correct, and by displaying different representations, the game will force the pupil to apply mathematics to them in order to solve the problem.

### 2.2.5 Difficulty

Quantifying difficulty in a game can be done by introducing levels, where a higher level corresponds to an increased difficulty. A predefined set of levels, referred to as static levels, is most common, but it has only one learning curve implying that each pupil should learn equally much. This curve might be steep for some, while others are not challenged at all. Therefore the difficulty could also be based on each pupil's skill and adapt to it.

But how can the game introduce difficulty? What present parameters can be tweaked?

#### 2.2.5.1 Parameters

The difficulty could be quantified with the following variable parameters:

- Number of houses
- Number of packages
- Number of cars
- Package size and spread
- Car capacity
- Package and car representation

A lower or higher number and different representations would thus give a lower or higher level of difficulty. For static levels, the parameters would get exact values for each level, but with dynamic levels it would be more tricky. On what basis should the parameters be tweaked?

#### 2.2.5.2 Static levels

Developing static hard-coded levels would be the easiest option, and they would go well together with world templates generated in advance. As mentioned earlier, this would defeat the purpose of assisting children in developing their skills individually

because of the uniformity of progression steps. Static levels also indicate that the game, and thus the mathematical learning, has an ending, because they are finite.

#### 2.2.5.3 Dynamic levels

Instead of having hard-coded levels that are the same for everyone, the difficulty of the game could be based on the player's skills.

An intelligent engine that can quantify the difficulty and provide variables for a level by looking at how the player has performed, e.g during the latest X games, would have to be created. This way the system would attempt to challenge the players individually and work much like a personal trainer by adapting the difficulty to their skills.

In a press release from The Ministry of Children and Education in Denmark, concerning the use of digital solutions in education, it is stated that:

> The purpose of digital solutions is to improve education for all pupils regardless of their level of skills and competences, and work as a method of including and maintaining as many pupils as possible in everyday education. This can be achieved by using for instance elements from computer games as digital teaching tools, to motivate the pupils while adapting the level of difficulty after individual level of skill and need [24].

Which is exactly what dynamic levels would add to the game. Such an intelligent engine does, however, raise some questions:

- Should it be possible to repeat a level?
- Should it be possible to choose between levels?
- How should the teacher see which level the player are at?
- Should the levels be generated on-the-fly?
- Should there be some static levels and content that act as a base for the levels?
- Should a teacher be able to modify these parameters for a class?

Also, while the parameters to tweak difficulty have already been outlined, how should the system tweak them? Some data would need to be logged such that the parameter values can be calculated. The logged data could be used to find out how the player performed in the X latest game or X best games. In order for the system to get some initial data about the player's skills, the first few games should be used for calibration.

The parameters could be set by calculating on the following data from previously played levels:

- Number of total perfect fills
- Highest number of consecutive perfect fills (combos)
- Number of broken combos
- Number of cars
- Number of packages
- The range of numbers of which is used on a given level (number range)
- Representations
- Correctness of each representation
- Time consumption

If the levels are dynamic, limiting the world size would indirectly set an upper limit on the highest possible difficulty. Options as to how the world should be sized is something that should be explored and considered if working with dynamic levels.

One of the things that could be done to scale the world is changing the cell sizes. Decreasing the cell size would allow more cells to fit into the same space, thus creating a bigger world. This could become a problem when the world gets too big or if some of the players do not have a great sight.

Another way to counter the problem is drag-based expansion as seen in games such as Civilization[34], Sim City[15] and Age of Empires[33]. They keep the cell size and viewport constant but has a bigger world than the viewport. This allows the player to drag other areas of the world into the viewport. However, this would hide some content on the screen and might confuse the player as to where to find the right packages.

### 2.2.6 Motivation

> When I watch children playing video games at home or in the arcades, I am impressed with the energy and enthusiasm they devote to the task. ... Why can't we get the same devotion to school lessons as people naturally apply to the things that interest them? [23]

The most challenging part of creating a game is to make it entertaining enough to keep the users playing it. Having a single quest line without any branches or side quests, thus a very simple story line in the game, will often not be played for very long because the player is not challenged enough due to either the simplicity of the quest and story line or because the user is not free enough to make his own choices hence not giving the player a feeling of controlling the game.

More and more games have implemented side quests, providing a more free world allowing the user to move more freely around than previously and not chained as much to a single quest line.

> When people are intrinsically motivated to learn, they not only learn more, they also have a more positive experience [6].

A few techniques exist today that can help increase the longevity of a game and motivate users to play it, thus increase the learning experience. Some of these techniques are discussed in the following sections.

#### 2.2.6.1 Points

An essential part of a video game is the points and some sort of highscore or leaderboard. Humans are natural competitors and most love to win. While this should be an educational game, giving points for completed games can be a very difficult task. It is important that every pupil is learning something without being discouraged by a poor score because he or she is having difficulties with the mathematics. If a class were to have an autistic pupil, he or she could be teased by his enormous score if he was the only one in the class in his bracket, say 100% above second place. Likewise, pupils that perform worse than the majority could be easy targets for teasing.

Giving scores for each completed game while encouraging further play and limiting the chance of potential teasing in a class can prove to be a difficult task and something that should be handled very delicately.

One way to solve this could be to calculate the score for a game based on the pupil's prior games and skills and giving something that indicates the pupil's level compared to when he or she started. Another way could be to grade the game based on a relative score for the best solution for a given level.

Points are also a great way to keep the users entertained and one technique that does that very good is *streaks* or *combos* like the ones seen in games like Guitar Heros[14]. Building up a combo or streak encourages the player to pay more attention to the game to avoid losing the extra bonus points such a streak would give.

#### 2.2.6.2 Achievements

As already mentioned, keeping the players entertained is key to a popular game which in this case should increase the learning. Another way to increase the *addiction* to a game is through achievements.

Achievements can be disguised as many things. The most common forms are trophies, badges, awards, stamps and medals and can be used to unlock special content. They do not have a direct impact on the main goal of the game, and the main goal does not depend on achievements.

They are used to increase the longevity of a game and to provide players with an option to explore more of game than its main story line. Unlike secrets that traditionally provide the player with some sort of benefit, e.g. by making the game easier, achievements normally provide no such thing.

Adding achievements to the game would allow players to not only compete on a

score, which explained in section 2.2.6.1, can pose as a bad idea, but also compete on who completed the most achievements or even a specific achievement.

Appropriate achievements could be *perfectly fill X cars*, *remove X packages from the city*, *Overfill X cars*, *Complete X cities* all of which could motivate the players to compete, not on an intellectual level but instead on game completions which in turn would increase the chances of the players increasing their own knowledge individually.

## 2.3 Technical requirements

When implementing the game, it will have some technical requirements. How should the players, their scores and relevant data in general be kept track of? On what platform should the game be deployed? How can schools easily make the game available to their pupils?

Availability is the key concern when answering these questions. The game should be able to reach out to as many potential players as possible. Today, schools often provide computers and internet access to their pupils as digital teaching material already is becoming more and more popular.

Mostly, pupils do not have elevated privileges on the provided computers, which is great security wise, but may be a problem for the game, because it means that they are not allowed to install it locally. It would again mean that the game would have to be white-listed by each school, which can be a long process and limit the availability greatly. Hosting the game on a centralized platform instead would overcome these problems, but it would of course still be up to each school if they want to make use of it.

Because internet access is already provided by many schools, it would be an obvious choice to make the game browser based. The next sections will in detail outline possible software that can be used to meet the technical requirements.

### 2.3.1 Client-side

The client-side of the game describes the operations that are performed on the client, i.e. the player's computer. It includes the front end which provides a graphical user interface that the player can interact with.

#### 2.3.1.1 Graphical User Interface

The graphical user interface should be easy for the player to understand. Relevant information should be displayed properly in a way that does not lead to confusion. For the best user experience, the content should be presented as intended, i.e. houses should be shown as houses and it should be easy to identify interactive content.

#### 2.3.1.2 Game engine

An existing game engine can be used to shorten the development time of the game as it probably will take longer to build an engine specific for the game. Before comparing potential game engines, programming languages and software platforms must be assessed to reduce the huge list of game engines available. Table 2.1 shows a comparison of platforms and languages for game engines.

**Table 2.1:** Comparison of platforms and languages for game engines

| Software/Language | Level of knowledge | Support |
|---|---|---|
| Adobe Flash | Medium | 3rd party plugin |
| JavaScript | High | In browser, enabled by default |
| Microsoft Silverlight | Low | 3rd party plugin |
| WebGL | Low | In browser, but often disabled |

Microsoft Silverlight and Adobe Flash are both big players in the field of embedding video, audio and games in websites, but since the release of HTML5[1] that introduced the ability to watch video and listen to audio (although still in very limited formats) on a website without any plugin, both are becoming less popular. The official Adobe Flash plugin has often been criticized because of severe security flaws, and while the desktop version is still being developed, Adobe focuses on HTML5 in the mobile version and has released a 'Flash to HTML5' conversion tool[36][19]. A new major version of Silverlight is not on the current official roadmap of Microsoft and rumours are that the current version is the last[9]. Moreover, Microsoft states that its support life cycle for the current version of Silverlight will end no later than 2021[21].

Most modern browsers come with a bundled JavaScript engine to interpret and execute JavaScript in websites. The engines perform and execute differently, which can change the behaviour of a script across browsers. Therefore, even though it is widely supported, the script should be tested in the given browsers to verify that it is interoperable. WebGL is a JavaScript API[2] for rendering GPU[3] accelerated 2D and 3D graphics in the HTML5 canvas element, but even though it is cross-platform, it is only supported by modern browsers and often only experimental or partial. Internet Explorer only supports WebGL from version 11 and onwards.

While Adobe Flash and Microsoft Silverlight are still superior in rendering 3D graphics compared to JavaScript, their already mentioned disadvantages combined with the fact that they require a third party plugin has resulted in JavaScript being chosen. Past experience with JavaScript and because of its support makes it the best candidate.

There are many free JavaScript game engines and table 2.2 only names a few that are popular and seem appropriate for this project with their relevant features.

* Unity3D[35] Free uses its own variety of JavaScript and requires a plugin to be executed in the browser.

[1] HyperText Markup Language
[2] Application Programming Interface
[3] Graphics Processing Unit

**Table 2.2:** Comparison of JavaScript Game Engines

| | Crafty v0.5.3 | EnchantJS v0.6 | Sprite.js v1.2.1 | Unity3D v4* |
|---|---|---|---|---|
| Browsers | All modern | All modern | All modern | Plugin |
| Size | 107 kb | 70 kb | 49 kb | N/A |
| Collisions | Yes | | Yes | Yes |
| Events | Yes | Yes | | Yes |
| Entities | Yes | Yes | | |
| Audio | Yes | Yes | | Yes |
| Path finding | Yes | | Yes | |
| Plugins | Yes | Yes | Yes | Yes |
| Physics | | Yes | Yes | Yes |
| WebGL | Yes | Yes | Yes | |
| Time tracking | | | Yes | |
| Sprite map | Yes | | Yes | |
| Touch screen | | | Yes | |
| 3D | | | | Yes |
| AI | | | | Yes |
| Lighting | | | | Yes |

Note that because the engines support plugins, the features in the table might in fact be supported through one of these. The feature support are taken from the documentation of each of the engines, and both EnchantJS[8] and Sprite.js[5] have moved several of the claimed features to plugins, which can explain their small core sizes - those features are still stated in the list.

When choosing JavaScript, HTML5 is an obvious choice for controlling the layout. HTML5 introduced a canvas element that allows rendering of shapes and bitmap images by scripting. The canvas element is raster-based and can be used to dynamically generate the graphics in the game. Alternatively, a DOM[4] tree, that is traditionally used for layouts in HTML documents - and which the canvas element is part of too - can also be used by being manipulated by JavaScript. The DOM tree consists of all the HTML nodes in the document, and creating HTML5 elements instead of shapes in a canvas element make it possible to also use CSS[5] for the graphics. CSS3 allows for transitions and animations on HTML elements. A hybrid of both HTML elements and the canvas element could also be used.

Of the game engines, Unity3D was screened out as the first because of the need for a third party plugin. Crafty[7] seemed most stable, straightforward and the best candidate. Because the game should be viewed from above and uses cells, it would be an obvious choice to make the game tile-based, which enables the ability to create different tiles for each type of content. The entity support in Crafty will make it possible to easily implement a component for such tiles.

[4]Document Object Model
[5]Cascading Style Sheets

#### 2.3.1.3 Logging

Logging of data can serve multiple purposes. From a developer's point of view, logs can tell if errors are happening and if the same ones keep triggering. Analysing the logs can help determine if the application contains a bug that triggers a certain error, or if the system is going to collapse.

A game designer would be able to see if a particular part of a game was used more than others. This could help discover trends or popular features which could be used in the next production of a game or to implement or remove features from a current game by comparing analytic log data from games. A teacher would, in the case of this type of application, be able to see how each pupil progresses. Watch the performance of the pupils, their scores, what they do wrong, which parts of the mathematics they have a hard time with and which parts they find easy.

When it comes to logging events that happen on the client-side it is necessary to send it to the server for processing. The transmission of log data should be done asynchronously to avoid affecting the game play. Doing it asynchronously on-the-fly and not when a game is finished would allow for real time logs for the involved parties and also to ensure that everything is logged while the game is played, which would not be the case if logs were submitted after a game, and the client loses the connection to the server in the middle of the game. Two solutions for asynchronous communication with a server exist, namely AJAX and WebSocket.

**AJAX** is an acronym for Asynchronous JavaScript and XML. It is used to exchange data between a browser and a server without having to reload a page in the browser. Because this allows the website content to be manipulated dynamically, it is widely used to create web applications featuring such functionality. A request does not necessarily need to be asynchronous, and the data exchange does not need to use XML.

**WebSocket** is a technology that provides a bidirectional communication channel between a browser and a server and is full-duplex, meaning that data exchange can happen simultaneously both ways. This makes it suitable to enable live content on a website.

The challenge with cross origin communication is something that could be problematic when AJAX is used, but not with WebSocket. One other important advantage of WebSocket when comparing to AJAX is the low latency. Lower latency in logging and other messages between client and server will result in a more smooth experience when playing the game. Each AJAX call would have to both send and receive HTTP[6] headers for each request, resulting in a lot of unnecessary bandwidth used which is not the case for WebSockets that only suffer on the initial hand shake, due to the connection being persistent. The WebSocket protocol uses an event driven model, whereas AJAX uses polling[7] or long polling. However, WebSockets are relatively new, only supported by modern browsers and requires the server to explicitly

[6] Hypertext Transfer Protocol
[7] A technique to emulate bidirectional communication between client and server

support it too, whereas AJAX is more used and also more supported by browsers[8]. Because the game engine already requires a modern browser, WebSockets are clearly the best candidate for logging.

### 2.3.2 Server-side

The server-side of the game describes the operations that are performed on the server. This project will need a game server and a database server. The game server should provide services in the game and in the web application containing the game.

#### 2.3.2.1 Game server

The game server is in charge of providing resources and handling requests for the web application. This could be user authentication and user management as well as supplying dynamic content for the website. Moreover, it should work as the back end for the selected game engine by giving necessary content to the client and process and verify content from the client.

**Table 2.3:** Comparison of languages for game server

| Language | Level of knowledge | Platform |
|---|---|---|
| ASP.NET | Medium | Windows |
| PHP | High | Windows, Linux, OSX |
| Node.js | Low | Windows, Linux, OSX |

Below is an assessment of the languages mentioned in table 2.3.

PHP and ASP.NET are both well-documented and widely used as back end for dynamic websites. PHP is installed on more than 2.1 million web servers and used by over 244 million websites[13], making it the most popular language used on websites and web applications.

Node.js is a platform built on Chrome's JavaScript runtime; it is basically server-side JavaScript, and therefore it has all the same features. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient for data-intensive real-time applications. While it is documented, it is relatively new and not widely used, though it is gaining popularity[27].

Because of the wide support for and the past experience with PHP, this is the best candidate for the game server, but because WebSockets are not straightforward to implement in PHP, Node.js should be used for the logging, which will use this protocol.

#### 2.3.2.2 Database

For proper data management a database should be used. Due to the large quantity of data (heavy logging for data analysis) needed by the application to analyse

[8] As AJAX is a group of different techniques, AJAX support refers to the support of these

and evaluate user performances, the size of the database can and will increase very rapidly if the game is played on regular basis, and this should be taken into consideration when designing it.

The application should also contain user authentication in order to get certain administration privileges (such as user CRUD[9]). The database should therefore also handle data management for users. Another reason for user authentication is the ability to split users up into classes or groups, such that teachers will be able to manage and review the performances of the pupils in their respective classes.

| Database | Level of knowledge | Platform |
|---|---|---|
| MSSQL | Medium | Windows |
| MySQL | High | Windows, Linux, OSX |
| Percona Server | High | Windows, Linux, OSX |
| NoSQL | Low | Windows, Linux, OSX |

**Table 2.4:** Comparison of databases

Below is an assessment of the database types mentioned in table ??.

**MSSQL** is a RDBMS[10] from Microsoft and is often paired with ASP.NET. This is also the only database software of the mentioned that is not free to use. It is the obvious choice if ASP.NET was chosen for the game server, see section 2.3.2.1, but it is a huge drawback that it is commercial.

**MySQL** is one of the most widely-used RDBMSs and is very popular to use as a database in web applications. It is cross-platform supported and features independent storage engines, such as InnoDB. InnoDB provides referential integrity and transactions. Percona Server is a fork of MySQL, and one of the primary benefits of this over regular MySQL is the stability and performance boost of InnoDB. Enabling logging on the server will require a high performance database solutions to cope with the possible amount of log entries inserted per second, and Percona Server is preferred over MySQL for this.

**NoSQL** is a database that does not manage data like the relational databases, but in a more simple model. This can for instance be key-value storage or graph storage. The original intention of NoSQL was modern web-scale databases, but the term NoSQL is often thought of as a collection of common characteristics such as:

- Schema-free
- Easy replication support
- Simple API
- Eventually consitent / BASE[11]

[9] Create, Read, Update and Delete

[10] Relational Database Management System

[11] **B**asicly **A**vailable, **S**oft state, **E**ventually consitency. See [29] for a comparison of ACID and BASE.

NoSQL databases are gaining traction, and the most popular NoSQL database is MongoDB[4][31], while databases like 'Cassandra', 'Hypertable', 'CouchDB', 'DynamoDB' and 'Redis' are some of the databases most have heard mentioned due to their origin from companies like Amazon, Google and Facebook.

MySQL seems most appropriate because of its relational storage, its support and past experience working with it in PHP.

# 3 Design

Based on the analysis of the conceptual requirements for the game, this chapter describes a platform-free design. The game design covers the design of the world, the system and content, the story, the levels and the user interface. A database design is also described, which is also based on the technical requirements.

## 3.1 User module

A user management module should be in place and greet the user. Only by logging in should a player be able to play the game, i.e. the user has to be authenticated. The application needs to be protected by some access control for multiple reasons:

- Log data is based on users
- Pupils can be categorized into classes or groups

To set up this authentication and authorization, the application would require users to sign up with some credentials such as username and password. Getting the pupil's age would allow for some statistics and perhaps be of benefit if the difficulty method was expanded to use such information, see section 2.2.5.

When the user module has authenticated the request, the user should be able to play the game.

## 3.2 Game

Each week, thousands of garbage trucks drive around the neighbourhoods to collect trash from the households.

The trucks do not compress the trash when collecting it, and each of them has space for different amounts. It is very important that the trucks are completely filled before they drive back to the junk yard, so that there is not too little or too much trash in them. However, the garbage men do not have an overview of the size of the trash before actually collecting it and therefore they need the player's help.

The game is titled "Clean Town" ("Ren By" in Danish).

### 3.2.1 Rules

The player control one garbage truck at a time. Each truck has a given capacity and starts in the start tile.

The houses in the world have a trash package of a given size, and it is up to the player to find the packages that fit the truck capacity - it can be one or more packages.

In order to fill the truck with a trash package, the player must click on the house with said package.

It is important to pick up the trash packages that fit the truck's capacity, such that the truck ends with neither too little nor too much trash. Optimal filling will give bonus points.

A truck can at any time be driven out if the player wants by clicking on the exit cell, but a optimally filled truck will drive out automatically.

After a truck has driven to the exit tile, the player gets control of the next one.

The game is won when all garbage trucks have collected garbage and has driven to the exit tile.

### 3.2.2 Graphical User Interface

The world is seen from above in a 2/2.5D perspective and is tile based. The different types of tiles are easy to distinguish and consist of the following: truck, road, house and grass.

There is a marked start tile and a marked exit tile in the road network where each truck enters and leaves the game when appropriate. A mockup of the trucks and road network can be seen in figure 3.1.

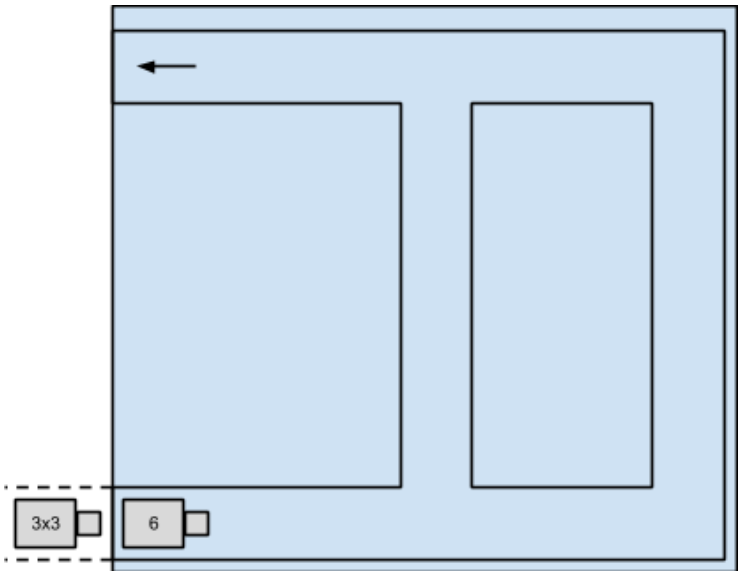



**Figure 3.1:** Truck and road network mockup

At all times the player should be able to see the capacity and fill of the current truck in a given representation. A mockup of this can be seen on figures 3.2 and 3.3

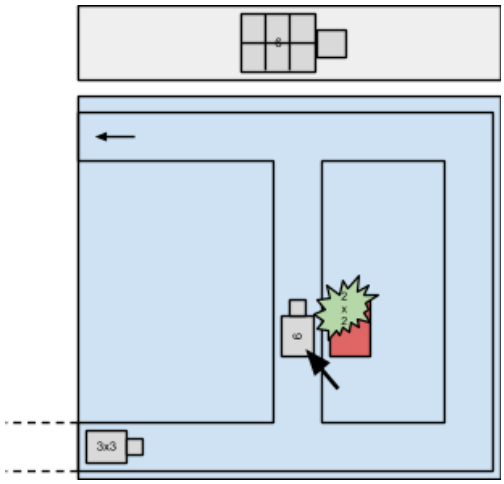



**Figure 3.2:** Capacity before trash pick up

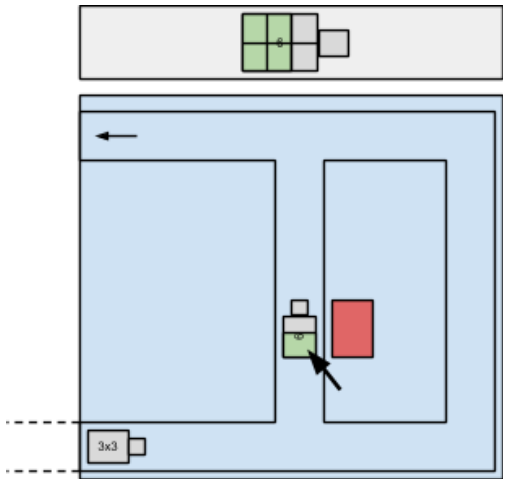



**Figure 3.3:** Capacity after trash pick up

#### 3.2.2.1 Trash indication and representation

Because of the different representations, especially the tables that take up a lot of space, there is not enough space to show the trash size just above a house, as shown in figure 3.4. If the size really was to be shown above the house, it could potentially float off the world, if the house was positioned too close to the border. A lot of considerations have been taken into account in order to find the golden mean between showing the size as clear as possible to the player and keeping a good user experience by not hiding other information or breaking the interface.

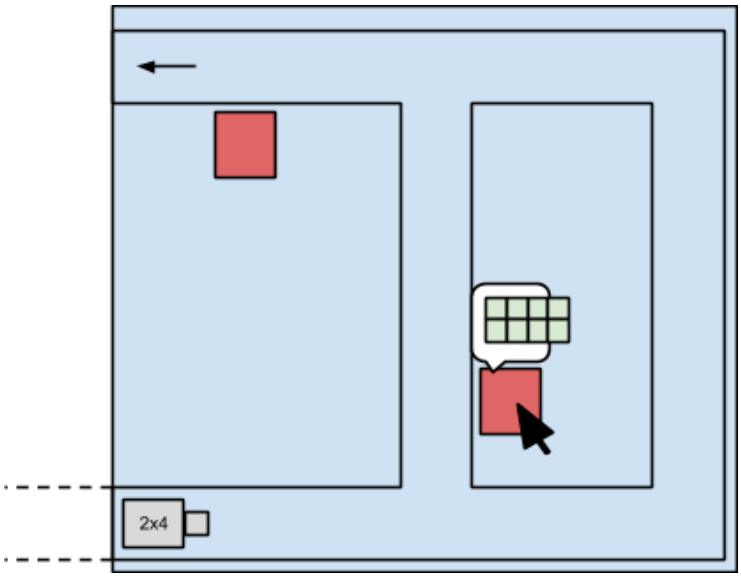



**Figure 3.4:** Representation mockup 1

Instead of letting the player guess which houses contain packages of trash, when

not showing the trash size, there should be an indicator saying which houses that are relevant in the game. This indicator can e.g. be a exclamation mark just above all the houses with trash. The trash size should then be displayed to the player when commanded.

Moving the cursor over a house with trash will reveal the size in the given representation. Displaying it above the house has been ruled out, and displaying it in the center of the screen is also not possible, because some relevant tiles like the truck or other houses could then be hidden behind it as seen in figure 3.5.

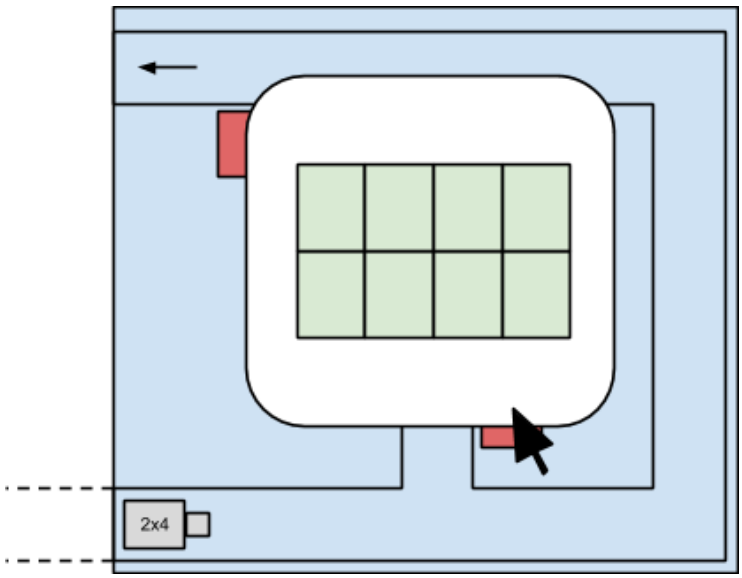



**Figure 3.5:** Representation mockup 2

The optimal solution was found to be displaying the size in a so called information bar when the cursor is moved over a house. This can be seen on figure 3.6.

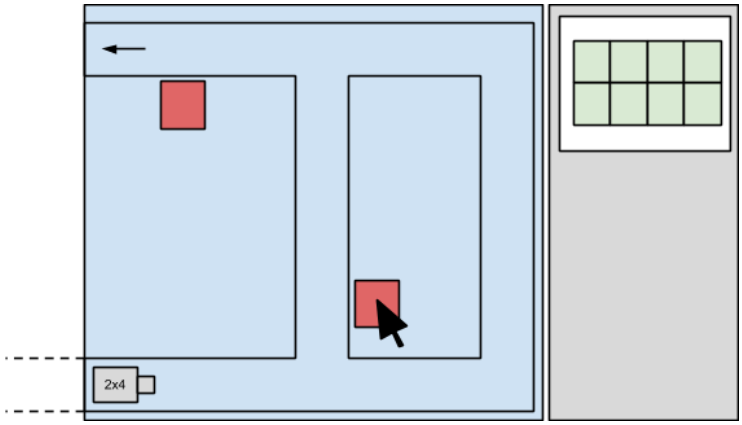



**Figure 3.6:** Representation mockup 3

#### 3.2.2.2 Information bar

Many games with a similar perspective as this game contain a bar that shows relevant information to the player. By dividing the game canvas into small sub-

canvases, it gives a simple overview of the current status while not stealing focus from the actual task in the game. The information bar can be attached to each side of the world, most convenient in either the left or right side, and not take up too much space.

The relevant information is:

- Representation of the trash size in a given house
- Representation of the fill and capacity of the current truck
- Points (see section 3.2.4.5)
- Remaining trucks

#### 3.2.2.3 Animation

To bring the game to "life", some movements must exist, and to make it clear what is going on, transitioning from one state to another must somehow be shown to the user. This means that the movement of a truck should be displayed as if the truck is actually driving to its destination, instead of just making it jump from A to B in a flash. When the player obtains points, it should be clear how he got them and where they came from.

### 3.2.3 Gameplay

The interaction in the game is done by simple gestures with the cursor. It does not require a keyboard to be played.

To see the size of the trash in a given house, the cursor should be moved over said house, and the size will then be shown in its representation in the information bar. When moving the cursor away from the house, the shown size will disappear from the graphial user interface. Moving the cursor over a house without trash or a house that already has been visited does nothing.

To make the truck collect trash from a house, the cursor should be moved over said house and then left clicked once. The truck will then drive to the house and upon visit, the trash will be transferred to truck. Clicking on a house without trash or a house that has already been visited does nothing - it is not possible to make the truck drive to such houses.

When a truck is full or overfilled, it will automatically drive to the exit, but clicking at the exit at any time will make the truck drive to it.

A truck in motion cannot be stopped, but its path can be changed by clicking on another valid tile, i.e. a house with trash or the exit.

A list of use cases for the game can be found in Appendix A.

### 3.2.4 System

Algorithms in the underlying structure of the game control the functionality. These algorithms ensure that the game in all cases behaves as specified, that the solutions to the mathematical problems are correct, and therefore implicitly that the game can be played and won.

#### 3.2.4.1 Path finding

The path finding of trucks uses an A* algorithm because of the use of a special heuristic giving superior strength when the nodes in a graph should be treated differently.

The weighs of the different tile types should be:

- Road: very low
- House: very high

The bigger the span between the sizes are, the less possible for driving through a house it will be. When working with a grid map where the tiles are squares, as the game will consist of, each node will have 8 neighbours. Because a truck should not be able to move diagonally, i.e. only have 4 directions of movement, the path finding should reflect this. This can be done by using a special distance heuristic, and the Manhattan distance[26] is just what is needed. It can be seen in equation 3.1, where *node* is the node currently being inspected and *goal* is the node that is searched for. $D$ is the minimum cost from moving from one node to an adjacent node.

$$\text{heuristic}(\text{node}, \text{goal}) = D \cdot \left( |\text{goal}_x - \text{node}_x| + \left|\text{goal}_y - \text{node}_y\right| \right) \tag{3.1}$$

#### 3.2.4.2 Logging

To get information about how a player performs, some data needs to be logged. Every time a truck collects some trash, the most relevant data to be logged is:

- Level identifier
- Truck representation
- Truck capacity
- Truck fill before trash pickup
- Truck fill after trash pickup
- Trash size
- Trash representation

This data is the most critical to make an assessment of how a player performs, i.e. whether he or she can solve the given mathematical problems.

#### 3.2.4.3 Generation and difficulty

The choice fell on dynamic levels because learning happens in varying paces across pupils. The difficulty is therefore based on previously played games.

This requires algorithms to calculate the number of trucks, houses and trash, the capacity of trucks and the size and spread of trash, based on some parameters from the X latest finished games. The chance of a given representation in a game is also based on how well the player has solved problems with said representation before. Because a level of difficulty is based on the player performance in previously played games, it should be calculated by analysing the log data from these, see section 3.2.4.2.

The first 3 games are meant to calibrate the difficulty; these games will be referred to as calibration.

Following the constraints analysed and noted in section 2.2.3.2, the number of trucks should be calculated with equation 3.2, where $n$ is the number of played levels and $i$ is the number of levels being taken into consideration, i.e. there can be 100 played levels, but only the $i$ latest levels are used, thus $i \leq n$. $t_i$ is the number of trucks used in the $i$ latest games, and $p_i$ is the number of perfect fills in the $i$ latest level.

$$t = \begin{cases} 2, & n < 4 \vee \frac{p_i}{c_i} \in [0.50; 0.85] \\ \left\lceil \frac{t_i}{i} \right\rceil + 1, & \frac{p_i}{t_i} > 0.85 \\ \max\left( \left\lceil \frac{t_i}{i} \right\rceil - 1, 1 \right), & \frac{p_i}{t_i} < 0.50 \end{cases} \tag{3.2}$$

Thus the number of trucks can be increased or decreased by one in each level, and only after calibration. $\frac{p_i}{t_i}$ indicates how well the player has scored overall in the last $i$ levels. The minimum number of trucks to be generated is 1.

$(r_{min}, r_{max})$ defines a range of numbers of which a level can contain. The formula for determining the range for a given level is shown in equation 3.3.

$$(r_{min}, r_{max}) = \begin{cases} (1, 10), & n < 4 \vee \frac{p_i}{t_i} \in [0.50; 0.85] \\ (\lceil r_{min} \cdot 1.2 \rceil, \lceil r_{max} \cdot 1.2 \rceil), & \frac{p_i}{t_i} > 0.85 \\ (\lceil r_{min} \cdot 0.9 \rceil, \lceil r_{max} \cdot 0.9 \rceil), & \frac{p_i}{t_i} < 0.50 \end{cases} \tag{3.3}$$

The range can be increased by 20% or decreased by 10% after calibration. Note that as the difficulty gets harder, the range will not only be expanded but also moved. This way, players at hard difficulties will not be presented with too small numbers.

The capacity of each truck can be calculated like equation 3.4.

$$c = \text{rand}\left(r_{min},\ r_{max}\right) \tag{3.4}$$

Each truck must have a perfect fit with some trash, and this trash can be spread into several packages. The spread of the trash for each truck should be calculated like equation 3.5.

$$s = \begin{cases} 1, & r_{min} \cdot 2 > c \vee c < 10 \\ \text{rand}\left(1, \left\lfloor \frac{c}{r_{min}} \right\rfloor\right), & c = 10 \\ \text{rand}\left(\min\left(\left\lfloor \frac{c}{10} \right\rfloor, \left\lfloor \frac{c}{r_{min}} \right\rfloor\right), \left\lfloor \frac{c}{r_{min}} \right\rfloor\right), & c > 10 \end{cases} \tag{3.5}$$

When knowing the spread of the trash for a truck, the accumulated size of the packages should correspond to the truck's capacity which has already been calculated. The size of each trash package for a truck should be calculated as shown in equation 3.6. The equation should be iterated $s$ times for each truck, such that $s$ will be decreased whenever a package size $t$ has been calculated. In the first iteration, $max$ is equal to $c$, and for all other iterations, the calculated $t$ will be subtracted from $max$.

$$t = \begin{cases} max, & s = 1 \\ \text{rand}\left(r_{min}, \text{rand}\left(\left\lfloor \frac{max}{2} \pm 2 \right\rfloor\right)\right), & s = 2 \wedge rand(1,2) = 1 \\ \text{rand}\left(r_{min}, \left\lfloor \frac{max}{2} \right\rfloor\right), & s \geq 2 \end{cases} \tag{3.6}$$

The number of houses is the same as the number of trash packages, i.e. there will not be generated any houses without trash.

The road network is generated completely at random without a known seed, but the probability of intersections is increased as the difficulty gets harder, creating a more complex network. Houses cannot be placed beside the borders, as the tiles here are in a protected area only used to place the start and exit. These will always be placed in a way so that they face each other, i.e. north and south or east and west. Their exact locations are random though, so they might not be placed in a straight line.

The probability that an intersection will be placed at any given spot is given in equation 3.7.

$$int = 0.2 + \frac{trucks - 2}{30} \tag{3.7}$$

The probability for 1 truck, i.e. the minimum probability is 0.167 (16.7%). The equation will increase the intersection probability by 0.1 for every 3 trucks. When intersection points have been placed, the rest of the road network will be generated. A* is used for this and also uses the Manhattan heuristic, mentioned in section

2.2.2.3, because the roads cannot be connected diagonally. The start and exit cells are also connected to the rest of the road network this way.

#### 3.2.4.4 Representations

The value representations to be used in the game are:

- Basic multiplication equations
- Areas of tables
- Numbers

These are all relevant when dealing with addition and multiplication, and it should again be stressed out that each of them can represent the same value, which is important that the pupil understands.

The probability of getting a representation, $P_j$, should be calculated as shown in equation 3.8 where $R$ is the possible representations, $R_c$ is the correctness percentage of a representation and $R_j$ is the representation for which the probability is calculated. The correctness percentage of a given representation is found by inspecting all trucks from the $\mathsf{X}$ latest finished levels. If a truck is perfectly filled, the representations of the trash that this truck has collected will then count as being correct. Otherwise it will count as incorrect.

$$P_j = \frac{100}{\sum_i R_{ic} \cdot (|R| - 1)} \cdot \left( \sum_i R_{ic} - R_{j_c} \right) \tag{3.8}$$

While $\frac{100}{\sum_i R_{ic}} \cdot R_{j_c}$ would increase the probability for a given representation, if the player was good at solving problems with said representation, equation 3.8 instead "reverses" the probability. This means that representations with a high percentage of correctness will have a lower probability of getting picked, and vice versa.

#### 3.2.4.5 Points

Each truck's capacity represents the maximum amount of points possible when collecting trash for it - excluding any bonuses there might be.

It would be rather harsh if a player is not granted any points when a truck is slightly overfilled. When the target audience is young and because the game should be educational, some compensation for small miscalculations is a good idea to stop the players from giving up entirely. A way to compensate for miscalculations is to use some form of normal distribution around the truck's capacity for calculating the points. To balance between being harsh and compensating the amount of given points could decay by 33% for each point away from the truck's capacity, such that if a player overfills a truck by 4, 0 points are given. Equation 3.9 shows how points are calculated.

$$points = \begin{cases} t_s, & c_f + t_s \leq c_c \\ 0, & \frac{|c_c - (c_f + t_s)|}{3} > 1 \\ \left\lfloor \left(1 - \frac{|c_c - (c_f + t_s)|}{3}\right) \cdot t_s \right\rfloor, & \frac{|c_c - (c_f + t_s)|}{3} \leq 1 \end{cases} \quad (3.9)$$

The variables in equation 3.9 can be read as:

$t_s$ = trash size
$c_c$ = truck capacity
$c_f$ = truck fill

To motivate the player when he solves a problem correctly, it should be possible to gain combos. Rewarding is a great motivator and encourages the player to keep trying hard. The combos are increased by one for each problem solved correctly, and they will multiply any points obtained from a trash package with the current combo size. I.e. if the player has filled one truck perfectly, the combo will be increased from one to two, and thus the next points will be multiplied by two. Combos are reset to one whenever the player solves a problem wrong.

## 3.3 Database

The database will contain information about users, games and played levels. For each played game there should be details about trucks, packages, houses, truck routes and additional log data.

This data can be used to make in-depth analysis of each played game and makes it possible to even re-watch how the pupil has played.

The database is designed with the structure shown in figure 3.7. This structure is relational, and thus a RDBMS is needed to implement it.

While each level is unique by its id, the contents of the level, such as trucks and houses, can be reused to reduce the size of the database. The contents are also unique in the database, but the same content can be used for more levels. Because the database is relational, the tables are connected with different relations such as one-to-one, one-to-many and many-to-many. I.e. an instance in user_games can have many trucks instances.

games will contain the actual games, i.e. the database can contain more games than just "Clean Town". user_games will then contain data about levels played by the users.

For log data and additional level data, the EAV[1] model has been used. This allows for managing data in a much more flexible way as rows instead of traditionally in static columns. Because the database can be used to store data from different types of games, each game - perhaps even each played level - might need to manage different data. Instead of having to restructure the database design with new columns when adding new types of data, it can instead be stored in its own row.

[1] Entity-attribute-value

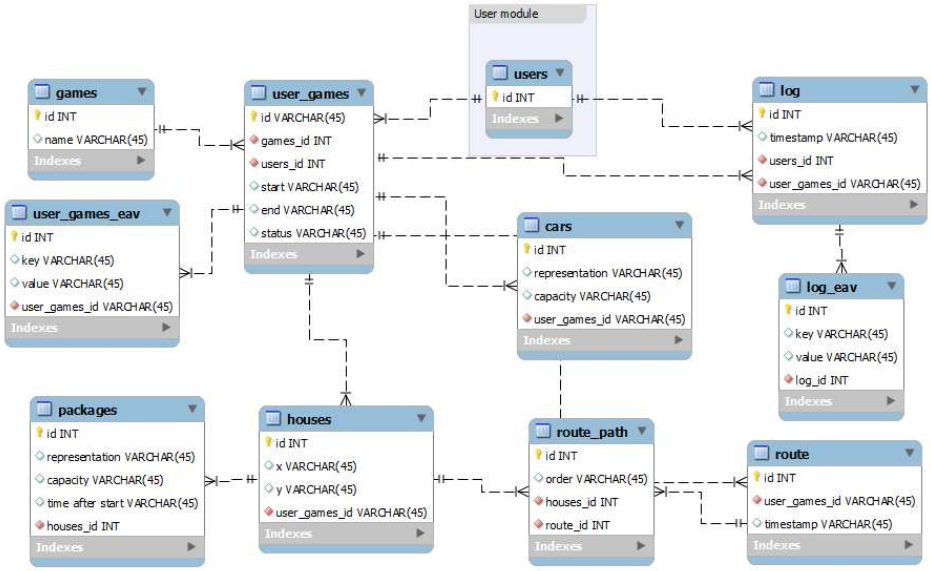



**Figure 3.7:** Database Diagram

However, EAV can lead to heavy, complex queries and should be used sparsely, which is the why packages, houses, trucks and routes have their own table. One can argue that EAV is great to manage lots of different types of data, when the data is not critically needed in the application that uses the data.

# 4

# Implementation

The game has been implemented with methods and tools found to be appropriate based on the analysis, technical requirements and the chosen design. This chapter outlines the development process and how the actual game ended up by using different technologies.

## 4.1 Application structure

Figure 4.1 shows a flow diagram of how the application has been structured technically when a user wants to play a game.

When a user visits the application in a browser, PHP will generate a login page, on which the user enters his or her credentials to log in. These are then sent to PHP that authenticates the user and redirects to another address, where it generates the game page. Upon visit of this page, the user will automatically request a game from Node.js. Node.js generates some game content, and some of it is sent to PHP, which returns a game world to Node.js. The world and content is then sent to the user, where it will be displayed and is ready to be played. During gameplay, some log data is sent from the user to Node.js. Most operations in PHP and Node.js involve some database interaction.

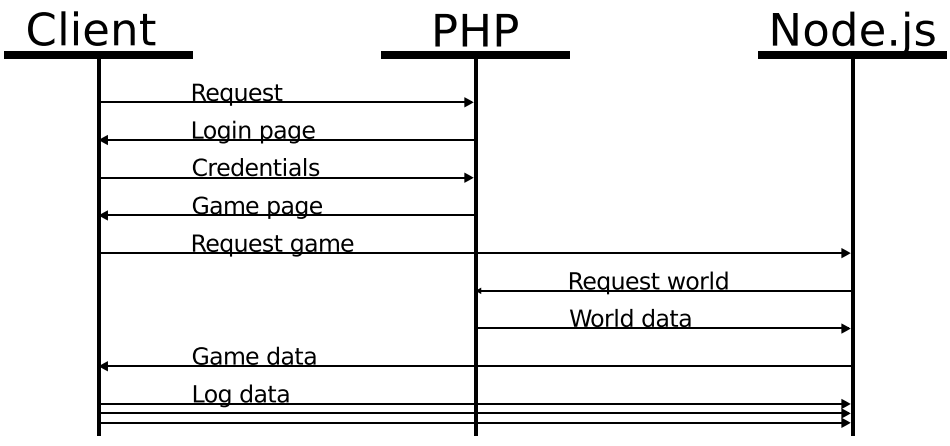



**Figure 4.1:** Application flow

## 4.2 Game server

The game server consists of PHP and Node.js, where the former is mostly used for the user module and the latter for the actual game logic.

During the earliest stages of implementation of the game server, it became clear that it was better to use Node.js for an increasing number of tasks instead of PHP. This was because the server and client should be able to communicate with each other during the game without the player noticing. WebSockets, implemented with Node.js, was an obvious choice for this, and it is therefore used for far more than logging.

### 4.2.1 PHP

The user module in this application has been coded with PHP. This is because the prior experience with PHP is extensive and a previously developed drop-in user management module for FuelPHP provides a great base for the purpose of this project.

> FuelPHP is a simple, flexible, community driven PHP 5.3 web framework based on the best ideas of other frameworks with a fresh start [10].

FuelPHP uses a HMVC[1] pattern and comes with different tools for fast and flexible development of web applications. Past experience with the framework and the fact that it takes advantage of the object orientation of PHP has been the main reason for why it was chosen.

#### 4.2.1.1 Administration

An administration panel has been created to provide management of the users without having to think of unauthorized use. It also provides a simple overview, as seen in figure 4.2, and CRUD for the users. The administration panel is built in a modular way and can be expanded to provide even more information easily.

The user administration is currently utilizing groups which provides some authorization. This can be changed to provide a concept of classes of which some users can be assigned as teachers, so that the teachers for instance can view statistics and analysis of the pupils' performances throughout the game and watch their individual progress.

It is required that the user has created an account in the application and is logged in before he is authorized to play the game. This is because the application needs to identify a player when the game should log information about his or her performance in a level.

[1] Hierarchical model-view-controller

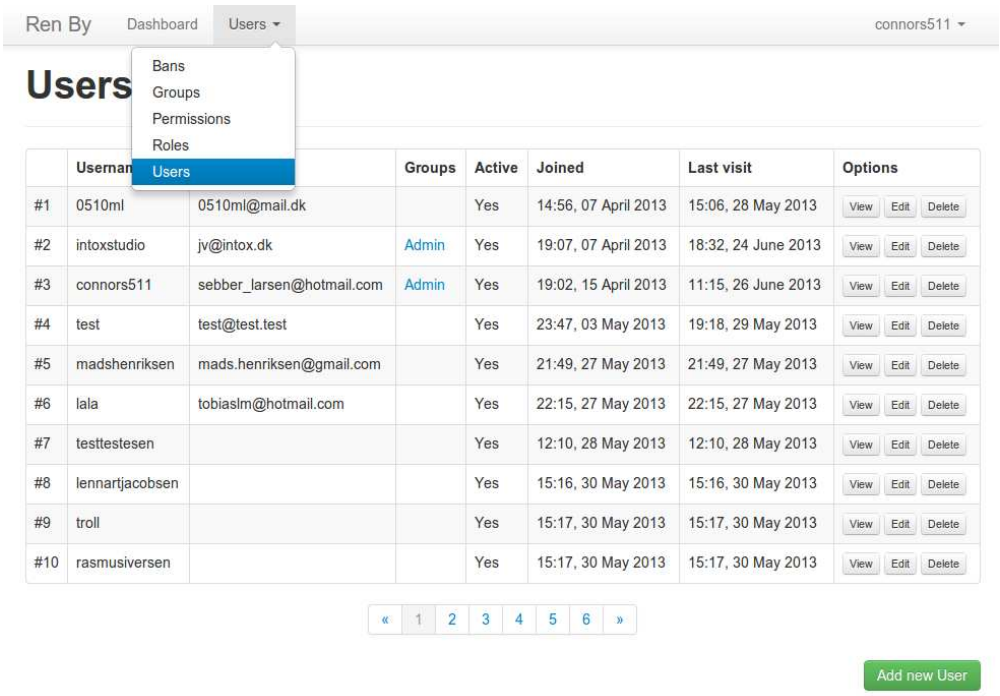



**Figure 4.2:** Users overview

#### 4.2.1.2 Generation of road network

The road network is generated by PHP with A* using a Manhattan distance heuristic. The minimum cost for moving from one node to an adjacent node is determined by the Manhatten distance heuristic, but will be even more expensive if it requires a change of direction due to an alteration made. This alteration will result in more straight roads. See listing 4.1 for the alteration.

```
1  // The g score is the shortest distance from start to current node.
2  // We need to check if the path we have arrived at this neighbor is↩
       the shortest one we have seen yet.
3  $gScore = $node->g + $neighbour->cost;
4
5  // Which direction are we coming from?
6  $dir = 1;
7  if ($node->parent != null && $node->parent->x == $node->x) {
8      if ($node->x == $neighbour->x) {
9          $dir = 0;
10     } else {
11         $dir = 1;
12     }
13 } else if ($node->parent != null && $node->parent->y == $node->y) {
14     if ($node->y == $neighbour->y) {
15         $dir = 0;
16     } else {
17         $dir = 1;
18     }
19 }
20 if ($neighbour->cost == Types::$TYPE_GRASS)
21     // TURN_COST = 5000
22     $gScore += $dir * self::TURN_COST;
```

**Listing 4.1:** The modification to A* that discourages change of direction

PHP will also place the generated houses at random places beside the roads, see appendix C for the full implementation of the world generator, or specifically lines 163-196 for the part that generates houses.

### 4.2.2 Node.js

The asynchronous nature and event driven design of Node.js provides a good base for the heavy logging in the application. Because some functionality of the game was moved to the server, which increased the need for a high performing I/O server, see section 4.5, Node.js has also been put in place as the central game server.

Node.js allows the use of both AJAX and WebSockets for communication between the client and the server. AJAX is just HTTP calls, which would obviously be supported out of the box, and support for WebSockets could come from a Node.js module such as Socket.IO, see section 4.3.1.

#### 4.2.2.1 House, truck and trash generation

The top-down method for content generation, noted in section 2.2.3.2, proved to be hard to implement and showed to be rather computational expensive. Instead the bottom-up method, also noted in 2.2.3.2, was chosen because of its simplicity, which means that it is easier to ensure that the amount of content match up.

Trucks, trash and houses are generated with Node.js and the algorithm could therefore be optimized by its asynchronous and parallel behaviour. Figures 4.3 and 4.4 accordingly show how both the top-down and bottom-up methods would look.

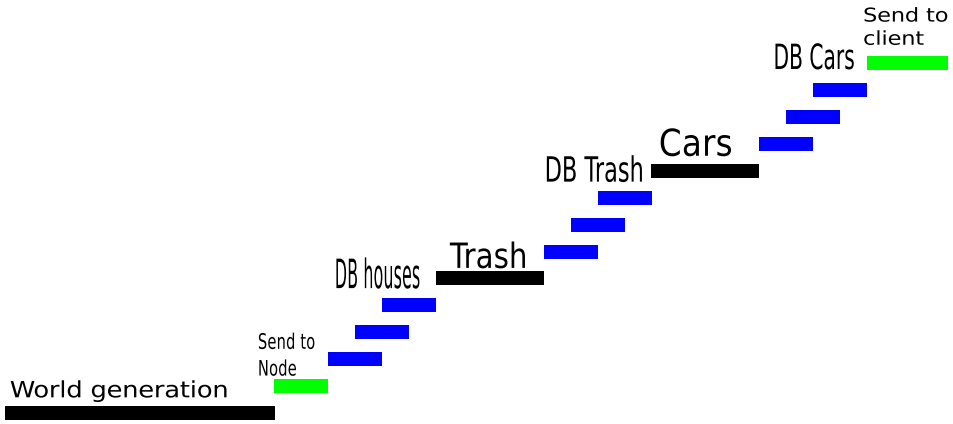



**Figure 4.3:** Top-down content generation in Node.js

Black bars represent processing, blue bars involve database interaction and green bars represent transmission over the wire and involves latency. All bars do not correctly represent time units, but are a rough indication of where and how processing time is used.

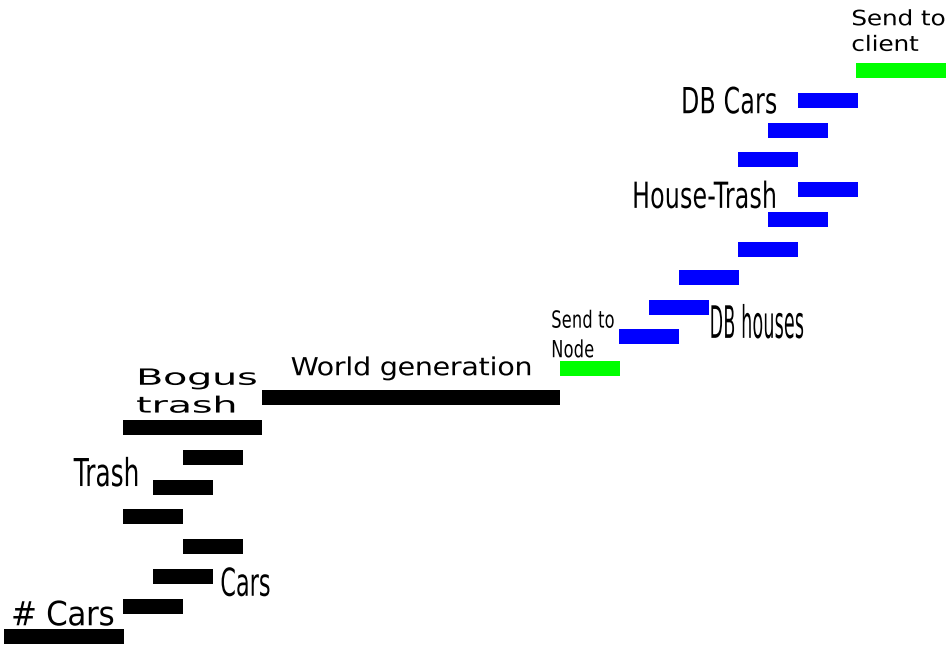



**Figure 4.4:** Bottom-up content generation in Node.js

Besides being a simpler algorithm, the bottom-up method shows to make most use of the parallel optimization of Node.js.

When a level is being generated, the first part is to determine the number of trucks on the level by examining the data from the latest 10 games. This is done with the functions seen on listing 4.2 and 4.3.

```
1  function getStats(cb) {
2      var stats = [];
3      // Base stats on the last 10 games
4      db.get_stats(sessionMgm.getSessionById(client.id).user_id, 10, ↩
           function(res) {
5          var std = {
6              range: {
7                  min: 1,
8                  max: 10
9              }
10         };
11
12         var reps = getPossibleRepresentations();
13         for (var i = reps.length - 1; i >= 0; i--) {
14             if (!res.stats.reps[reps[i]]) {
15                 res.stats.reps[reps[i]] = {
16                     correct: 0,
17                     wrong: 0
18                 };
19             }
20         };
21
22         // We need 3 games to make stats
23         if (res.length < 5) {
24             return cb(std, res);
25         }
26
27         // If more than 75% of the pickups is perfect.. increase ↩
```

```
            range
28          // If less than 50% of the pickups is perfect.. decrease ↩
                range
29          if (res.total.perfects / res.total.cars > 0.75) {
30              // Increase range by 20%
31              // This will increase the gap between min and max.
32              return cb({
33                  range: {
34                      min: Math.ceil(res.stats.range.min*1.2),
35                      max: Math.ceil(res.stats.range.max*1.2)
36                  }
37              }, res);
38          } else if (res.total.perfects / res.total.cars < 0.50) {
39              // Decrease range by 10%
40              return cb({
41                  range: {
42                      min: Math.ceil(res.stats.range.min*0.9),
43                      max: Math.ceil(res.stats.range.max*0.9)
44                  }
45              }, res);
46          } else {
47              return cb(std, res);
48          }
49      });
50  }
```

**Listing 4.2:** Algorithm for calculating statistics for the last 10 games

Listing 4.2 shows that it sums up the amount of correct and wrong solutions the user has given with different representations before it determines whether or not it should increase, decrease or stay in the current range of numbers based on how many perfects fills the user have had in the past 10 games. As seen on line 4, it is easy to increase the amount of games taken into consideration which would allow for even more fine tuning of the difficulty.

```
1   function getCarCount(stats) {
2       var std = 2;
3       // We need 3 games to make stats
4       if (stats.length < 5) {
5           return std;
6       }
7
8       // If more than 85% of the pickups is perfect.. increase cars
9       // If less than 50% of the pickups is perfect.. decrease cars
10      if (stats.total.perfects / stats.total.cars > 0.85) {
11          // Increase cars by one above avg
12          return Math.ceil(stats.total.cars / stats.total.games) + 1;
13      } else if (stats.total.perfects / stats.total.cars < 0.50) {
14          // Decrease cars by one below avg
15          return Math.max(Math.ceil(stats.total.cars / stats.total.↩
                games) - 1, 1);
16      } else {
17          return std;
18      }
19  }
```

**Listing 4.3:** Algorithm for determining the number of trucks for a course

Listing 4.3 uses the same technique as listing 4.2 to determine whether or not the

number of trucks should be increased, decreased or stay the same. If the trucks should be increased or decreased it is done by adding or subtracting one from the average number of trucks from the last couple of games.

After the number of trucks has been determined the trucks and trash are generated and saved to the database as seen on figure 4.4. When the number of trucks has been found, each truck is assigned a random capacity within the allowed number range for current level with the algorithm found in listing 4.4.

```
1  function getRandomRepresentation(number, stats) {
2      var std = 2;
3      // We need 3 games to make stats
4      if (stats.length < 5) {
5          // We dont have all the data
6          if (number == 1) {
7              // 1 will cause fractions if used for x*y format
8              return rand(1,2);
9          } else {
10             return rand(1,3);
11         }
12     }
13
14     if (number == 1) {
15         // 1 will cause fractions if used for x*y format
16         return rand(1,2);
17     }
18
19     // (100/(sum af korrekthed*(antal mulige reps - 1))*(sum af ↩
           korrekthed-korrekthed af rep))
20     var chance = rand(1,100);
21     var c = 0, tmp = 0;
22     var cp = getPossibleRepresentations().length - 1;
23     for (var key in stats.stats.reps) {
24         if (key == 'total')
25             continue;
26         tmp = (100/(stats.stats.reps.total.correct*(cp))) * (stats.↩
               stats.reps.total.correct - stats.stats.reps[key].correct↩
               );
27         if (tmp == 0) {
28             return key;
29         }
30         if (chance <= c + tmp) {
31             log("Rep = " + key + " it is", 'rnd');
32             return key;
33         } else {
34             c += tmp;
35         }
36     };
37     // Should never happen
38     return rand(1,3);
39 }
```

**Listing 4.4:** Algorithm for finding a semi-random representation

When the trucks is assigned a capacity the algorithm shown in listing 4.5 decides how many pieces of trash that capacity should be split into. A few rules are set, as seen in listing 4.5, to ensure that the front end can render some fairly readable representations of the values.

```
1  function getTrashCountForCar(car, range) {
```

```
2       // If the range.min is not at least twice as big
3       // you cant split the car
4       // car.capacity also needs to be above 10 before splitting
5       if (range.min * 2 < car.capacity || car.capacity < 10) {
6           return 1;
7       }
8       // 10 is fairly readable when rolling table representation
9       var min = car.capacity > 10 ? 2 : 1;
10      if (min == 2) {
11          // Minimum amount of splits equal which ever is less:
12          // 10% of full capacity or how many range.min packages
13          // car.capacity can contain
14          min = Math.min(
15                  Math.floor(car.capacity / 10),
16                  Math.floor(car.capacity / range.min)
17              );
18      }
19
20      return rand(min, Math.floor(car.capacity / range.min));
21  }
```

**Listing 4.5:** Algorithm for spreading trash

After the trucks and trash have been generated, the number of houses needed is known. A request to the PHP world generator is sent with the number of houses required and the possibility of an intersection, see section 3.2.4.3 for details.

After the world is generated and sent back to the Node.js server, the world is processed and houses are parsed and saved to the database. Afterwards the world, trucks, trash and houses are sent to the game engine on the client.

As seen on figure 4.4 'DB cars' and 'house-trash' operations are executed in parallel and 'DB houses' and 'house-trash' operations are executed in a 'waterfall'. These methods of execution are achieved with Node.js' Async module[20].

The parallel function runs an array of functions in parallel, without waiting until the previous function has completed while the waterfall method runs the function in series passing each of the functions result to the next function in the array.

```
1   async.waterfall([
2       function (callback) {
3           // Add houses to db
4           async.each(houses, function (item, cb) {
5               db.add_house({
6                   x: item.x,
7                   y: item.y,
8                   user_games_id: sessionMgm.getSessionById(client.id)↩
                        .user_games_id,
9                   index: item.index
10              }, function(data) {
11                  houses[data.index].id = data.id;
12                  cb(null);
13              });
14          }, function (err) {
15              sessionMgm.getSessionById(client.id).houses = houses;
16              callback(null, houses);
17          });
18      },
19      function (houses, callback) {
```

```
20
21          async.parallel({
22              trash: function(callback) {
23                  // Connect houses and trash
24                  async.each(trash, function (item, cb) {
25                      item.house_id = getHouseForTrash(item, houses);
26                      // Add trash to db
27                      db.add_trash({
28                          representation: item.representation,
29                          capacity: item.capacity,
30                          house_id: item.house_id,
31                          index: item.index
32                      }, function(data) {
33                          trash[data.index].id = data.id;
34                          cb(null);
35                      });
36                  }, function (err) {
37                      if (err) {
38                          return callback(err);
39                      }
40                      sessionMgm.getSessionById(client.id).trash = ↩
                            trash;
41                      callback(null, trash);
42                  });
43              },
44              cars: function(callback) {
45                  // Save cars to DB
46                  async.each(cars, function (item, cb) {
47                      db.add_car({
48                          representation: item.representation,
49                          capacity: item.capacity,
50                          user_games_id: sessionMgm.getSessionById(↩
                                client.id).user_games_id,
51                          index: item.index
52                      }, function(data) {
53                          cars[data.index].id = data.id;
54                          cb(null);
55                      });
56                  }, function (err) {
57                      // Error happened
58                      if (err) {
59                          return callback(err);
60                      }
61                      sessionMgm.getSessionById(client.id).cars = ↩
                            cars;
62                      callback(null, cars);
63                  });
64              }
65          },
66          function(err, results) {
67              // results is now equals to: {trash: [..], cars: [..]}
68              callback(null, null);
69          });
70      }
71  ],
72  function(err, results) {
73      // Send game data to client
74      client.emit('world data', {
75          cars: sessionMgm.getSessionById(client.id).cars,
76          houses: sessionMgm.getSessionById(client.id).houses,
```

```
77          trash: sessionMgm.getSessionById(client.id).trash,
78          world: data
79      });
80  });
```

**Listing 4.6:** Asynchronous and parallel behavior of Node.js

#### 4.2.2.2 Path finding

When a user clicks on either a house with trash or the exit tile, the client sends a request to the server informing the server that the current user would like his current truck to drive to this particular point.

The server validates parts of the request and then tries to find a path in the graph representing the level using A*. If a path is found, the server sends the path information to the client, which in turn instructs the truck to drive via the path.

## 4.3 WebSocket

After testing performance on WebSockets vs AJAX requests for the logging, WebSockets was a clear winner.

**Table 4.1:** Time spend sending requests to the server. All numbers are in milliseconds

| | pickup trash 1 | | pickup trash 2 | | house click | |
|---|---|---|---|---|---|---|
| # Requests | WS | AJAX | WS | AJAX | WS | AJAX |
| 10 | 3.5 | 86.4 | 4.6 | 80.5 | 5.2 | 49.6 |
| 100 | 15.7 | 2,103.0 | 12.1 | 2,334.1 | 7.8 | 2,717.1 |
| 1000 | 80.9 | 425,285.9 | 69.2 | 452,712.6 | 70.8 | 471,607.4 |

Table 4.1 shows the time it took sending a number of requests to the server based on the type of message. The tests were performed by timing how long it took the client to send 10, 100 and 1000 requests to the server. Each test was repeated 10 times and the table reflects the average time spent per test.

It was found that the size of an AJAX request was 157B while the WebSocket was just 70B, a 55% size reduction **per request**.

Simulating a high number of concurrent users sending log data at the same time from a single machine like this test, may not be entirely accurate, but will give some indication of the actual performance.

The code used for finding the results from table 4.1 can be found in appendix B.

### 4.3.1 Socket.IO

Socket.IO is a JavaScript library for Node.js that wraps multiple protocols, but primarily uses the WebSocket protocol. It is designed to start off with a WebSocket connection, and if that is not successful, degrade through its fallbacks until it finds a transport method that works.

Socket.IO supports WebSocket, Adobe Flash sockets, JSONP polling and AJAX long polling. The benefit of using Socket.IO is that it uses the same event driven interface for all protocols, such that the developer does not have to worry about which protocol is actually in use for a given client. This is great for availability, because, as mentioned in section 2.3.1.3, older browsers do not support WebSockets.

Besides being a wrapper for WebSocket it also provides a lot of other useful features such as broadcasting to multiple sockets at once, storing data associated with each client[2] and asynchronous I/O.

### 4.3.2 Logging

The low overhead of messages transmitted over an open WebSocket connection provides the possibility to log a lot of data without the user ever noticing due to the very low latency compared to making AJAX requests which, for each request would require the client to send HTTP headers and log data before waiting for and receiving HTTP headers confirming whether or not the log data was actually received by the server.

The application is logging a long list of parameters. For each level, the following is saved in the user_games table:

- User id
- Start time
- End time
- Amount of perfects
- Amount of combos
- Number range
- Final score
- Status (started, ended)

For each truck, the following is saved in the cars table:

- Representation
- Capacity
- Game id

For each package of trash, the following is saved in the trash table:

- Representation
- Capacity

[2] Also known as session data

- Game id
- House id

For each house, the following is saved in the houses table:

- X position
- Y position
- Game id

For each logged event during a level the game id is logged with a timestamp and saved to the log table. The event can either be a click event, which contains the following extra information and is saved in the log_eav table:

- Truck id
- Truck position
- Click coordinates

Or a trash pickup event, which contains the following extra information and is saved to the log_eav table:

- Trash id
- Trash capacity
- Trash representation
- House id
- Truck id
- Truck capacity
- Truck representation
- Truck coordinates
- Truck fill

Most of this logging is used by the algorithms that determine the level of difficulty, see section 3.2.4.3.

#### 4.3.2.1 Analysis of log data

The massive amount of data saved and log data recorded during a game can be used to more than determination of the difficulty and inspection of potential errors. Statistics can be generated to see how a player or a group of players have performed. Such group could be a school class, where the teacher would be able to observe any progress or regress either overall or for each pupil. Some of the statistics that the data is able to provide are:

- Representations
  - Number recognition
  - Pattern recognition
- Number range
  - Multiply tables
- Speed
- Correctness
- Spatial intelligence (more trucks, more trash per truck = a lot to juggle)

## 4.4 Game engine

The choice of game engine fell on Crafty, because of its lightweightness, ease-of-use, documentation and features. With Crafty it is possible to create games both with HTML elements and in the canvas element.

Because of years of prior experience working with HTML and CSS and no experience with the canvas element, the game has been implemented with the former. Using HTML for elements and CSS to describe them is great for rapid prototyping.

### 4.4.1 JavaScript and HTML

In Crafty scenes are used to describe different displays in a game. The scenes used in this game are main and loading. The loading scene shows a text about the game being loaded while waiting for the assets to be loaded in the browser and the game to prepared by the game server. After this, the scene will switch to main, which is where the action happen.

The entity-model in Crafty makes it possible to create components later to be instantiated as entities. When working with HTML, an entity is basically just a node in the DOM tree with some extra properties and functions. The game creates two custom components: TilePos and Car. Because the game is tile based, each road, house, truck, start and exit is a tile, i.e. TilePos. The truck needs some more functionality such as movement and trash management so it also uses the Car component. Because grass tiles are not used for anything in the game, entities are not created for them even though the grass is being generated server-side. This is for better performance because lots of nodes in the DOM tree can slow down a traverse of it.

When entities are created, HTML elements are injected into the document. For houses with trash, an element to act as an indicator is injected as a child node to the tile. Because houses with trash should be able to be described differently than other houses, an extra CSS-class is appended to the element. When trash is collected from the house, both the child node and the CSS-class should be removed from the house. Removing the child nodes and toggling CSS-classes are done with

jQuery[16]. jQuery is a JavaScript framework and used because of its excellent documentation.

The graphics for the tiles are made of images. Because Crafty supports image sprites, all the images are combined into one file as shown in figure 4.5. Each sprite is defined and managed as seen in listing 4.7.

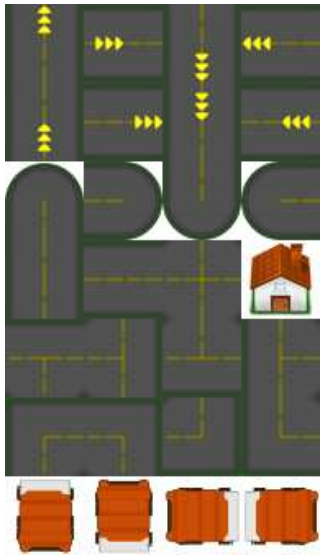


**Figure 4.5:** Image Sprite for tiles

```
1  Crafty.sprite(tileSize, asset_path + 'img/content-sprite.png', {
2          road_horizontal: [0, 3],
3          road_vertical:   [1, 3],
4          road_T:          [2, 4],
5          road_down_T:     [0, 4],
6          road_left_T:     [3, 4],
7          road_right_T:    [1, 4],
8          road_x:          [2, 3],
9          road_right_up:   [0, 5],
10         road_right_down: [3, 5],
11         road_left_up:    [1, 5],
12         road_left_down:  [2, 5],
13         start_up:        [0, 1],
14         start_right:     [1, 1],
15         start_down:      [2, 1],
16         start_left:      [3, 1],
17         end_up:          [0, 0],
18         end_right:       [1, 0],
19         end_down:        [2, 0],
20         end_left:        [3, 0],
21         road_end_up:     [0, 2],
22         road_end_right:  [1, 2],
23         road_end_down:   [2, 2],
24         road_end_left:   [3, 2],
25         car:             [6, 0],
26         house:           [3, 3]
27     });
```

**Listing 4.7:** Crafty Sprite Map

In figure 4.5 and listing 4.7 it can be seen that there are many different images for

roads, one for each direction and intersection, and that there also is an image for each direction of the truck. Two images exist for each start and exit tile in each direction. This is because Crafty can animate between different images - in this case making the arrows "pulse" back and forth. The same way, the image for the truck is switched whenever it changes direction in its movement, which is enabled by the .animate function seen in listing 4.8.

```
1 car = Crafty.e("TilePos, Car")
2   .attr({ z: 11 })
3   .tilePos(start.x, start.y, 0, 'car')
4   .initTrash(data.cars[i])
5   .animate("walk_left", 3, 6, 3)
6   .animate("walk_right", 2, 6, 2)
7   .animate("walk_up", 0, 6, 0)
8   .animate("walk_down", 1, 6, 1);
```

**Listing 4.8:** Crafty truck instantiation

To the right in the game, an entity for a sidebar is created, as well as one for capacity, score and garage. The capacity entity shows the capacity of the current truck, and when the player moves the cursor over a house, it shows the size of the trash in its representation. So instead of needing space for both truck capacity and a size of a trash package in different places, they share the same space. The score entity shows the current score and combos (if any), and when a truck collects some trash from a house, a new temporary entity is created on top of the house. This entity shows the points obtained by picking up the trash and it will fly over to the score entity, where it disappears as the points get updated. The garage entity just shows how many trucks there are left in the game.

### 4.4.2 CSS

CSS is used to style everything that is not displayed with images. It makes the background green (as grass), and it is used to give the sidebar a blue glossy look with wells for capacity, score and garage. Representations in the capacity entity are made big and the table representation is checkered to make it easier for the player to count each table cell. The temporary entity created when points are obtained are also styled with CSS as a white, round bubble with red text. Because the canvas element is not used, Crafty itself uses CSS for the entities to make them appear at the right positions in the game.

The movement of the truck is done with CSS-transitions, which make it float between each tile instead of jumping, when the tile position is updated with JavaScript.

To bring even more movement into the game, the indicators above the houses with trash - which of course are styled with CSS as small white bubbles with red exclamation marks - are animated with CSS. Listing 4.9 shows the snippet of how to make the indicator jump up and down in the game.

Animating and transitioning with CSS is preferred over JavaScript, because CSS is native to a browser engine while JavaScript instead is an interpreted language and will have to modify the properties of a DOM node using timers and loops - this is not at all optimal for many houses with indicators being animated. Because CSS is native to a browser, the animations will be optimised which for instance can avoid

unnecessary calculations and repaints[28][22]. Some browsers even enable hardware acceleration. Animating with CSS is not supported by older browsers because it is part of CSS3, which is a drawback on availability, but because the game engine already requires a modern browser, most animations are implemented with CSS due to its superior performance compared with animations in JavaScript.

```
1  .house-trash {
2    [...]
3    transform: translate(0, -15px);
4    animation:jump 1.2s ease infinite;
5  }
6  @keyframes jump {
7    50% {
8      transform: translate(0, -5px);
9    }
10 }
```

**Listing 4.9:** CSS3 Animation for trash indicator

The CSS-class appended to a house with trash is used to make the cursor be displayed as a pointer when hovering a house. This indicates to the player that the house is clickable.

### 4.4.3 Graphical User Interface

The game is 960x576px wide with tiles 48x48px big and contains a world with 15x12 tiles. The sidebar uses the rest of the width. Today, most computers use a screen resolutions of at least 1366x678px[32][3], so the width, having 28 factors, will fit both the user's screen and the contents of the game. 48px, being one of the 28 factors, is chosen because it seems neither too little or too big. There are room for approx 72 houses in this size, if all should be reachable and when no tiles except start and exit can be generated near the border.

Because of the use of image sprites and CSS, it was easy to replace place-holders with the final graphics - and it will be easy to switch between different themes. The current theme was designed with simplicity and usability in mind without loosing the small details that are treats for the eye.

Figure 4.6 shows how the game looks and indicates the general difficulty presented to new players during the calibration.

As the level of difficulty is increased, the number of houses, trucks and trash will too. The number range also follows the difficulty. To the left in figure 4.7 is a game with slightly higher difficulty, and to the right the difficulty is much higher. In both figures it can be seen that bogus trash has been generated and/or that the trash for each truck have been split into more packages.

In figure 4.8 a table representation of a value can be seen. The size of the table cells are adjusted to the width and height of the table, and the game prefers that the table cells are square, but it is not always the case due to the limited space.

The bubble that appears when obtaining points can be seen on figure 4.9 flying from the given house to the points entity. Note that because the trucks in both games in the figure have been perfectly filled, they are on their way to the exit, and

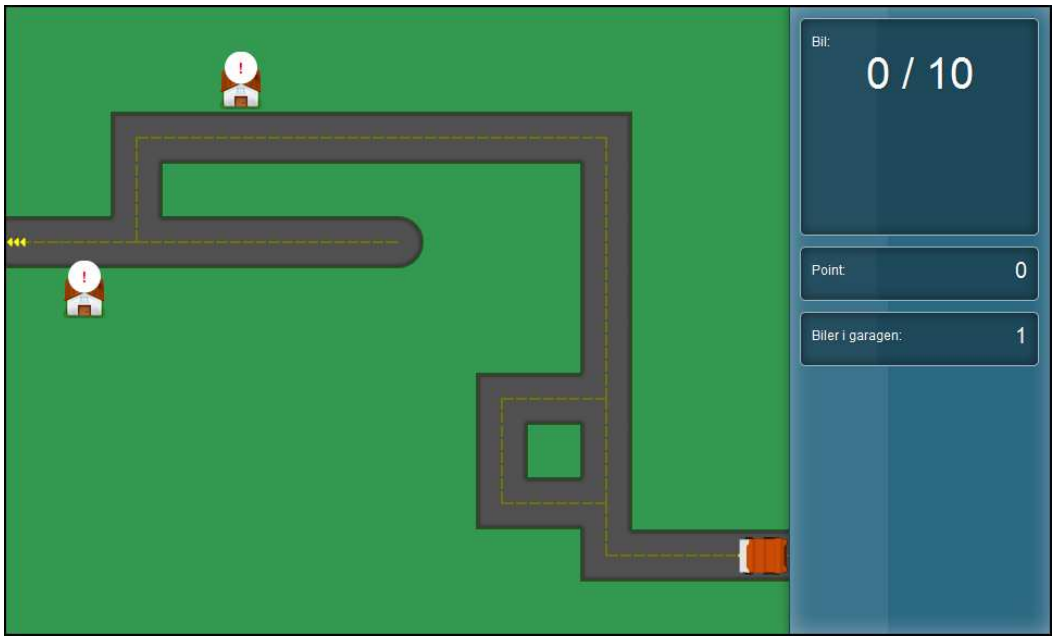



**Figure 4.6:** Screenshot of calibration level

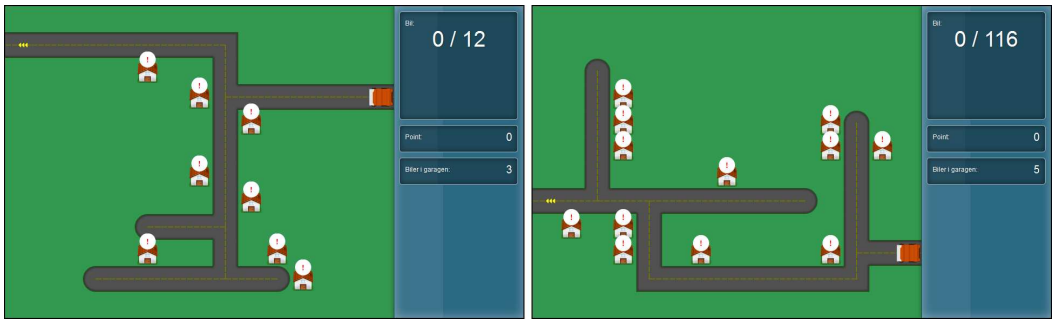



**Figure 4.7:** Screenshot of levels with higher difficulty

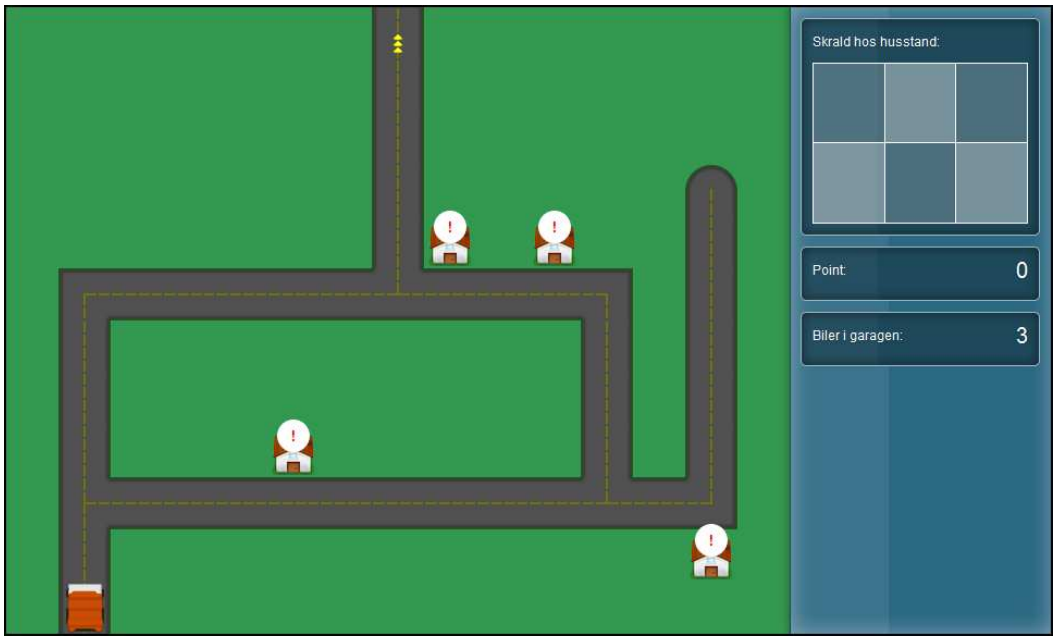



**Figure 4.8:** Screenshot of table representation

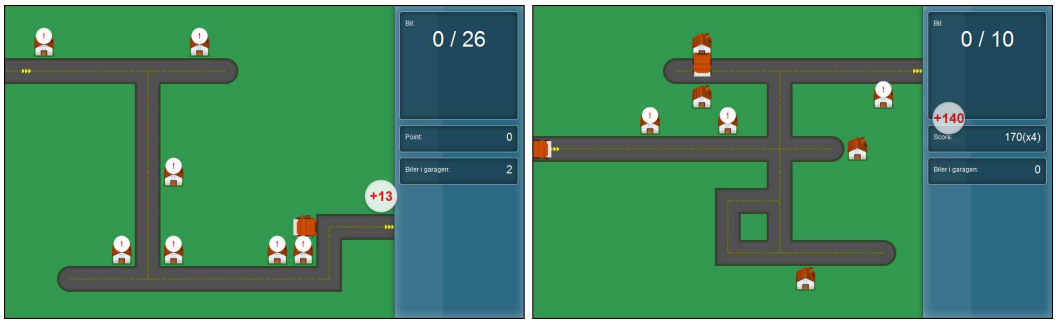



**Figure 4.9:** Screenshot of obtained points. Combo included to the right

a new truck is already driving in. This explains why it seems like the trucks have not been filled at all; the capacities refer to the new trucks.

## 4.5 Client and server-side refactoring

In a client-server relationship, certain jobs can be carried out by both and the choice rely on scalability and security. If a job is carried out by the client, it means that the server can use less resources and that it will not be hogged, as it potentially could be if it carried out the same job for all clients. However, without validation, the server cannot be sure that the client actually did the job as it should - in a game it means that cheating is possible. On the other hand, the server-side is often faster to carry out jobs and letting it do most of the heavy lifting means that all code is executed in one place and in one abstraction level.

The following in the game could be carried out by both the client and server:

- World generation
- Path finding
- Logging
- Database interaction
- Score calculation and tracking

If it were to be done by the client, everything should be validated before processing it further, which is a time taking task to develop properly. To avoid this, the jobs were all moved to the server. Path finding is not critical, as the validation could be as simple as to make a check between timestamps when the truck starts to move and arrives to its destination, however, some problems arose when trying to implement A* in a Crafty component, and it was decided to move this job to the server as well.

This could also be fixed by rewriting the client-side path finding so that it avoided or worked around the problem that were faced with Crafty, but it would most likely take more time.

The client is then in charge of rendering the data from the server and deciding when[3] to emit events to the server. This could be done because logging already was planned to use sockets which is very capable handling many requests, see section 4.3. Besides giving us the benefit of all data flowing through one system so that all data could be validated on the server-side, it also gathered all the game logic on the server increasing maintainability.

Moving all logic to the server did present another question: *How is performance affected when the world of all connected clients are stored in memory?* For the scope of this application this might not be a problem, but nonetheless it was something

[3]Not all click events is emitted. For instance, some checks are done to see if it is a valid event together with some throttling to prevent the same event from emitting too fast.

that has to be considered if the application should handle more than a couple of users at a time.

Redis[30] is a fast key-value store that could be used as a session store[4] to move the session data to another server but still keep the application fast. This would be a more expensive solution to the problem, but very scalable.

## 4.6 Database

As the database should be able to handle a large amount of data, it is crucial that all columns are of correct and optimal data types, e.g. id columns are either int or bigint and are unsigned. By comparison, unsigned int can take the maximum value $2^{32} - 1 (\approx 4.29 \cdot 10^9)$ while it for unsigned bigint is $2^{64} - 1 (\approx 18.45 \cdot 10^{18})$.

Figure 4.10 is a diagram of the database that has been implemented based on the analysis and design.

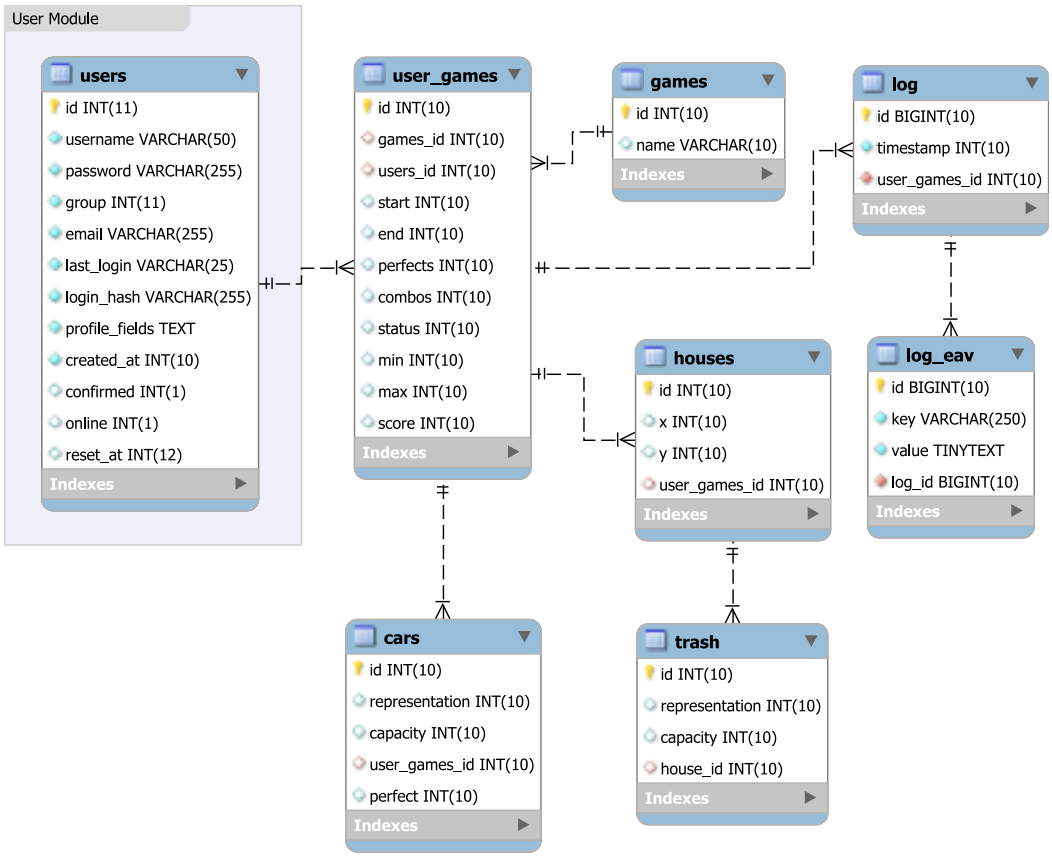



**Figure 4.10:** Database Diagram

Compared to the database design in section 3.3, this implementation is more simple. route, route_path and user_games_eav have been left out because they would take too much time to implement. Instead, the data that user_games_eav should contain will now be inserted in user_games. Because routes are not stored in the database,

[4] A place to store data about the current session

it will not be possible to watch a replay of a level.

All tables use InnoDB, because this engine make it possible to create SQL[5] relations. This means that if a table has a foreign key (i.e. a key that identifies a row in another table), one can create actions for what should happen to the rows in the current table, if the related row in the other table is updated or deleted.

These relations can be seen in the diagram. The relations and actions are stated by SQL in listing 4.10

```
1  ALTER TABLE `user_games`
2      ADD CONSTRAINT `fk_user_games_games`
3      FOREIGN KEY (`games_id` ) REFERENCES `games` (`id` )
4      ON DELETE CASCADE ON UPDATE NO ACTION;
5  ALTER TABLE `user_games`
6      ADD CONSTRAINT `fk_user_games_users1`
7      FOREIGN KEY (`users_id` ) REFERENCES `users` (`id` )
8      ON DELETE CASCADE ON UPDATE NO ACTION;
9  ALTER TABLE `cars`
10     ADD CONSTRAINT `fk_cars_user_games1`
11     FOREIGN KEY (`user_games_id` ) REFERENCES `user_games` (`id` )
12     ON DELETE CASCADE ON UPDATE NO ACTION;
13 ALTER TABLE `houses`
14   ADD CONSTRAINT `fk_houses_user_games1`
15     FOREIGN KEY (`user_games_id` ) REFERENCES `user_games` (`id` )
16     ON DELETE CASCADE ON UPDATE NO ACTION;
17 ALTER TABLE `trash`
18   ADD CONSTRAINT `fk_trash_houses1`
19     FOREIGN KEY (`house_id` ) REFERENCES `houses` (`id` )
20     ON DELETE CASCADE ON UPDATE NO ACTION
21 ALTER TABLE `log`
22   ADD CONSTRAINT `fk_log_user_games1`
23     FOREIGN KEY (`user_games_id` ) REFERENCES `user_games` (`id` )
24     ON DELETE CASCADE ON UPDATE NO ACTION;
25 ALTER TABLE `log_eav`
26   ADD CONSTRAINT `fk_log_eav_log1`
27     FOREIGN KEY (`log_id` ) REFERENCES `log` (`id` )
28     ON DELETE CASCADE ON UPDATE NO ACTION;
```

**Listing 4.10:** SQL relations and actions

### 4.6.1 Connection Pooling

At first the Node.js server was connected to the database with a single connection. This approached worked fine when one player was connected. When another player started playing, the single connection quickly proved to be insufficient. The entire system was slowed by the single connection.

Instead of opening one connection per user, the problem was approached differently. Utilizing a connection pool the application maintains a defined number of database connections.

The pool monitors the use of each individual connection and closes the connection if it has been idle for a defined amount of time. When a database connection is required, the pool is asked to return a connection to the application.

[5] Structured Query Language

If a connection is idle and thus available a connection is returned. If however, all active connections are in use, the pool opens a new connection to the database if it is allowed to or otherwise waits for a busy connection to free up before returning that.

Reusing connections with a pool removes some of the overhead of opening and closing database connections, both resource and time wise, whenever a query needs to be executed. This is a great both for the heavy logging and the many queries used when generating levels for many concurrent users, see section 4.6.3.

### 4.6.2 Queries

Because of the utilization of Node.js' asynchronous I/O and the use of Async[20] library a lot of things are being done in parallel, or somewhat close to parallel. This results in a demand of a few simultaneous database connections in the generation phase.

If the amount of database connections were to become an issue for the connection pool explained in section 4.6.1, MySQL's multiple row insert could be used; see listing 4.11.

```
1 INSERT INTO table (keys) VALUES (..),(..),(..)
```

**Listing 4.11:** MySQL multiple row insert

Switching to MySQLs multiple row insert would mean that some of the database work would have to be rewritten to cache the values that should be inserted and then execute the single INSERT query after all the data to be inserted has been processed.

Another way to optimize the queries is to re-use the ID of trucks and houses with the same attributes as the ones generated for the level to be inserted into the database.

```
1 $query = mysql_query("INSERT INTO table (keys, ...) VALUES (values,↩
      ...) ON DUPLICATE KEY UPDATE id = LAST_INSERT_ID(id);");
2 $result = mysql_insert_id();
3 $result2 = mysql_affected_rows();
```

**Listing 4.12:** MySQL detection of duplicate

Listing 4.12 shows how to achieve the above. After the INSERT query has been executed, $result will contain the ID if the truck or house of either the inserted object or the already existing one. $result2 will be an integer between 0 and 2. If the query did not change anything, the value is 0. Otherwise $result2 will be 1 if a new object was inserted or 2 if a duplicate was found, nothing was inserted and only the ID of the existing object was returned.

### 4.6.3 Scalability

For every click on a house, 3 log queries are executed. When the truck arrives at a house with trash, an additionally 4 INSERT queries are executed along with 1 UPDATE query.

For the first three calibration games, each game contains two trucks, two houses and two packages of trash. If one were to get all three games perfect, that is to fill all trucks to their exact capacity, the server would generate 42 log entries.

$$3 \cdot 2 \cdot (3 + 4) = 42 \tag{4.1}$$

At this point, it might not seem like a lot of log entries are being generated, but as the levels progress so does the logging.

As mentioned above, a log entry is generated for each click and for each trash pick up. That means the number of log entries can be roughly estimated with a few assumptions. The accumulated number of trucks generated after $n + 3$ games can be found by the formula given in equation 4.2 assuming that all trucks have been filled perfectly. $n$ notates the number of games after calibration such that $C_0$ is the accumulated number of trucks generated during calibration.

$$\begin{aligned} C_0 &= 6 \\ C_n &= C_{n-1} + \left\lceil \frac{1 + C_{n-1} + \left\lceil \frac{C_{n-1}}{3} \right\rceil}{3 + n} \right\rceil \end{aligned} \tag{4.2}$$

The accumulated number of trash packages for $n + 3$ games can be found by the formula in equation 4.3 where $T$ is the accumulated number of trash packages for $n + 3$ games. The trash spread method shown in equation 3.5 shows that the minimum number of packages a piece of trash can be split into is 1, while the maximum spread is 10 at worst case.

$$T = C \cdot \left[1; \frac{10 \cdot 1.2^{10}}{1.2^{10}}\right] = C \cdot [1; 10] \tag{4.3}$$

If a player was to do 10 perfect games this would result in 42 trucks following equation 4.2. Equation 4.4 can be used to find the number accumulated of trash packages the levels would have.

$$\begin{aligned} T &= 42 \cdot [1; 10] \\ &= [42; 420] \end{aligned} \tag{4.4}$$

The worst case scenario results in $420 \cdot 7 = 2,940$ log entries per user, for only the first ten games. If the game was used in a class of 20 pupils for just the first ten games, that would result in $20 \cdot 2,940 = 58,800$ log entries.

unsigned int:

$$\left\lfloor \frac{2^{32} - 1}{58,800} \right\rfloor = 73,043 \tag{4.5}$$

unsigned bigint:

$$\left\lfloor \frac{2^{64} - 1}{58,800} \right\rfloor = 313,720,137,307,985 \tag{4.6}$$

If unsigned integers were used for log id, 73,043 classes could play the first ten games as shown in equation 4.5, while using unsigned bigint would allow vastly more classes as seen in equation 4.6.

A popular game is often played a lot. If the pupils were to have 50 perfect games, that would result in the trash count interval [301; 3010] and the number of log entries in the interval [2107; 21070]. A class of 20 pupils results in $21,070 \cdot 20 = 421,400$.

Running the numbers again:

unsigned int:

$$\left\lfloor \frac{2^{32}-1}{421,400} \right\rfloor = 10,192 \tag{4.7}$$

unsigned bigint:

$$\left\lfloor \frac{2^{64}-1}{421,400} \right\rfloor = 43,774,902,880,184 \tag{4.8}$$

As seen in equations 4.7 and 4.8 using bigint for the log id is far more scalable than using int.

## 4.7 Speed optimizations

On the internet speed is everything[18]. Squeezing out every bit of performance is an important aspect of web application and even moreso for games.

### 4.7.1 JavaScript & CSS minification and combination

In order to reduce the number and size of requests in the front end, thus making the page load faster, a FuelPHP package named Casset[6] has been implemented. This package can minify and combine a selected number of JavaScript and CSS files and save the results on the webserver, which it only does, if the result files do not exist when a user visits the application. If they do, these files are fetched instead - the minification and combination execution is therefore only executed once. Benchmarks are shown in table 4.2 and 4.3.

**Table 4.2:** Minification and combination benchmarks **without** output compression

| | Raw | | Combined | | Combined + Minified | |
|---|---|---|---|---|---|---|
| **Files** | Size [KiB] | # files | Size [KiB] | # files | Size [KiB] | # files |
| **JavaScript** | 385.9 | 3 | 385 | 1 | 154 | 1 |
| **CSS** | 145.6 | 3 | 163 | 1 | 121 | 1 |

As seen when comparing tables 4.2 and 4.3 the size of the JavaScript assets went from 385.9KB to just 41.2KB, which is a size reduction of 89%.

[6] A class for managing assets

**Table 4.3:** Minification and combination benchmarks **with** output compression

| | Raw | | Combined | | Combined + Minified | |
|---|---|---|---|---|---|---|
| **Files** | Size [KiB] | # files | Size [KiB] | # files | Size [KiB] | # files |
| **JavaScript** | 91.2 | 3 | 90.1 | 1 | 41.2 | 1 |
| **CSS** | 21.6 | 3 | 20.7 | 1 | 19.3 | 1 |

## 4.8 Platform Compatibility

The use of PHP, Node.js and MySQL allows the execution of the application on almost all platforms, because the technologies used are open source and available on most platforms.

### 4.8.1 Server-side

**Web server** Needed in order to run the application.

**PHP** The web server should support PHP version 5.3+.

**MySQL** Support for the engine InnoDB is required. The application has been tested with MySQL 5.1+, which supports this.

**Node.js** Required to run the game server.

### 4.8.2 Client-side

**JavaScript** Required for the game engine to work.

**Modern browser** Needed for CSS3 animations, CSS3 transitions and WebSocket.

# 5 Results

When the game was being implemented, some choices and limitations had to be made to meet the deadline. Some were good while others can be improved, and some even revealed errors. The game was tested by a group of pupils which gave important information and lead to new ideas.

## 5.1 Current status and limitations

The application has been developed with flexibility and extensibility in mind. It is only required to change two areas of the code base to add a new representation; the controller that selects a representation for an object and the renderer that is in control of how each representation is displayed in the graphical user interface.

The application structure is relatively modular and allows for new game modes for the current game or entirely new games. New game modes for the implemented game could be to increase difficulty in mathematics by implementing differential and integral equations, cosine and sinus, degree and radian conversion etc.

During the development of the game, a few assumptions have been made, and have resulted in the following restrictions and limitations by design:

- All users have a limit on the size of the username on 50 characters.
- Also, all passwords **has** to be at least 4 characters long.
- Trucks do not show representations, even though representations are assigned to them.
- Time is not used, but is being logged.
- Authentication not used in-game (see the list of known issues in section 5.1.1).

### 5.1.1 Known issues

Due to lack of time, some issues were not resolved. Most are low priority bugs and will not be an immediate problem for the player, but there is one fatal error of high priority. The list is sorted by severity.

1. Too many frequent connections will make Node.js crash. This is caused by a bug in the A* implementation for Node.js' path finding.

2. The game server does not authenticate a request. It is assumed a valid user id is given upon connection. This could be fixed by moving the user module to Node.js.

3. Big values can be unreadable in the graphical user interface if represented by a table. This can be solved by colouring some table cells that represent 10's or 100's. Alternatively, other representations than tables could be used.

4. When the difficulty is very hard, the player sometimes get easy games with low values (as low as 1% of the average values in the game just finished).

5. When a truck has collected trash from a house that has two or more neighbouring road tiles, it might drive out on another tile than the one it drove in from, making it look like it moves through the house. This can be fixed by forcing the truck to drive out on the same road tile it drove into the house from.

6. The generation of road network allows an unlimited amount of intersections. With 32 generated trucks, the probability for an intersection is 1.2. An upper limit should be set to avoid this.

7. The generation of trucks has no upper limit, so at one point there will be more houses than there is space for. This can be fixed by calculating the maximum amount of houses a game can have and make this the upper limit; each truck would then have to collect one trash package, i.e. they would not be spread.

### 5.1.2 Reflections on implementation

While the game has been implemented satisfactory and acts like described in game design with only a few known issues and alterations, moving all server-side functionality to Node.js would be a good idea. As already mentioned, during the implementation more and more tasks were moved from PHP to Node.js because of its superior performance. Even though Node.js has been designed to run in a single thread, the Cluster module[2] can be used to take advantage of multi-core systems by launching a cluster of Node.js processes in different threads which will improve scalability even more. Moving all server-side functionality to one platform would also increase maintainability and reduce latency. Because of the extensive experience with PHP and FuelPHP and little-to-nothing with Node.js at the beginning of this project, it is doubtful that the game and application would have got to the same status if only the latter was used. If only using Node.js server-side, user management could be implemented with the Passport module[11] while the web application and static files could be served using Express[12].

The log data is saved in a EAV-model in the MySQL database, which is highly flexible, but can lead to an anti-pattern and more complex and heavy queries compared to a relationally-modelled data schema. If not using EAV, one would have to create tables for each log type, and this could quickly add up. A better approach would

be to use a NoSQL database such as MongoDB, which gives the same flexibility as the EAV-model but better query performance.

It is doubtful that using a canvas element for the graphical user interface would increase performance, and because the graphics have worked flawlessly on every computer tested, some more than five years old, using HTML5 and CSS3 seemed not to be a problem. Refactoring functionality between client-side and server-side is a never-ending topic and the needs will depend on scalability and the hardware on the server as well as the deemed hardware on the clients.

While the educational subject of the game is addition and multiplication, it has been developed with extensibility in mind as described in section 5.1. Spelling could also be added for instance by having the truck carry words such as 'a', 'an' and 'the' with the goal of pairing it with a house that contained a noun. Even learning new languages could also be an interesting twist, by translating word by word or even sentences. These ideas for other interactive games are easily supported by the database. This way, the back end could act like a shared platform for games used for education.

## 5.2 Usability tests

Monday 10th of June 2013, the game was played by pupils from three different classes at Skolen ved Bülowsvej, Frederiksberg, Denmark, which made it possible to study both quantitative and qualitative how the game was experienced and played. It were second grade, third grade and fourth grade pupils, which represent the mid and upper part of the target audience of the game.

Each test was started with a very short introduction to the game, but the rules and concept were not told. This meant, for all classes, that some pupils needed help understanding the game while others figured it out right away.

### 5.2.1 Technical Environment

The application was tested with two setups. Initially, the application was hosted on Amazon's AWS[1] using EC2[2] with 615 MiB of memory and one virtual core. For the database RDS[3] was used with 630 MiB memory, both on Amazon's free tier.

Because of issues with connectivity for the users, the setup was switched to a private server. This provided 6 physical cores at 3.2 GHz bursting up to 3.6 GHz and 16 GiB dedicated RAM[4]. This setup decreased the connectivity issues.

All pupils used Google Chrome version 27.0.1453.110 and the internet connection was measured to 30+/20+ Mbit/s.

[1] Amazon Web Services
[2] Amazon Elastic Compute Cloud
[3] Amazon Relational Database Service
[4] Random Access Memory

### 5.2.2 Third grade pupils

The first class to test the game was a third grade class. This was the only test carried out with the AWS setup.

The following were experienced by one or more players:

- Trouble in understanding the mouse gestures
- Not knowing how to drive out the truck if it is not filled and there is no more trash (i.e. other trucks have been filled improperly)
- The game was too easy
- Progressed in harder difficulty fast
- Waited patiently during the AWS server problems instead of playing another game also being tested that day

Most of the pupils that figured out the game on their own also started comparing their score to their neighbour. Due to the nature of the adaptive difficulty, section 3.2.4.3, and the point system, section 3.2.4.5, directly comparing scores does in no way show how they perform compared to each other.

This problem of competitive behaviour was discussed and one solution to the problem was to convert the score to some kind of currency that could be used to increase the longevity of the game by allowing customization to the game.

To embrace the competitive behaviour of the pupils, to keep them playing the game, a form of *score index* that indicated their score relative to a max score for their level and factor in their own progression level could be implemented. The pupils would then compete with each other by how much they have learned, such that even the ones that are not good at mathematics can be in the competition.

If such a score index was implemented a fun way to inspire the players could be to add live updates, so that each time a pupil in the class completes a game, all those in that class will receive an instant update saying that *X just completed a game with score index Y*. Such live updates could even inspire the players even more.

The points for each game could be converted to credits, which the user could use to buy cosmetics or functionality. Credits coupled with a score index seems like a good choice to increase the game's longevity and the pupils' interest in the game.

### 5.2.3 Second grade pupils

After the setup was moved to the private server, the tests went better, but some computers were falling back to long polling when playing the game, while other computers at the same time could play the game just fine. The computers that fell back to long polling would show a black screen.

The following were experienced by one or more players:

- Trouble in understanding the representation display - that both truck and trash were showed in the same spot
- Trouble in understanding the "current fill / capacity" display of the truck
- Many wanted help from the start
- One half meant the game was too easy, the other half that it was too hard
- Questioned whether the game could be played after school hours
- Trouble in understanding how to make the truck drive

### 5.2.4 Fourth grade pupils

During this test there were no server problems at all. There were two pupils for each computer.

The following were experienced by one or more players:

- Kept clicking at the same house even though the truck was moving for it
- Pressed keys corresponding to the values in a trash package as if it would do something

One of the pupils mentioned that he missed some more interactivity and suggested that when clicking on a house to instruct the truck to make a pickup, one should first solve the representation. If the trash for instance was given as an equation $x \cdot y$, he proposed that he should type $xy$ before the truck would pick up the trash.

### 5.2.5 Reflection on obtained data

In the usability tests carried out in the three classes, more than 200 games were completed which yielded more than 3200 entries in the log_eav table of the database. This quantitative data can be used for statistics, which can be reflected on together with the qualitative assessments made in each class. The statistics found in this section are based entirely on some of the before mentioned log data and are made with the Log Analyser tool found in appendix D.

The qualitative assessments from the usability tests showed that the game was not intuitive for all pupils, and that it was too easy for those who understood it right away. Our own assessment before the tests was that the difficulty engine might have been made to increase the levels too fast.

Figure 5.1 shows how the three classes performed averagely. Some pupils played more than six games, but this was the number of games that all pupils got to play. The qualitative assessment of the pupils showed that many found the game to be too easy, the figure shows the same – an average correctness percentage of 80-90% is very good – however, six games is not enough to conclude whether the difficulty engine should be tweaked, but combined with the qualitative assessment, it is sufficient to conclude that the starting difficulty ought to be at a higher level;

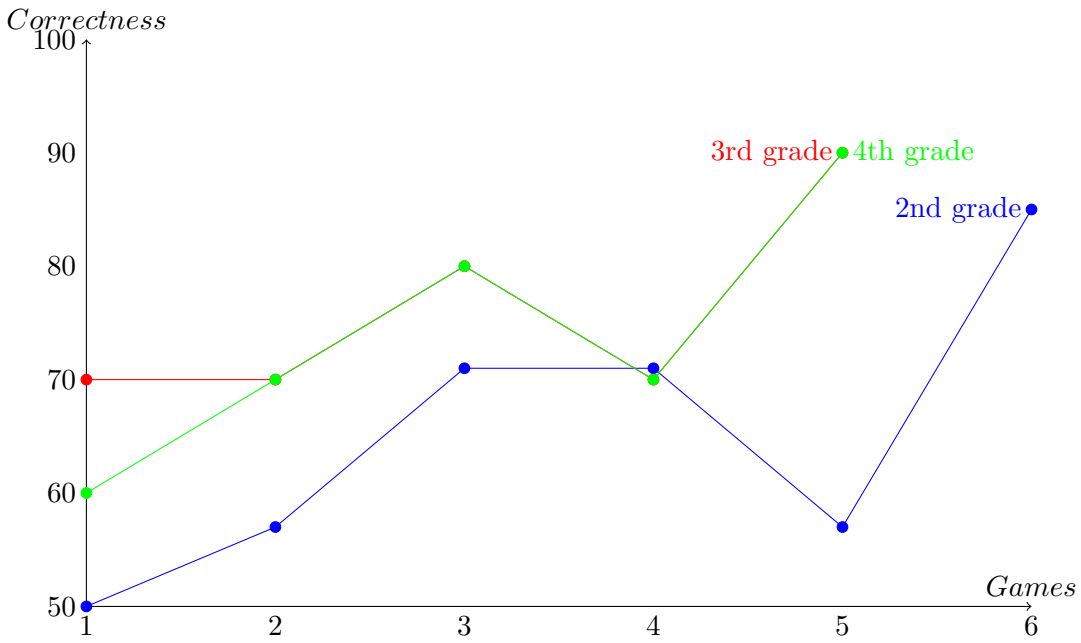



**Figure 5.1:** Correctness percentage for games completed by the entire classes

it could be modified by setting different ranges for each age or class – or letting the teacher do it.

Figures 5.2, 5.3 and 5.4 show that the difficulty engine works by generating trucks and therefore packages of trash based on the player's performance. The number range is not visible on the figures, but this adapts as well. In figure 5.2 it can be seen that the 6th game has an increased level of difficulty, where the player fills no truck perfectly. In the 7th game, the level of difficulty is therefore decreased again.

Figure 5.3 shows that the player has some difficulties in the first few games, but catches up – perhaps because we or another pupil told how to play the game. It is again seen that the number of trucks is decreased whenever the performance is not (near) perfect and increased when it is.

Figure 5.4 shows a player from the 2nd grade class that performs perfectly, i.e. solves all mathematical problems correctly. The player pointed out that he or she did not find the game difficult at all, but quite enjoyable. When asked if he or she had noticed an increase in the number of houses and trucks and a higher number range, this was confirmed, but it was not enough to challenge the skill of this player.

In figure 5.5 it is indicated that the player either misses the point of the game or that he or she just plays with no desire in solving the mathematical problems correctly. After playing 22 games by clicking on what seems to be totally random houses, something happens and the rest of the games are played perfectly. It could be us, a teacher or other pupils telling the player how the mathematical problems ought to be solved.

In the beginning, the pupils generally had difficulties in understanding the game concept, which indicates that the gameplay is not fully intuitive. A short introduction to the rules could be shown, but the pupils might skip it if they are able

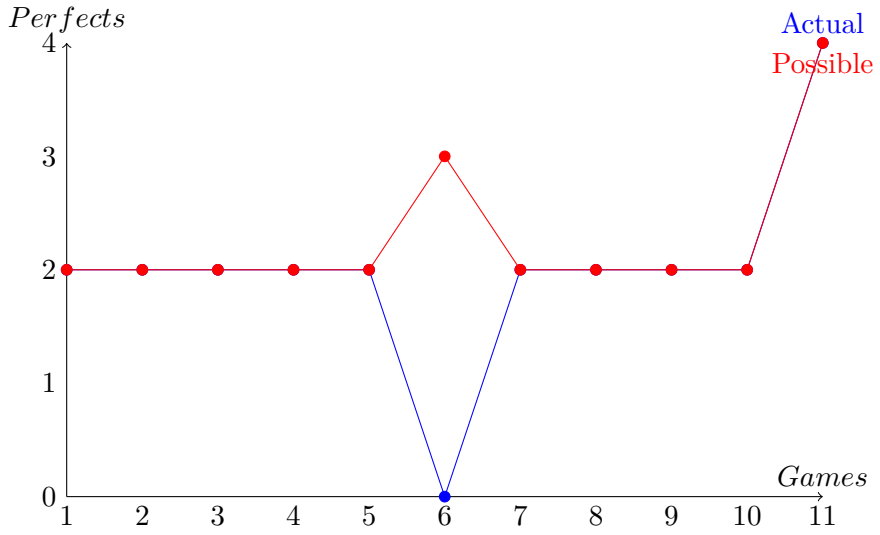



**Figure 5.2:** A player from 2nd grade experiencing adapting difficulty

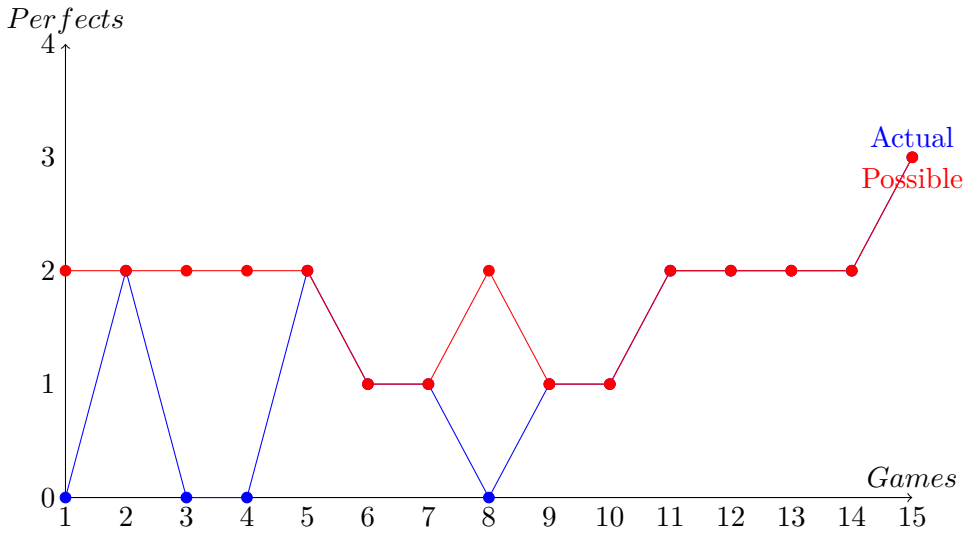



**Figure 5.3:** A player from 4th grade experiencing adapting difficulty

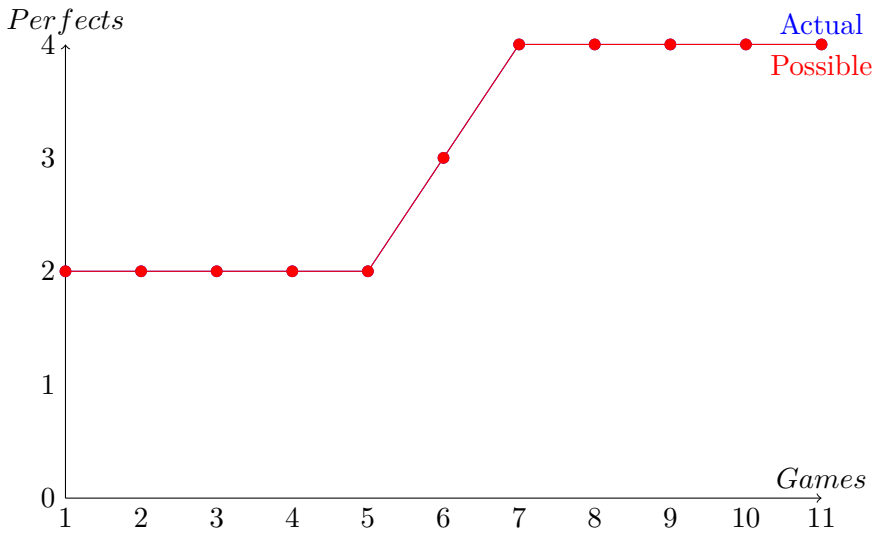



**Figure 5.4:** A player from 2nd grade solving all games perfectly

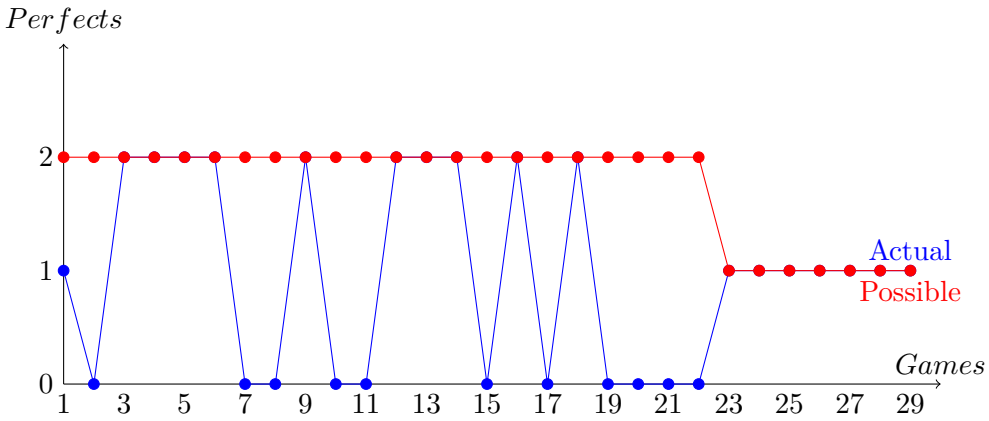



**Figure 5.5:** A player from 2nd grade possibly clicking on random houses

to – the rules are already present in the application and will be displayed with the click of a button. While the pupils in general found the played levels very easy, they found it enjoyable and applauded the combination of play and teaching. Some pupils wanted more interactivity in the game or wanted secondary objectives while the truck was driving. Clearly this shows that either click gestures are too little interactivity, or that the truck drives too slowly – or both.

## 5.3 Future work

Looking at the results from the usability tests and the implementation itself, there is clearly room for improvements. Besides fixing the known errors and use different technologies for parts of the game, as described in section 5.1, the game and application show potential for increasing the user and learning experience if some features are added or changed.

### 5.3.1 Improved difficulty engine

The difficulty engine could be tweaked to better and faster adapt to the skill level of each player. Various solutions exist, such as:

- Increase the difficulty faster in levels of lower difficulty
- Make assumptions on the players' skills based on specific actions
- Use time as a parameter
- Use age or class as a parameter

### 5.3.2 Improved User Interface

It could be clearer to the player when a combo is obtained and how big it is. A horizontal or vertical bar that in which small parts will be coloured whenever the combo increases could be used. In order for the pupil to get more attached to the content in the game, the city could have a name from a real city in Denmark. Preferably from a list containing the city names near the player.

Sounds have not been discussed or used in the game, but sound can do as much impression on the player as the visuals. Crafty support audio through HTML5, and playing sounds for the truck movement, trash pick up and combo increase would be a good idea.

The first level could act like a tutorial for how to play the game. This would introduce the concept more actively than just displaying the rules as a text somewhere in the application.

### 5.3.3 Administrative Performance Monitoring

Expanding the ACL[5] for teacher users by giving them control over their respective classes and pupils would be a good idea. This way the teacher would be able to get statistics of how a pupil or class performs. The teacher would then be able to monitor how a player progresses, which gives the teacher a good reason to include the game in the lessons.

### 5.3.4 Improved Motivation

As previously described in section 2.2.6.2, achievements are a great motivator. As the game engine is already logging a lot of data while the game is being played, implementing achievements should be an easy task that could greatly benefit the game in the form of increased motivation for the players.

Moreover, the game could indicate whenever the player performs well by showing encouraging messages and playing specific sounds.

### 5.3.5 Improved Score System

An algorithm such as B*[1] could be used to calculate the best score possible for a game. The final score given to the player could then be based on the performance relative the best possible score in that level.

Because the level of difficulty adapts to each player's skill, the score system could also be more global, such that players do not compare their score on a level basis, but rather on an overall basis. This could be achieved by introducing some sort of wallet, where points obtained in each level would be converted to some credits and added to the wallet. Such credits could then be used to unlock new features or purchase things for the game.

### 5.3.6 Educational Gaming Platform

As mentioned in section 5.1.2, the application could be used as a platform containing different games for teaching. Schools and teachers would then have a single centralized place to visit when wanting to include digital solutions in the teaching material.

[5] Access Control List

# 6 Conclusion

An interactive game for teaching elementary mathematics has been engineered successfully. The game is about filling garbage trucks with trash from houses such that the trucks neither end up with too little or too much trash. Only mouse-gestures are needed in the game, and addition and mulitplication problems are displayed to the player with different representations, namely basic multiplication equations, areas of tables and numbers. The player is rewarded with points and bonus points on correct solutions, and the level of difficulty will adapt to each individual's skills by examining log data about the player's performance from previous levels.

The implementation of the game is browser based and has been developed with PHP, Node.js, WebSocket, MySQL, JavaScript, HTML and CSS, but the architecture can be simplified by moving the PHP implementation to Node.js, thus running the server entirely on the latter. MongoDB seems more ideal for a database solution because of its storage pattern similar to the EAV model used for log data.

Usability tests on second to fourth grade pupils showed that the pupils could identify the mathematical problems presented to them in the game. The tests also showed that the game design was not completely intuitive, but once the pupils got the hang of it, it was an enjoyable experience.

# References


[1] B*, June 2013. http://en.wikipedia.org/wiki/B*.

[2] Cluster node.js v0.10.12 manual & documentation, June 2013. http://nodejs.org/api/cluster.html.

[3] Screen resolution statistics, June 2013. http://www.screenresolution.org/.

[4] 10gen Inc. Mongodb, June 2013. http://www.mongodb.org/.

[5] Batiste Bieler. Sprite.js framework, June 2013. https://github.com/batiste/sprite.js/.

[6] Tom S. Chan and Terence C. Ahem. Targeting motivation – adapting flow theory to instructional design. *Journal of Educational Computing Research*, 21(2):151 – 163, 1999.

[7] Crafty. Crafty - javascript game engine, html5 game engine, June 2013. http://craftyjs.com/.

[8] Ubiquitous Entertainment Inc. / enchant.js Inc. enchant.js - a simple javascript framework for creating games and apps, June 2013. http://enchantjs.com/.

[9] Mary Jo Foley. A new year, a new microsoft roadmap: Stepping up the delivery pace, January 2013. http://www.zdnet.com/a-new-year-a-new-microsoft-roadmap-stepping-up-the-delivery-pace-7000009402/.

[10] FuelPHP. *FuelPHP documentation*, 3 2013. http://docs.fuelphp.com.

[11] Jared Hanson. Passport - simple, unobtrusive authentication for node.js., June 2013. http://passportjs.org/.

[12] TJ Holowaychuk. Express - node.js web application framework, June 2013. http://expressjs.com/.

[13] Andy Ide. Php just grows & grows, January 2013. http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html.

[14] Activision Publishing Inc. Guitar hero, June 2013. http://guitarhero.com/.

[15] Electronic Arts Inc. Simcity official website, June 2013. http://www.simcity.com/.

[16] jQuery. *jQuery documentation*, 3 2013. http://docs.jquery.com.

[17] AL Kalet, HS Song, U Sarpel, R Schwartz, J Brenner, TK Ark, and J Plass. Just enough, but not too much interactivity leads to better clinical skills performance after a computer assisted learning module. *Medical Teacher*, 34(10):833–839, 2012.

[18] Steve Lohr. For impatient web users, an eye blink is just too long to wait, February 2012. http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html.

[19] Adobe Systems Software Ireland Ltd. Flash to html5, June 2013. http://www.adobe.com/dk/products/flash/flash-to-html5.html.

[20] Caolan McMahon. Async utilities for node and the browser, June 2013. https://github.com/caolan/async.

[21] Microsoft. Microsoft support lifecycle, June 2013. http://support.microsoft.com/lifecycle/?LN=en-us&c2=12905.

[22] Mozilla Developer Network. Using css animations, June 2013. https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_animations.

[23] Donald A. Norman. Things that make us smart: Defending human attributes in the age of the machine. page 38, 1993. ISBN 978-0201626957.

[24] Ministry of Children and Education. Digital solutions strengthen the education of all pupils. Press release, 06 2013. http://eng.uvm.dk/News/~/UVM-EN/Content/News/Eng/2013/130405-Digital-solutions-strengthen-the-education-of-all-pupils.

[25] Ministry of Children and Education. Gør en god skole bedre - et fagligt løft af folkeskolen. PDF, 06 2013. ISBN 978-87-92985-06-4. http://www.uvm.dk/I-fokus/Goer-en-god-skole-bedre/Laes-udspillet-Goer-en-god-skole-bedre.

[26] Amit Patel. Heuristics, June 2013. http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html.

[27] Q-Success. Usage statistics and market share of node.js for websites, June 2013. http://w3techs.com/technologies/details/ws-nodejs/all/all.

[28] Siddharth Rao. Css3 vs jquery animations, June 2013. http://dev.opera.com/articles/view/css3-vs-jquery-animations/.

[29] Charles Roe. Acid vs. base: The shifting ph of database transaction processing, June 2013. http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/.

[30] Salvatore Sanfilippo. Redis, June 2013. http://redis.io/.

[31] solid IT. Db-engines ranking - popularity ranking of database management systems, June 2013. http://db-engines.com/en/ranking.

[32] StatCounter. Screen resolution alert for web developers, June 2013. http://gs.statcounter.com/press/screen-resolution-alert-for-web-developers.

[33] Microsoft Studios. Age of empires, June 2013. http://www.ageofempires.com/.

[34] Inc. Take-Two Interactive Software. Sid meier's civilization, June 2013. http://www.civilization.com/.

[35] Unity Technologies. Unity - game engine, June 2013. http://unity3d.com/.

[36] Danny Winokur. Flash to focus on pc browsing and mobile apps; adobe to more aggressively contribute to html5, November 2011. http://blogs.adobe.com/conversations/2011/11/flash-focus.html.

# A

# Use cases

There is one type of actor in the game:

- Player: User logged in

## A.1 Moving cursor over and out of house

**Actor:** Player

**Scenario:**

- Starts a new game
- Moving cursor over a house with a given trash size
- Trash size displayed to the player in its representation
- Moving cursor out of said house
- Trash size hidden to the player

**Alternative scenarios:**

1. Clicked house has no trash
    - Nothing happens

## A.2 Clicking on a house

**Actor:** Player

**Scenario:**

- Starts a new game
- Clicks on a house with a trash size matching current truck capacity
- Truck drives to house and collects the trash

- Full points and a combo bonus are given to player
- Truck drives to exit

**Alternative scenarios:**

1. Clicked house has too much trash compared to current truck capacity
    - Truck drives to house and collects the trash
    - Points subtracted a penalty are given to player
    - Truck drives to exit
2. Clicked house has too little trash compared to current truck capacity
    - Truck drives to house and collects the trash
    - Points subtracted a penalty are given to player
3. Clicked house has no trash
    - Nothing happens

## A.3 Changing path of a moving truck

**Actor:** Player

**Scenario:**

- Starts a new game
- Clicks on a house with trash
- Truck drives to house
- Clicks on another house with trash before truck reaches first house
- Path is changed and truck drives to that house instead

**Alternative scenarios:**

1. Truck reaches first house before second click
    - A new path is given when clicking on another house

## A.4 Clicking on exit

**Actor:** Player

**Scenario:**

- Starts a new game

- Clicks on exit tile
- Truck drives to exit
- Next truck in the garage drives in

**Alternative scenarios:**

1. There are no trucks in the garage
    - Game ends

# B

# WebSocket and AJAX timings

```
1  var app = require('express')()
2  , server = require('http').createServer(app)
3  , io = require('socket.io').listen(server);
4
5  server.listen(1337);
6
7  // io.configure(function (){
8  //   io.set('log level', 1);
9  // });
10
11 app.get('/', function (req, res) {
12     res.sendfile(__dirname + '/index.html');
13 });
14 app.post('/ajax', function (req, res) {
15     res.writeHead(200, {'Content-Type' : 'text/plain'});
16     res.end();
17 });
18
19 io.sockets.on('connection', function (client) {
20
21     _events = [];
22
23     client.on('dom ready', function() {
24         client.emit('start');
25     })
26
27     // socket.emit('pickup trash', {id: getCurrentCar().id, x: ↩
       getCurrentCar()._posX, y: getCurrentCar()._posY}, {}, true);
28     // socket.emit('pickup trash', {id: this_car.id, x: this_car.↩
       _posX, y: this_car._posY}, trash.id);
29     client.on('pickup trash 1', function (var1, var2, var3) {
30         if (!_events['pickup trash 1']) _events['pickup trash 1'] =↩
               0;
31         // Thank you!
32         _events['pickup trash 1']++;
33         console.log("Got pickup trash " + _events)
34     });
35     client.on('pickup trash 2', function (var1, var2, var3) {
36         if (!_events['pickup trash 2']) _events['pickup trash 2'] =↩
               0;
37         // Thank you!
38         _events['pickup trash 2']++;
39         console.log("Got pickup trash " + _events)
40     });
41
```

```
42      // socket.emit('house click', {id: getCurrentCar().id, x: x, y:
            y}, {x: this._posX, y: this._posY});
43      client.on('house click', function (var1, var2) {
44          console.log("Got house click")
45          if (!_events['house click']) _events['house click'] = 0;
46          // Thank you!
47          _events['house click']++;
48      });
49
50      client.on('_events', function(name) {
51          console.log("Got _events")
52          client.emit('_events', _events[name], name);
53          _events[name] = 0;
54
55          client.emit('start', name);
56      })
57  });
```

**Listing B.1:** app.js

```
1   <html>
2   <body>
3   <script src="http://localhost:1337/socket.io/socket.io.js"></script
        >
4   <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/
        jquery.min.js"></script>
5   <script>
6
7   var count = 1000;
8   var repeats = 10;
9
10  var times = [];
11  var results = {
12    socket: [],
13    ajax: []
14  };
15
16  var socket = io.connect(document.location.href);
17
18  socket.on('_events', function(count, name) {
19    var time = new Date() - times[name];
20
21    if (!results.socket[name]) results.socket[name] = [];
22    results.socket[name].push(time);
23
24    console.log("["+name+"] " + count + " in " + time + " ms");
25    console.log("["+name+"] " + (count/time).toFixed(3) + " req/ms");
26    console.log("["+name+"] " + ((count/time)*1000).toFixed(3) + "
          req/sec");
27  });
28
29  var requests = 0;
30
31  socket.on('start', function(name) {
32    if (!name) {
33      console.log('start 1st socket');
34      // socket.emit('pickup trash', {id: getCurrentCar().id, x:
            getCurrentCar()._posX, y: getCurrentCar()._posY}, {}, true);
35      times['pickup trash 1'] = new Date();
36      for (var i = 0; i < count; i++) {
37        socket.emit('pickup trash 1', {id: i, x: i, y: i}, {}, true);
```

```
            // LENGTH: 67, i < 10
    };
    socket.emit('_events', 'pickup trash 1');
  } else if (name == 'pickup trash 1') {
    console.log('start 2nd socket');
    // socket.emit('pickup trash', {id: this_car.id, x: this_car._posX, y: this_car._posY}, trash.id);
    times['pickup trash 2'] = new Date();
    for (var i = 0; i < count; i++) {
      socket.emit('pickup trash 2', {id: i, x: i, y: i}, i); // LENGTH: 61, i < 10
    };
    socket.emit('_events', 'pickup trash 2');
  } else if (name == 'pickup trash 2') {
    console.log('start 3rd socket');
    // socket.emit('house click', {id: getCurrentCar().id, x: x, y: y}, {x: this._posX, y: this._posY});
    times['house click'] = new Date();
    for (var i = 0; i < count; i++) {
      socket.emit('house click', {id: i, x: i, y: i}, {x: i, y: i}); // LENGTH: 70, i < 10
    };
    socket.emit('_events', 'house click');
  } else if (name == 'house click') {
    requests++;
    if (requests == repeats) {
      // And the same for AJAX
      requests = 0;
      setTimeout(function() {
        a1();
      }, 2000); // Wait 2 sec
    } else {
      // Start another round
      setTimeout(function() {
        socket.emit('dom ready');
      }, 1000); // Wait 2 sec
    }
  }
})

var c1 = 0, c2 = 0, c3 = 0;
function a1() {
  console.log("Start 1st ajax");
  c1 = 0;
  for (var i = 0; i < count; i++) {
    var start_time = new Date();
    var time = 0;
    $.ajax({
        type: "POST",
        url: "/ajax",
        data: { first: {id: i, x: i, y: i}, second: {}, third: true },
        timeout : 10000,
        success : function (data) {
            var end_time = new Date();
            time += end_time - start_time;
            c1++;
            if (c1 == count) {
              console.log("[AJAX][pickup trash 1] " + count + " in " + time + " ms");
```

```
91                console.log("[AJAX][pickup trash 1] " + (count/time).↩
                      toFixed(3) + " req/ms");
92                console.log("[AJAX][pickup trash 1] " + ((count/time)↩
                      *1000).toFixed(3) + " req/sec");
93
94                if (!results['ajax']['pickup trash 1']) results['ajax↩
                      ']['pickup trash 1'] = [];
95                results['ajax']['pickup trash 1'].push(time);
96
97                // Fire next ajax test
98                a2();
99              }
100         },
101         error : function (jqXHR, textStatus, errorThrown) {
102             console.log('Error: ' + textStatus + " " + errorThrown)↩
                    ;
103         }
104     });
105   };
106 }
107
108 function a2() {
109   console.log("Start 2nd ajax");
110   c2 = 0;
111   for (var i = 0; i < count; i++) {
112     var start_time = new Date();
113     var time = 0;
114     $.ajax({
115         type: "POST",
116         url: "/ajax",
117         data: { first: {id: i, x: i, y: i}, second: i },
118         timeout : 10000,
119         success : function (data) {
120             var end_time = new Date();
121             time += end_time - start_time;
122             c2++;
123             if (c2 == count) {
124               console.log("[AJAX][pickup trash 2] " + count + " in ↩
                      " + time + " ms");
125               console.log("[AJAX][pickup trash 2] " + (count/time).↩
                      toFixed(3) + " req/ms");
126               console.log("[AJAX][pickup trash 2] " + ((count/time)↩
                      *1000).toFixed(3) + " req/sec");
127
128               if (!results['ajax']['pickup trash 2']) results['ajax↩
                      ']['pickup trash 2'] = [];
129               results['ajax']['pickup trash 2'].push(time);
130
131               // Fire next ajax test
132               a3();
133             }
134         },
135         error : function (jqXHR, textStatus, errorThrown) {
136             console.log('Error: ' + textStatus + " " + errorThrown)↩
                    ;
137         }
138     });
139   };
140 }
141
```

```
142 function a3() {
143   console.log("Start 3rd ajax");
144   c3 = 0;
145   for (var i = 0; i < count; i++) {
146     var start_time = new Date();
147     var time = 0;
148     $.ajax({
149         type: "POST",
150         url: "/ajax",
151         data: { first: {id: i, x: i, y: i}, second: {x: i, y: i} },
152         timeout : 10000,
153         success : function (data) {
154             var end_time = new Date();
155             time += end_time - start_time;
156             c3++;
157             if (c3 == count) {
158               console.log("[AJAX][house click] " + count + " in " +↩
                      time + " ms");
159               console.log("[AJAX][house click] " + (count/time).↩
                      toFixed(3) + " req/ms");
160               console.log("[AJAX][house click] " + ((count/time)↩
                      *1000).toFixed(3) + " req/sec");
161
162               if (!results['ajax']['house click']) results['ajax'][↩
                      'house click'] = [];
163               results['ajax']['house click'].push(time);
164
165               requests++;
166               if (requests == repeats) {
167                 done();
168               } else {
169                 setTimeout(function() {
170                   a1();
171                 }, 1000); // Wait 2 sec
172               }
173             }
174         },
175         error : function (jqXHR, textStatus, errorThrown) {
176             console.log('Error: ' + textStatus + " " + errorThrown)↩
                    ;
177         }
178     });
179   };
180 }
181
182 $(document).ready(function() {
183   socket.emit('dom ready');
184 });
185
186 function done() {
187   console.log("Parsing..")
188   var types = ['pickup trash 1','pickup trash 2','house click'];
189   $s = 'type,size,' + types.join(',') + "\n";
190   $s += 'socket,'+count+',';
191   for (var i = 0; i < types.length; i++) {
192     $s += avg(results.socket[types[i]]) + ',';
193   };
194   // Remove last ,
195   $s = $s.substring(0, $s.length - 1) + "<br>\n";
196
```

```
197   $s += "ajax,"+count+',';
198   for (var i = 0; i < types.length; i++) {
199     $s += avg(results.ajax[types[i]]) + ',';
200   };
201   // Remove last ,
202   $s = $s.substring(0, $s.length - 1);
203   $s += "\n";
204   document.write($s.replace("\n","<br>"));
205
206 }
207
208 function avg(times) {
209   var sum = times.reduce(function(a, b) { return a + b });
210   var avg = sum / times.length;
211   return avg;
212 }
213
214 </script>
215   <div id="users">Hello!</div>
216 <div style="float:left;height:250px;overflow:scroll-y;padding:10px;↩
      ">
217   <div id="chat"></div>
218 </div>
```

**Listing B.2:** index.html

# C

# World generator

```
1  <?php
2
3  namespace World;
4
5  class World {
6      private $grid = array();
7      private $intersections = array();
8      private $intersection_coords = array();
9      private $houses = array();
10
11     private $intersection_pct = 0;
12
13     private $start = array();
14     private $end = array();
15
16     private $debug = false;
17
18     public function __construct($nx, $ny = false, $house_pct = ↩
           0.05, $pct = 0.05, $allow_nearby_house = true, $house_count ↩
           = false, $random_start = true)
19     {
20         if (!$ny) {
21             $ny = $nx;
22         }
23         $this->intersection_pct = $pct;
24
25         for ($i=0; $i < ($nx * $ny); $i++) {
26             $this->intersections[$i] = array(
27                 'x' => 0,
28                 'y' => 0
29             );
30         }
31
32         if ($random_start) {
33             // Start placed north, south, east or west?
34             switch(rand(0,3)) {
35                 case 0:
36                     // North
37                     $this->start = array(0,rand(1,$ny-2));
38                     $this->end = array($nx-1,rand(1,$ny-2));
39                 break;
40                 case 1:
41                     // South
42                     $this->start = array($nx-1,rand(1,$ny-2));
43                     $this->end = array(0,rand(1,$ny-2));
44                 break;
```

```
45                  case 2:
46                      // East
47                      $this->start = array(rand(1,$nx-2),$ny-1);
48                      $this->end = array(rand(1,$nx-2),0);
49                  break;
50                  case 3:
51                      // West
52                      $this->start = array(rand(1,$nx-2),0);
53                      $this->end = array(rand(1,$nx-2),$ny-1);
54                  break;
55              }
56          } else {
57              $this->start = array(1,0);
58              $this->end = array($n-2,0);
59          }
60
61          for ($i=0; $i < $nx; $i++) {
62              for ($j=0; $j < $ny; $j++) {
63                  if ($this->start == array($i,$j) || $this->end == ↩
                        array($i,$j)) {
64                      $this->grid[$i][$j] = Types::$TYPE_INTERSECTION↩
                            ;
65                      $this->intersection_coords[] = array('x' => $i,↩
                            'y' => $j);
66                      $this->intersections[$i]['x']++;
67                      $this->intersections[$j]['y']++;
68                  } else if ($i == 0 || $j == 0 || $i == $nx-1 || $j ↩
                        == $ny-1) {
69                      $this->grid[$i][$j] = Types::$TYPE_WALL;
70                  } else {
71                      // var isWall = Math.floor(Math.random()*(/↩
                            wallFrequency));
72                      $intersection = (rand(0,1000) / 1000) <= $pct;
73
74                      if ($intersection) {
75                          if ($this->canHaveIntersection($i,$j)) {
76                              $this->grid[$i][$j] = Types::↩
                                    $TYPE_INTERSECTION;
77                              $this->intersection_coords[] = array('x↩
                                    ' => $i, 'y' => $j);
78                              $this->intersections[$i]['x']++;
79                              $this->intersections[$j]['y']++;
80                              $pct = $this->intersection_pct;
81                          } else {
82                              $this->grid[$i][$j] = Types::↩
                                    $TYPE_GRASS;
83                          }
84                      } else {
85                          $this->grid[$i][$j] = Types::$TYPE_GRASS;
86                      }
87                  }
88              }
89          }
90
91          if ($this->debug) echo "1\n";
92          for ($i=1; $i < count($this->intersection_coords); $i++) {
93              if ($this->intersection_coords[$i]['x'] == $this->↩
                    intersection_coords[$i-1]['x'])
94                  $this->createRoad($this->intersection_coords[$i],↩
                        $this->intersection_coords[$i-1]);
```

```
95          }
96
97          if ($this->debug) echo "2\n";
98          usort($this->intersection_coords, function ($a, $b) {
99              return $a['y'] < $b['y'];
100         });
101
102         if ($this->debug) echo "3\n";
103         for ($i=1; $i < count($this->intersection_coords); $i++) {
104             if ($this->intersection_coords[$i]['y'] == $this->↩
                    intersection_coords[$i-1]['y'])
105                 $this->createRoad($this->intersection_coords[$i],↩
                        $this->intersection_coords[$i-1]);
106         }
107
108         if ($this->debug) echo "4\n";
109         // Make road from start to nearest intersection
110         $nearest_start = $this->findNearestIntersection($this->↩
                getStart('x'),$this->getStart('y'));
111         $star = new Astar($this->getMap(), $this->getStart('x'), ↩
                $this->getStart('y'), $nearest_start['x'], ↩
                $nearest_start['y']);
112         $path = $star->findShortestPath();
113         $this->pathToRoad($path);
114
115
116         if ($this->debug) echo "5\n";
117         // Make road from end to nearest intersection
118         $nearest_end = $this->findNearestIntersection($this->getEnd↩
                ('x'),$this->getEnd('y'));
119         if ($this->debug) var_dump($nearest_end);
120         $star = new Astar($this->getMap(), $this->getEnd('x'),$this↩
                ->getEnd('y'), $nearest_end['x'], $nearest_end['y']);
121         $path = $star->findShortestPath();
122         if ($this->debug) var_dump($path);
123         $this->pathToRoad($path);
124
125         if ($this->debug) echo "6\n";
126         // Ensure all intersections are reacable from another ↩
                intersection
127         $points = $this->getNonConnectedIntersections();
128         foreach($points as $point) {
129             $nearest = $this->findNearestIntersection($point['x'],↩
                    $point['y']);
130             $star = new Astar($this->getMap(), $point['x'],$point['↩
                    y'], $nearest['x'], $nearest['y']);
131             $path = $star->findShortestPath();
132             $this->pathToRoad($path);
133         }
134
135         if ($this->debug) echo "7\n";
136         // Ensure that end is reachable from start
137         $star = new Astar($this->getMap(), $this->getStart('x'),↩
                $this->getStart('y'), $this->getEnd('x'),$this->getEnd('↩
                y'));
138         $path = $star->findShortestPath();
139         $this->pathToRoad($path);
140
141         if ($this->debug) echo "8\n";
142         // Ensure all intersections are reachable from start
```

```
143         foreach ($this->intersection_coords as $coord) {
144             $star = new Astar($this->getMap(), $coord['x'],$coord['↩
                    y'], $this->getStart('x'), $this->getStart('y'));
145             $path = $star->findShortestPath();
146             if (!empty($path))
147                 $this->pathToRoad($path);
148         }
149 
150         // Place random houses
151         if ($house_count == false)
152         {
153             $houses_min = $nx*3;
154             $houses_max = $nx*4;
155             $house_count = rand($houses_min,$houses_max);
156         }
157         $tries = 0;
158         $max_tries = 3;
159         $timeout = 2;
160         $time_start = microtime(true);
161         $start =
162         $use = 'timeout';
163         while($house_count > 0) {
164             if ($use == 'timeout') {
165                 if (microtime(true) - $time_start > $timeout) {
166                     break;
167                 }
168             } else {
169                 if ($tries == $max_tries) {
170                     break;
171                 }
172             }
173             for ($i=0; $i < $nx; $i++) {
174                 for ($j=0; $j < $ny; $j++) {
175                     if ($house_count == 0) {
176                         break;
177                     }
178                     if ($this->grid[$i][$j] == Types::$TYPE_GRASS)
179                     {
180 
181                         // Higher chance near intersections?
182                         if ($this->hasNearbyRoad($i,$j) && (↩
                                $allow_nearby_house || !$this->↩
                                hasNearbyHouse($i,$j))) {
183                             // var isWall = Math.floor(Math.random↩
                                ()*(/wallFrequency));
184                             $chance = (rand(0,1000) / 1000) <= ↩
                                $house_pct;
185 
186                             if ($chance) {
187                                 $this->grid[$i][$j] = Types::↩
                                    $TYPE_HOUSE;
188                                 $this->houses[] = array('x' => $i, ↩
                                    'y' => $j);
189                                 $house_count--;
190                             }
191                         }
192                     }
193                 }
194             }
195             $tries++;
```

```
196         }
197
198         // Maybe connect intersections so that there is at most X between any two or start?
199     }
200
201     public function hasNearbyRoad($x,$y)
202     {
203         return $this->grid[$x-1][$y] == Types::$TYPE_INTERSECTION ||
204                 $this->grid[$x+1][$y] == Types::$TYPE_INTERSECTION ||
205                 $this->grid[$x][$y-1] == Types::$TYPE_INTERSECTION ||
206                 $this->grid[$x][$y+1] == Types::$TYPE_INTERSECTION ||
207                 $this->grid[$x-1][$y] == Types::$TYPE_ROAD ||
208                 $this->grid[$x+1][$y] == Types::$TYPE_ROAD ||
209                 $this->grid[$x][$y-1] == Types::$TYPE_ROAD ||
210                 $this->grid[$x][$y+1] == Types::$TYPE_ROAD;
211     }
212
213     public function hasNearbyHouse($x,$y)
214     {
215         return $this->grid[$x-1][$y] == Types::$TYPE_HOUSE ||
216                 $this->grid[$x+1][$y] == Types::$TYPE_HOUSE ||
217                 $this->grid[$x][$y-1] == Types::$TYPE_HOUSE ||
218                 $this->grid[$x][$y+1] == Types::$TYPE_HOUSE;
219     }
220
221     public function getNonConnectedIntersections()
222     {
223         $res = array();
224         foreach($this->intersection_coords as $val)
225         {
226             if (!$this->isConnected($val)) {
227                 $res[] = $val;
228             }
229         }
230         return $res;
231     }
232
233     public function isConnected($pos)
234     {
235         return ($pos['x'] - 1 > 0 && $this->grid[$pos['x'] - 1][$pos['y']] == Types::$TYPE_ROAD) ||
236             ($pos['x'] + 1 < count($this->grid) && $this->grid[$pos['x'] + 1][$pos['y']] == Types::$TYPE_ROAD) ||
237             ($pos['y'] - 1 > 0 && $this->grid[$pos['x']][$pos['y'] - 1] == Types::$TYPE_ROAD) ||
238             ($pos['y'] + 1 < count($this->grid[0]) && $this->grid[$pos['x']][$pos['y'] + 1] == Types::$TYPE_ROAD);
239     }
240
241     public function findNearestIntersection($x,$y)
242     {
243         $best_dist = 0;
244         $coord = array();
245         foreach($this->intersection_coords as $val)
246         {
```

```
247             if ($val['x'] == $x && $val['y'] == $y) {
248                 continue;
249             }
250             $dist = abs($x - $val['x']) + abs($y - $val['y']);
251             if ($best_dist == 0 || $best_dist > $dist)
252             {
253                 $coord = $val;
254                 $best_dist = $dist;
255             }
256         }
257         return $coord;
258     }
259
260     private function pathToRoad($path)
261     {
262         foreach($path as $node)
263         {
264             if ($this->grid[$node->x][$node->y] != Types::↩
                    $TYPE_WALL && $this->grid[$node->x][$node->y] != ↩
                    Types::$TYPE_INTERSECTION)
265                 $this->grid[$node->x][$node->y] = Types::$TYPE_ROAD↩
                        ;
266         }
267     }
268
269     private function createRoad($pos1,$pos2)
270     {
271         // Should use A*
272         if ($pos1['x'] == $pos2['x']) {
273             $start = min($pos1['y'],$pos2['y']) + 1;
274             $end = max($pos1['y'],$pos2['y']);
275             for ($i=$start; $i < $end; $i++) {
276                 if ($this->grid[$pos1['x']][$i] != Types::↩
                        $TYPE_WALL && $this->grid[$pos1['x']][$i] != ↩
                        Types::$TYPE_INTERSECTION)
277                     $this->grid[$pos1['x']][$i] = Types::$TYPE_ROAD↩
                            ;
278             }
279         } else if ($pos1['y'] == $pos2['y']) {
280             $start = min($pos1['x'],$pos2['x']) + 1;
281             $end = max($pos1['x'],$pos2['x']);
282             for ($i=$start; $i < $end; $i++) {
283                 if ($this->grid[$i][$pos1['y']] != Types::↩
                        $TYPE_WALL && $this->grid[$i][$pos1['y']] != ↩
                        Types::$TYPE_INTERSECTION)
284                     $this->grid[$i][$pos1['y']] = Types::$TYPE_ROAD↩
                            ;
285             }
286         }
287     }
288
289     private function canHaveIntersection($x,$y)
290     {
291         return $this->intersections[$x-1]['x'] == 0 &&
292                 $this->intersections[$x+1]['x'] == 0 &&
293                 $this->intersections[$x-1]['y'] == 0 &&
294                 $this->intersections[$x+1]['y'] == 0 &&
295                 $this->intersections[$y-1]['x'] == 0 &&
296                 $this->intersections[$y+1]['x'] == 0 &&
297                 $this->intersections[$y-1]['y'] == 0 &&
```

```
298                 $this->intersections[$y+1]['y'] == 0;
299     }
300 
301     public function __toString()
302     {
303         $out = "<div>";
304         foreach ($this->grid as $key => $value) {
305             $out .= "<div class='clear'>";
306             foreach ($value as $key2 => $value2) {
307                 $out .= '<span class="grid_item ';
308                 if ($value2 == Types::$TYPE_WALL) {
309                     $out .= 'wall';
310                 } elseif ($value2 == Types::$TYPE_ROAD) {
311                     $out .= 'road';
312                 } elseif ($value2 == Types::$TYPE_INTERSECTION) {
313                     $out .= 'intersection';
314                 } elseif ($value2 == Types::$TYPE_HOUSE) {
315                     $out .= 'house';
316                 }
317                 $out .= '"></span>';
318             }
319             $out .= "</div>";
320         }
321         $out .= "</div>";
322 
323         return $out;
324     }
325 
326     public function getGrid()
327     {
328         return $this->grid;
329     }
330 
331     public function getJson()
332     {
333         return json_encode($this->getGrid());
334     }
335 
336     public function getMap()
337     {
338         return $this->grid;
339     }
340 
341     public function getHouses()
342     {
343         return $this->houses;
344     }
345 
346     public function getStart($coord = false)
347     {
348         if ($coord)
349         {
350             return $this->start[$coord == 'x' ? 0 : 1];
351         }
352         else
353             return $this->start;
354     }
355 
356     public function getEnd($coord = false)
357     {
```

```
358         if ($coord)
359         {
360             return $this->end[$coord == 'x' ? 0 : 1];
361         }
362         else
363             return $this->end;
364     }
365 }
366 ?>
```

**Listing C.1:** world.php

# D

# Log analyser

```php
<?php
/*
  Query parameters
  --------------------
  ?c = class; [2, 3, 4, 32]
  ?mode = mode; [c = class, i = individual]
  ?skip, for mode = i only. Shows the 'skip'th user in 'class' class
  ?t = which data values to be shown (for latex graph) [possible, perfects]
  ?limit [optional], only show the 'limit' first games in the graph
 */
$class = isset($_GET['c']) ? $_GET['c'] : 2;

if ($class == 3) {
  $username = "bsc";
  $password = "bscpassword";
  $hostname = "bsc.crvwhcfju0fb.eu-west-1.rds.amazonaws.com";
} else {
  $username = "root";
  $password = "";
  $hostname = "localhost";
}

//connection to the database
$dbhandle = mysql_connect($hostname, $username, $password) or die("Unable to connect to MySQL");

$selected = mysql_select_db("bsc",$dbhandle) or die("Could not select db");

function getQuery($sql) {
  $result = mysql_query($sql) or die('Invalid query: ' . mysql_error());
  return $result;
}

function possiblePerfects($id) {
  $res = getQuery("SELECT COUNT(id) as c FROM cars WHERE user_games_id = " . $id);
  $row = mysql_fetch_assoc($res);
  return $row['c'];
}

function countPerfects($arr) {
  $c = 0;
```

```
41    foreach($arr as $v) {
42      $j = json_decode($v,true);
43      if ($j['perfect'] && $j['trash']['capacity'] != 0) {
44        $c++;
45      }
46    }
47    return $c;
48  }
49
50  $time_start = microtime(true);
51
52  // Users created during test
53  if ($class == 2) {
54    $result = getQuery('SELECT *,FROM_UNIXTIME(created_at) as ↩
        created_at_time FROM users WHERE created_at > UNIX_TIMESTAMP↩
        ("2013-06-10 10:00:00") and created_at < UNIX_TIMESTAMP↩
        ("2013-06-10 12:00:00")');
55  } else if ($class == 4) {
56    $result = getQuery('SELECT *,FROM_UNIXTIME(created_at) as ↩
        created_at_time FROM users WHERE created_at > UNIX_TIMESTAMP↩
        ("2013-06-10 12:00:00") and created_at < UNIX_TIMESTAMP↩
        ("2013-06-10 13:00:00")');
57  } else if ($class == 3) {
58    // Amazon is off by 2 hours
59    $result = getQuery('SELECT *,FROM_UNIXTIME(created_at) as ↩
        created_at_time FROM users WHERE created_at > UNIX_TIMESTAMP↩
        ("2013-06-10 06:00:00") and created_at < UNIX_TIMESTAMP↩
        ("2013-06-10 08:00:00")');
60  } else if ($class == 32) {
61    $result = getQuery('SELECT *,FROM_UNIXTIME(created_at) as ↩
        created_at_time FROM users WHERE created_at > UNIX_TIMESTAMP↩
        ("2013-06-10 08:00:00") and created_at < UNIX_TIMESTAMP↩
        ("2013-06-10 10:00:00")');
62  }
63  //print values to screen
64
65  $games = array();
66  while ($row = mysql_fetch_assoc($result)) {
67    $userid = $row['id'];
68    $sql = "SELECT * FROM (SELECT id FROM user_games WHERE user_games↩
        .users_id = {$userid} " .
69    "AND user_games.end IS NOT NULL GROUP BY id ORDER BY user_games.↩
        id ASC" .
70    ") t1, log, log_eav le " .
71    "WHERE log.user_games_id = t1.id " .
72    "AND le.log_id = log.id " .
73    "AND (le.key = 'trash_pickup' OR le.key = 'click_to')";
74    $logs = getQuery($sql);
75    //print_r($row);
76    if (mysql_num_rows($logs)) {
77      // Gather data
78      while($r = mysql_fetch_assoc($logs)) {
79        if ($r['key'] == 'trash_pickup') {
80          $games[$userid][$r['user_games_id']][$r['key']][] = $r['↩
              value'];
81        } else {
82          if (!isset($games[$userid][$r['user_games_id']]['clicks']))↩
                {
83            $games[$userid][$r['user_games_id']]['clicks'] = 0;
84          }
```

```
85          $games[$userid][$r['user_games_id']]['clicks']++;
86        }
87        if (!isset($games[$userid]['username'])) {
88          $games[$userid]['username'] = $row['username'];
89        }
90        /*
91            [key] => trash_pickup
92            [value] => {"trash":{"representation":3,"capacity":9,"↩
                house_id":2522,"index":1,"id":2517,"taken":true},"car↩
                ":{"representation":1,"capacity":8,"filled":9,"index↩
                ":0,"id":1997,"active":false},"perfect":false}
93        */
94      }
95      //$sql = "SELECT * FROM user_games WHERE user_games_id = {$r['↩
            user_games_id']}";
96      //$games[$r['user_games_id']]['game'] = mysql_fetch_assoc(↩
            getQuery($sql));
97    }
98  }
99
100 $mode = isset($_GET['mode']) ? $_GET['mode'] : 'c';
101
102 echo "<pre>";
103
104 $p_vs_c = array();
105 foreach($games as $uid => $ugames) {
106   $p_vs_c[$uid] = array(
107     'username' => $games[$uid]['username'],
108     'perfects' => array(),
109     'possible' => array()
110   );
111   foreach($ugames as $id => $game) {
112     if ($id == 'username') continue;
113
114     $p_vs_c[$uid]['perfects'][] = countPerfects($game['trash_pickup↩
            ']);
115     $p_vs_c[$uid]['possible'][] = (int)possiblePerfects($id);
116   }
117 }
118
119 if ($mode == 'i')
120 {
121   $json = array(
122     array('Game','Possible','Perfects')
123   );
124   $username = '';
125   $i = 0;
126   $skip = isset($_GET['skip']) ? $_GET['skip'] : 0;
127   foreach($p_vs_c as $uid => $stats) {
128   $username = $stats['username'];
129   $i++;
130   if ($i < $skip) continue;
131   foreach($stats['possible'] as $k => $v) {
132     $json[] = array($k+1,$v,$stats['perfects'][$k]);
133   }
134
135   $debug = $games[$uid];
136   break;
137   }
138 }
```

```
139
140 if ($mode == 'c')
141 {
142
143
144   $limit_games = isset($_GET['limit']) ? $_GET['limit'] : false;
145
146   // Sort out everything with less than 5 games
147   $min = 0;
148   foreach ($p_vs_c as $uid => $stats) {
149     if (count($stats['perfects']) < 5) { // One key is username
150       unset($p_vs_c[$uid]);
151       continue;
152     }
153     if ($min == 0 || $min > count($stats['perfects']))
154       $min = count($stats['perfects']);
155   }
156   // Find avg's
157   $correctness = array();
158   $players = array();
159   $i = 0;
160
161   foreach ($p_vs_c as $uid => $value) {
162     if ($limit_games) {
163       for ($i=0; $i < $limit_games; $i++) {
164         if (!isset($value['perfects'][$i])) break;
165         $correctness[$i][] = $value['perfects'][$i] / $value['↩
              possible'][$i];
166         if (!isset($players[$i]))
167           $players[$i] = 0;
168         $players[$i]++;
169       }
170     } else {
171       $tmp = count($value['perfects']);
172       for ($i=0; $i < $tmp; $i++) {
173         $correctness[$i][] = $value['perfects'][$i] / $value['↩
              possible'][$i];
174         if (!isset($players[$i]))
175           $players[$i] = 0;
176         $players[$i]++;
177       }
178     }
179   }
180   foreach ($correctness as $key => $value) {
181     $correctness[$key] = (array_sum($value) / count($value)) * 100;
182   }
183
184
185   $json = array(
186     array('Game','Correctness')
187   );
188   if (!$limit_games || $limit_games > $min) {
189     $json[0][] = 'Players';
190   }
191   $username = $class . '. klasse';
192   foreach($correctness as $game => $stats) {
193     if ($limit_games && $limit_games <= $min) {
194       $json[] = array($game+1,$stats);
195     } else {
196       $json[] = array($game+1,$stats,$players[$game]);
```

```
197       }
198     }
199   }
200
201   $time_end = microtime(true);
202   $time = $time_end - $time_start;
203   echo "Did nothing in $time seconds\n";
204
205   ?>
206   <html>
207     <head>
208       <script type="text/javascript" src="https://www.google.com/jsapi"></script>
209       <script type="text/javascript">
210         google.load("visualization", "1", {packages:["corechart"]});
211         google.setOnLoadCallback(drawChart);
212         function drawChart() {
213           // var data = google.visualization.arrayToDataTable([
214           //   ['Game', 'Possible', 'Perfects'],
215           //   ['2004',  1000,      400],
216           //   ['2005',  1170,      460],
217           //   ['2006',  660,       1120],
218           //   ['2007',  1030,      540]
219           // ]);
220           var data = google.visualization.arrayToDataTable(<?php echo json_encode($json); ?>);
221
222           var options = {
223             title: '<?php echo $username; ?>'
224           };
225
226           var chart = new google.visualization.LineChart(document.getElementById('chart_div'));
227           chart.draw(data, options);
228         }
229       </script>
230     </head>
231     <body>
232       <div id="chart_div" style="width: 900px; height: 500px;"></div>
233     </body>
234   </html>
235
236   <?php
237   $xmax = 0;
238   $ymax = 0;
239   $ymin = 100;
240   foreach ($json as $key => $value) {
241     if ((!is_int($value[0]))) continue;
242
243     if ($xmax < $value[0]) {
244       $xmax = $value[0];
245     }
246     if ($ymax < $value[1]) {
247       $ymax = $value[1];
248     }
249     if ($ymin > $value[1]) {
250       $ymin = $value[1];
251     }
252   }
253   ?>
```

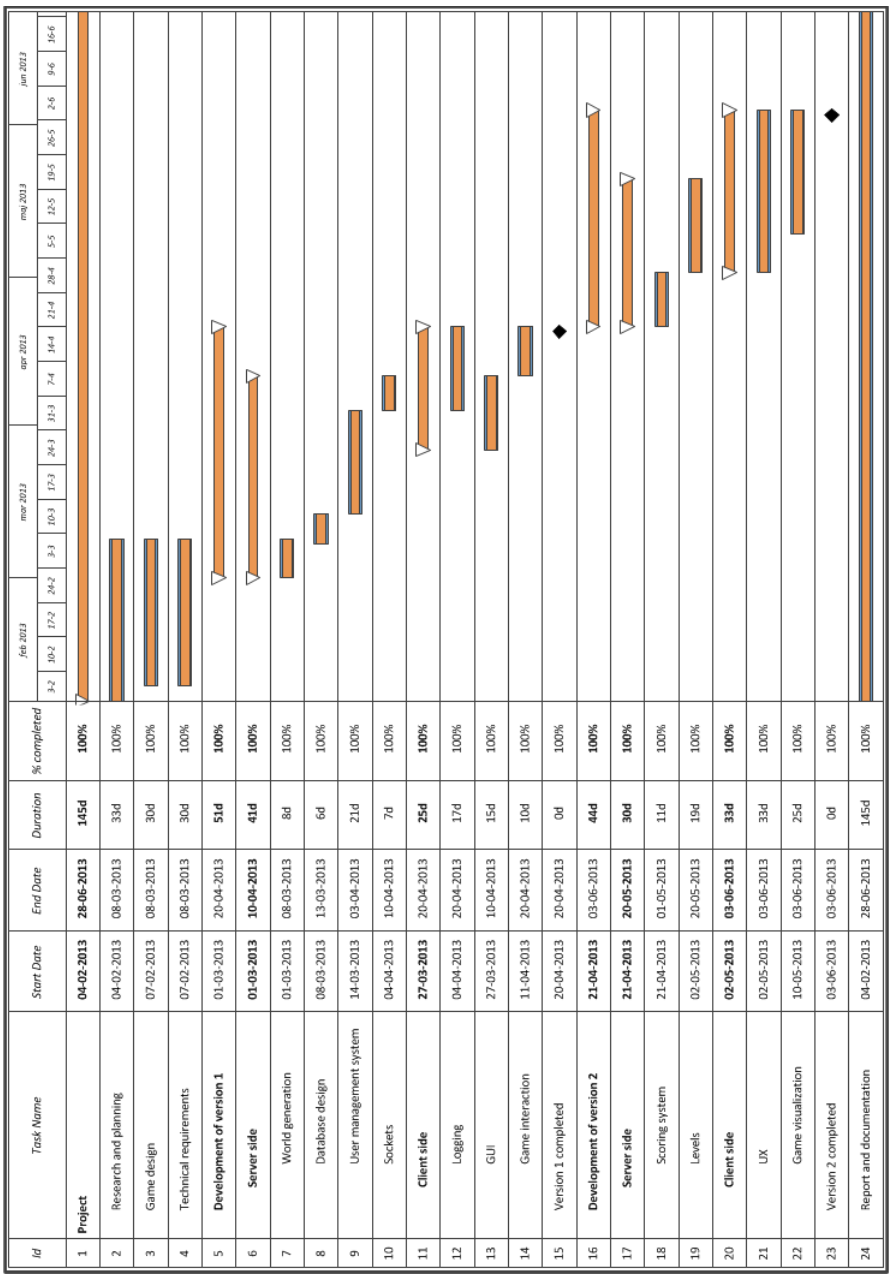
```
254
255 \documentclass{article}
256 \usepackage{tikz}
257
258 \begin{document}
259 \pagestyle{empty}
260
261 \begin{tikzpicture}[x=1cm,y=0.4cm]
262
263   \def\xmin{0}
264   \def\xmax{<?php echo ceil($xmax + 0.5); ?>}
265   \def\ymin{<?php echo ceil($ymin - 0.5); ?>}
266   \def\ymax{<?php echo ceil($ymax + 0.5); ?>}
267
268   % grid
269   \draw[style=help lines, ystep=2, xstep=1] (\xmin,\ymin) grid
270   (\xmax,\ymax);
271
272   % axes
273   \draw[->] (\xmin,\ymin) -- (\xmax,\ymin) node[right] {$x$};
274   \draw[->] (\xmin,\ymin) -- (\xmin,\ymax) node[above] {$y$};
275
276   % xticks and yticks
277   \foreach \x in {1,2,...,<?php echo ceil($xmax + 0.5); ?>}
278     \node at (\x, \ymin) [below] {\x};
279   \foreach \y in {<?php echo ceil($ymin - 0.5); ?>,<?php echo ceil(↩
        $ymin + 0.5); ?>,...,<?php echo ceil($ymax + 0.5); ?>}
280     \node at (\xmin,\y) [left] {\y};
281
282   % plot the data from the file data.dat
283   % smooth the curve and mark the data point with a dot
284   \draw[color=blue] plot[smooth,mark=*,mark size=1pt] file {data.↩
        dat}
285    node [right] {correctness};
286
287 \end{tikzpicture}
288
289 \end{document}
290
291 \begin{tikzpicture}
292 \begin{axis}
293 \addplot[color=black,solid,thick,mark=*, mark options={fill=white}]
294     coordinates {
295           <?php
296           foreach ($json as $key => $value) {
297             if ($key == 0) continue;
298             echo "(".$value[0].",".number_format($value[1],2).")\n"↩
                    ;
299           }
300           ?>
301         };
302 <?php
303 foreach ($json as $key => $value) {
304   if ($key == 0) continue;
305   echo "\\node [above] at (axis cs:  {$value[0]},  ".number_format(↩
        $value[1],2).") ".'{$'.number_format($value[1],2).'$};'."\n";
306 }
307 ?>
308 \end{axis}
309 \end{tikzpicture}
```

```
310
311 <?php
312 echo "<pre>";
313 foreach ($json as $key => $value) {
314   if ($mode == 'c')
315     echo $value[0]."\t".$value[1]."\n";
316   else
317     if (isset($_GET['t']) && $_GET['t'] == 'possible')
318       echo $value[0]."\t".$value[1]."\n";
319     else
320       echo $value[0]."\t".$value[2]."\n";
321 }
322 echo "</pre>";
323
324 class objectify
325 {
326   public function json_mapper($value, $recursive = true) {
327     if (!empty($value) && is_string($value) &&
328         $decoded = json_decode($value, true)) {
329       return $decoded;
330     } elseif (is_array($value) && $recursive) {
331       return array_map('objectify::json_mapper', $value);
332     } else {
333       return $value;
334     }
335   }
336
337   // currying, anyone?
338   public function json_mapper_norecurse($value) {
339      return objectify::json_mapper($value, false);
340   }
341
342   public function json_to_array($array, $recursive = true)
343   {
344     # if $array is not an array, let's make it array with one value↩
          of
345     # former $array.
346     if (!is_array($array)) {
347       $array = array($array);
348     }
349
350     return array_map(
351       $recursive ? 'objectify::json_mapper'
352                  : 'objectify::json_mapper_norecurse', $array);
353   }
354 }
355
356   $o = new objectify();
357 if (isset($debug)) {
358   print_r(json_encode($o->json_to_array($debug)));
359 } else {
360   print_r(json_encode($o->json_to_array($games)));
361 }
```

**Listing D.1:** analyser.php

# E

# Project plan



| Id | Task Name | Start Date | End Date | Duration | % completed |
|---|---|---|---|---|---|
| 1 | **Project** | **04-02-2013** | **28-06-2013** | **145d** | **100%** |
| 2 | Research and planning | 04-02-2013 | 08-03-2013 | 33d | 100% |
| 3 | Game design | 07-02-2013 | 08-03-2013 | 30d | 100% |
| 4 | Technical requirements | 07-02-2013 | 08-03-2013 | 30d | 100% |
| 5 | **Development of version 1** | 01-03-2013 | 20-04-2013 | **51d** | **100%** |
| 6 | **Server side** | **01-03-2013** | **10-04-2013** | **41d** | **100%** |
| 7 | World generation | 01-03-2013 | 08-03-2013 | 8d | 100% |
| 8 | Database design | 08-03-2013 | 13-03-2013 | 6d | 100% |
| 9 | User management system | 14-03-2013 | 03-04-2013 | 21d | 100% |
| 10 | Sockets | 04-04-2013 | 10-04-2013 | 7d | 100% |
| 11 | **Client side** | **27-03-2013** | 20-04-2013 | **25d** | **100%** |
| 12 | Logging | 04-04-2013 | 20-04-2013 | 17d | 100% |
| 13 | GUI | 27-03-2013 | 10-04-2013 | 15d | 100% |
| 14 | Game interaction | 11-04-2013 | 20-04-2013 | 10d | 100% |
| 15 | Version 1 completed | 20-04-2013 | 20-04-2013 | 0d | 100% |
| 16 | **Development of version 2** | **21-04-2013** | 03-06-2013 | **44d** | **100%** |
| 17 | **Server side** | **21-04-2013** | **20-05-2013** | **30d** | **100%** |
| 18 | Scoring system | 21-04-2013 | 01-05-2013 | 11d | 100% |
| 19 | Levels | 02-05-2013 | 20-05-2013 | 19d | 100% |
| 20 | **Client side** | **02-05-2013** | **03-06-2013** | **33d** | **100%** |
| 21 | UX | 02-05-2013 | 03-06-2013 | 33d | 100% |
| 22 | Game visualization | 10-05-2013 | 03-06-2013 | 25d | 100% |
| 23 | Version 2 completed | 03-06-2013 | 03-06-2013 | 0d | 100% |
| 24 | Report and documentation | 04-02-2013 | 28-06-2013 | 145d | 100% |

**www.compute.dtu.dk**

Department of Applied Mathematics and Computer Science
Technical University of Denmark
Matematiktorvet
Building 303 B
DK-2800 Kgs. Lyngby
Denmark
Tel: (+45) 45 25 30 31
Fax: (+45) 45 88 26 73
E-mail: compute@compute.dtu.dk