

Bridging Gaps with Evolution

Submitted April 2016, in partial fulfilment of the conditions of the award of
the degree BSc (Hons) Computer Science with Artificial Intelligence

Barnabas Forgo

4211949 / bxf03u

With supervision from Per Kristian Lehre

School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature: _____

Date: ____ / ____ / ____



Abstract

The aim of this project was to develop an online visualisation tool for genetic algorithms. In order to achieve this, a visual problem was created, a 'bridge building game', which served as the problem domain for the algorithm to solve. In this report, it is shown how some software engineering practices and modern web technologies such as Angular and EcmaScript 6 were used to develop this tool in a modular fashion. In the end, user evaluation has shown that an enjoyable game was developed, which also comes with a customisable genetic algorithm and an intuitive visualisation tool that can show how solutions to the problem are achieved.

Acknowledgments

I would like to thank my supervisor, Per Kristian Lehre, who provided me with invaluable advice and ideas throughout the project, especially with the implementation of the genetic algorithm.

I am grateful to my parents as well; without their unconditional, continuous support it would not have been possible for me to get this far.

I would like to thank Paulina Czubacka. Her reassuring words kept me going even when I had serious doubts about my abilities to carry out this project. Her feedback was a great source of ideas, which influenced the project in many ways.

Finally, I am thankful to all the volunteers who were evaluating the app at any stage of the project. Their feedback helped the software become what it is today.

Table of Contents

1	Introduction.....	1
1.1	Motivation.....	1
1.2	What is a Genetic Algorithm?.....	1
1.3	Problem Description	2
2	Related Work.....	3
2.1	NP-hard Problems	3
2.1.1	The Unit Commitment Problem	3
2.1.2	2D and 3D Bin Packing	3
2.2	GA Visualisation	4
2.2.1	A Genetic Car Simulation	4
2.2.2	Ancestry of Chromosomes	4
2.3	Truss Optimisation	5
2.3.1	Simple GA	5
2.3.2	Grammatical Evolution.....	6
3	System Specification	7
3.1	Functional Requirements	7
3.1.1	Requirements for the Simulation	7
3.1.2	Requirements for the GA	7
3.2	Non-Functional Requirements	9
4	UI Design.....	10
4.1	First Iteration	10
4.2	Second Iteration	14
4.3	Evaluation of the UI Design	16
5	Software Implementation	17
5.1	Key Implementation Decisions	17
5.1.1	Back-end: Node.js & Express	17
5.1.2	Front-end: Angular.....	18
5.1.3	Development Tools: Babel, Gulp & Yeoman	19
5.1.4	Game Engine: Turbulenz.....	20
5.2	Project Management.....	21
5.3	Internal Design.....	21
5.3.1	Simulation.....	21
5.3.2	GA	23
5.4	Implementation of System Components	23
5.4.1	Simulation.....	23

5.4.2 GA	24
5.5 Detailed Description of the User Interface	26
5.5.1 Main View.....	26
5.5.2 GA Settings Dialog	27
5.5.3 GA View	28
5.6 Problems Encountered	29
5.6.1 Turbulenz	29
5.6.2 GA	30
6 Evaluation of the Project.....	31
6.1 Comparison with Requirements	31
6.1.1 Functional Requirements	31
6.1.2 Non-Functional Requirements.....	31
6.2 Evaluating the GA	32
6.3 User evaluation.....	32
6.4 Personal Evaluation.....	33
7 Further Work.....	34
8 Summary.....	35
9 References	36
10 Appendix.....	39
A. Original System Requirements Specification	39
Functional Requirements.....	39
Non-Functional Requirements	40
B. Comparison with Functional Requirements	41
Passed tests	41
Failed tests.....	44

1 INTRODUCTION

1.1 Motivation

Genetic algorithms (GA) are stochastic search algorithms that were first proposed by Holland in 1975 [1]. Since then, many attempts have been made to gain a better understanding of the underlying mechanics that make this search method so effective. Many of these attempts were focused on visualising how candidate solutions' strength converges over time [2-6]. However, most of these attempts only yielded dull plots that are hard to understand and therefore do not truly reveal anything new about the processes that produced the solution.

To tackle this, others experimented with showing how each trait of a candidate is achieved by visualising its parent solutions [7, 8]. Another approach is to show how candidate solutions relate to each other by colour coding them [9] or by projecting high-dimension solutions to 2D [10].

The issue with these solutions is that the problem that is being solved by the GA is usually quite abstract and therefore it does not lend itself to being visualised. The aim of this project is to solve this problem by applying GAs to a new problem that is easier to grasp, so that the algorithm's performance can be visualised intuitively, using web technologies.

1.2 What is a Genetic Algorithm?

GAs are search algorithms that mimic natural selection to find solutions to hard problems. Simply speaking, this algorithm applies the notion of 'survival of the fittest' to search problems, where the fittest is the solution that is closest to some arbitrary goal. The knowledge of basic terms is essential before proceeding to explain GAs in greater detail.

- *Population*: A GA does not process solutions one by one, but instead a set of solutions. The size of the population is one of the most important parameters that decide the effectiveness of the algorithm. If this number were too high, it would take too long to compute; if it were too small then the search might stall in a local maximum.
- *Chromosomes* are an abstract representation of the problem. Initially, the most common representation was a binary string of fixed length, however there are some alternative solutions that work with a real number representation (real-coded GAs, RCGA, [11]). The population consists of many chromosomes.
- *Gene*: Each chromosome consists of many genes. Each gene represents a trait or a small portion of the solution. E.g., in a route planning problem a chromosome would be a whole route and a single turn would be a gene.
- *Fitness function*: To determine how good each of the candidates is, an objective evaluation function is needed. This function varies with each problem, and is constructed using knowledge about the problem.

There are six crucial steps for every GA:

1. *Initialisation*: the first population is generated, usually in a random manner; however, knowledge about the search problem can be used to generate better starting candidates.
2. *Evaluation*: in the current population, chromosomes' fitness is evaluated using the fitness function.
3. *Selection*: the fittest members of the population are selected for reproduction. There are several ways to do this; these include but not limited to roulette-wheel selection [1], stochastic universal selection [12, 13] or tournament selection [14].
4. *Crossover*: the selected fittest members recombine in a way that no new offspring will be identical to the parents; instead, they will inherit some traits from each parent. Some specific methods are k-point crossover, uniform crossover [15] or partially matched crossover [16].

5. *Mutation*: one or more of the solutions is modified in usually a random way. This step ensures diversity in the population. The most common mutation operator (in binary encoded GAs) is bit-flipping, that is, each gene's value is changed from 1 to 0 (and vice versa) with a certain probability.
6. *Replacement*: the original population is replaced by the generation created with steps 2-5. Again, there are several ways to do this; the most common is when the whole population is replaced by the new one.

Steps 2-6 are repeated until some criteria is met (for example a certain fitness level is achieved). This can be expressed using pseudo code as follows.

```

//Initialisation
Let population be a random set of chromosomes
//Evaluation
Let fitness be FitnessFn(individual) for each individual in population
Until some stopping criteria is met do {
  Let newPopulation be an empty set
  While newPopulation is not large enough {
    //Selection
    Let parent1 and parent2 be SelectionFn(population, fitness)
    //Crossover
    Let child1 and child2 be CrossoverFn(parent1, parent2)
    //Mutation
    With some low probability MutateFn(child1) or MutateFn(child2)
    Insert child1 and child2 into newPopulation
  }
  //Replacement
  Let population be newPopulation
}
Return best individual

```

Figure 1 GA pseudo-code

1.3 Problem Description

The goal of this project is to develop a 'bridge building game', which serves as the problem domain for the GA to solve. The aim of this game is to take a van from one side of a gap to the other by means of a bridge in a physics simulation. The bridge consists of edges of limited length that are connected together when they touch on the ends of each other. Each edge contributes to the cost of the bridge. The goal of this toy is to construct the most efficient bridge, that is, the cheapest bridge that can carry the most load.

The search problem that the GA will solve on this domain is to construct a bridge that satisfies these criteria. Since the nature of the problem is unique, some new mutation operators will be developed. As this complex world model is hard to describe mathematically, simulation results will be used to evaluate each candidate solution, instead of static analysis of the data. Finally, a new approach will be developed to visualise the algorithm's performance.

2 RELATED WORK

This chapter will discuss a few problems that were solved using GAs, some of which also use some form of visualisation to reveal details about the algorithm.

2.1 NP-hard Problems

The following two use-cases demonstrate how GAs can be used to accurately approximate solutions to problems that were thought to be unsolvable, or the solutions that existed did not provide results that were good enough to be used in real world problem instances.

2.1.1 The Unit Commitment Problem

The unit commitment problem tries to find the optimal uptime for individual power generators (units) in a power system. The objective is to minimise the operating costs, while taking several constraints into account (for example fuel costs, reserve requirements, crew constraints, minimum uptime, etc.). This problem was tackled in several ways (using for example Lagrangian Relaxation [17], Simulated Annealing [18], Hopfield Neural Networks [19]) during the years, but since the search space is very large, these solutions could not be applied to real world power systems. In 1996 Kazarlis, et al. [20] applied a GA to the unit commitment problem.

The fitness function used took the operating constraints into account, weighting them appropriately. Since a canonical GA could not provide adequate results in a reasonable time, several modifications had to be made. Two new mutation operators were added that modify a certain time window of units. Then two other mutation operators were added that only apply to the fittest chromosome. These mutations perform a hill climbing search, making the algorithm a hybrid GA. This time the experiment could find near optimal solutions. Lastly, the so called varying quality function technique was applied (first developed by Petridis and Kazarlis [21]). This essentially modifies the fitness function so that it applies inversely proportional weight on “penalties” (coming from breaking constraints) with respect to the generation number. The algorithm this time, could find the optimal solution.

As a conclusion this solution proved to be reliable, as with these modifications, an optimum solution was always found (sometimes even a better one than what a Lagrangian Relaxation would provide), even though for 100 units it took about 4.5 hours to run (in 1996). It is important to note though, that Fig. 10 in [20] shows a quadratic complexity, which means that this algorithm does not scale very well, but this might have been because the generation limit was chosen to be too high.

2.1.2 2D and 3D Bin Packing

Bin packaging is a well-studied NP-hard problem; many solutions exist to it both in 2D and 3D (Tabu search (2D/3D) [22, 23], Guided local search (2D/3D) [24], Set covering based heuristic (2D) [25], etc.) but most of them take too long to compute or produce suboptimal solutions. In 2013, Gonçalves and Resende [26] used a version of a random-key genetic algorithm (RKGA; first developed by Bean [27]) to solve this problem and produced very promising results. The difference from a simple GA is that in this version, genes are real numbers in the interval $[0, 1]$.

In their representation each chromosome contains $2n$ genes, where n is the number of boxes to package. The first n genes is the sequence in which boxes are packed, the second n genes is the vector for each box’s orientation. The fitness function traditionally was simply the number of bins that are needed to package every box; however, this does not provide enough granularity, so the researchers devised a new one that takes a solution’s improvability into account. A new packing heuristic was developed as well. This improved the evaluation step and thus, the GA produced better results.

The researchers experimented on more than 800 problem instances that were run on existing implementations and the one discussed above. The results were quite conclusive: when no rotations were enabled (the standard), the GA produced equal or better solutions than the other algorithms. When rotations were enabled (more realistic), the GA saved up to two bins in some cases. It can be said that this approach outperforms any previous ones.

2.2 GA Visualisation

2.2.1 A Genetic Car Simulation

The inspiration for this project comes from Rafael Matsunaga, who developed an unconventional use of GAs in 2014 [28]. This game uses a GA to evolve “cars” to get faster and further in a set 2D terrain.

Since there is only little explanation on how this particular implementation works, the source code [29] was analysed. It was found is that it is a simple GA with the following steps:

- the fitness function is the distance travelled
- the fittest members are copied to the next generation
- mating is random and the offspring is recombined using a 2-point crossover
- mutation is problem specific
- finally, the old generation is simply replaced

Since this is a fairly unique application of a GA, it could not be compared or benchmarked with another implementation. It is clear though that it is working properly since even after a few dozen generations, the average performance of cars improves drastically.

Why this solution is interesting is that it provides a great visualisation of GAs. By making the problem itself largely visual, it makes it easy to see how the different operations such as crossover and mutation affect the population. Furthermore, not only the problem is visual but the evaluation as well (as the simulation is used to evaluate individual cars), which makes this toy an interesting observational pastime.

This project takes several ideas from this game. Most importantly, the problem to be solved by this project, bridge design, was chosen, as it is similarly a visual problem that does not have a straightforward solution. Secondly, the fact that users can select the various parameters for the algorithm makes the game immersive, and it shows how each parameter can affect the population. However, this game only goes as far as four key parameters. To expand upon this idea this project will have a lot more adjustable parameters, so that the GA can be fine-tuned to fit the problem.

2.2.2 Ancestry of Chromosomes

Hart and Ross [8] developed a Java tool called GAVEL (GA Visualisation of Evolutionary Links) that can show how each gene in the chromosome came to be by tracking the ancestors of it, back to the initial generation. The way this is being represented is a simple tree graph, where each chromosome is a small box. These boxes are colour coded to show how each individual was produced (e.g. crossover & mutation or just copied from the previous generation, etc.). Moreover, ancestry can be tracked on each individual gene; the software can show which ancestors had the specific gene and trace it back to the first generation where it appeared. The system can also show graphs of how productive each operator was, that is, how many chromosomes were placed into the given population by applying the given operator.

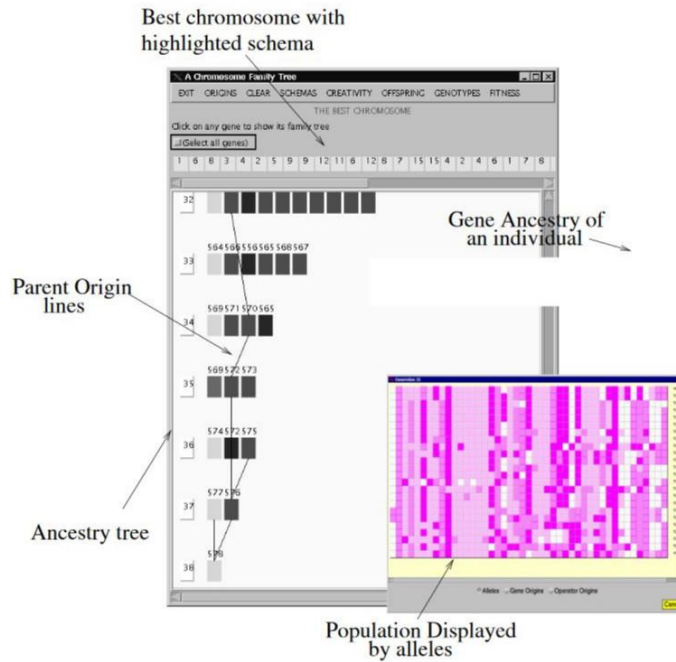


Figure 2 Example display from GAVEL [7]

A shortcoming of this tool is that it can only trace the ancestors of the best individual. Therefore, it does not show how the whole population of each generation interacts with each other. To demonstrate this tool the authors solved three problems (timetabling, job-shop scheduling, Goldberg and Horn's long-path problem), none of which can be visualised well. Therefore, when individual genes' origins are traced, the visualisation is not intuitive, and as such, it does not make it clearer how the end solution was achieved.

Nevertheless, this tool's main feature, the tree view of ancestors, is a very interesting way of looking at GAs and a similar ancestry view will be used in this project.

2.3 Truss Optimisation

This project is not the first time computers have been applied to designing bridges. In the literature, these structures are called trusses. However, trusses are more general structures than bridges. Trusses consist of members that distribute the load of the structure. Members are connected only on their ends (nodes). Members distribute the external force applied to the structure by compressing and extending. According to this definition, some roofing structures can be considered a truss structures as well as electricity pylons.

2.3.1 Simple GA

In 1992 Rajeev and Krishnamoorthy [30] used a simple GA to generate truss structures that consist of a pre-set number of edges. They used binary encoding to represent chromosomes. The selection was done by a fitness proportionate roulette wheel selection. The crossover operator was just a 2-point crossover. No mutation operator was used. Even though this configuration is simple, they produced surprising results, by designing a stable transmission tower consisting of 160 edges using this method.

What this example shows is that even the simplest GA configurations can develop great results. However, this solution has a shortcoming, which is, the fact that the members in the structure can only take some predefined places. This, on one hand limits the search space but it also introduces some bias in the system, which can lead to undesired results.

2.3.2 Grammatical Evolution

Classical structural optimisation algorithms (such as the one above) do not consider all possible configurations in the search space, because they only try to select the connections between fixed nodes and do not optimise the location of the nodes themselves. Rozvany [31] has shown that globally optimal solutions for such discrete optimisation problems cannot be found with enforced layouts.

To tackle this problem Fenton et al. [32] applied a version of GAs called grammatical evolution [33] to tackle this problem. Grammatical evolution is not solving a problem directly; instead, it tries to develop the program that solves the problem. The way it accomplishes this is that it uses a grammar in the Backus-Naur Form [34] as the ruleset for generating code. In this case, the generated code only defines the list of nodes. Connections between them are not explicitly considered but formed by connecting nodes to form triangles using Delaunay's triangulation algorithm [35]. Using this simplification, only kinematically stable structures are generated by the algorithm. The GA was configured to use tournament selection, with elitist replacement, and a single-point crossover.

To benchmark their solution the authors used example problems previously solved by other algorithms. In all cases, this algorithm achieved the optima that were presented in previous papers; in other problem instances, it discovered new optima. The only problem with this method is that it assumes a rectangular 'design envelope', that is, it can only design trusses that reside in a rectangle. If a problem instance requires a specific shape, this algorithm will not be able to solve it. However, this issue might be solved easily by introducing penalties when the shape requirement is broken. To summarize, this is a good example that shows that with an appropriate problem representation complex structures can be created with a GA.

3 SYSTEM SPECIFICATION

To aid the implementation a complete System Requirements Specification document was established at the beginning of the project. This document was used so that the milestones in the project can be easily identified and their completion date can be predicted with some accuracy. The full listing of original requirements can be found in [Appendix A](#). This document consists of two parts: functional and non-functional requirements. Functional requirements describe what the system should do, while non-functional requirements impose some constraints on the system such as performance requirements. The functional requirements have two key parts: simulation and GA solver, which will be discussed below.

3.1 Functional Requirements

3.1.1 Requirements for the Simulation

There will be a physics simulation that will be based on a grid system (world). There will be two platforms on the two sides of the world (start & goal). The distance (gap) and elevation between these two sides will be adjustable as well as the gravitation in the world. The aim of the game will be to build a bridge that can take a car or van to the goal side of the world. The bridge will be similar to a truss structure, that is, it will be built of members (or edges) of limited length that connect on the ends (nodes) by a revolute joint (constraint). Nodes will only reside on grid points. The cumulative cost of nodes and edges will be the cost of the bridge.

The original requirements included that edges may be one of multiple materials, however, this was changed to be only of a single material. The reason for this diversion is that after it was implemented it proved to be too hard to handle for the GA. When only one material is usable, the length limit of it is a factor that introduces a bias to the search, thus guiding it to some extent. When this bias is removed by adding more materials that can be longer, then it would be more likely for the GA to explore designs that were not fruitful, and consequently reducing the convergence time of the algorithm considerably.

Users will be able to design bridges by drawing edges in a click and drag fashion. Nodes will be generated automatically at the ends of edges instead of adding them explicitly. The editor will have undo and redo functionality and will display the cost of the bridge at all times. There will be an option to store and load bridges created in the editor; these will be stored in the browser's local storage. When the user is happy with the designed bridge, she will be able to start the simulation. In this mode, the maximum load of the bridge will be displayed which will be the main indicator of the quality of the bridge.

The van will always be placed on the left side of the world, and it will try to move towards the goal at a constant speed. The bridge will be the only means for the van to cross the gap. The weight of the van will be an adjustable parameter.

To summarize there will be four adjustable parameters for the simulation: gravitation, van's weight, gap width, and elevation. These parameters will also affect the GA as well so that it can be observed how these parameters change the bridges generated by the algorithm. The reason why these parameters were chosen is that they are easy to understand and they have a fundamental impact on the simulation.

3.1.2 Requirements for the GA

At the start of the project, it was rather unclear how the GA would operate, therefore in the original requirements there were some gaps left that were to be filled as the implementation progressed.

For the GA implementation a fitness function is required, which objectively scores how good generated bridges are. Bridges will be scored based on simulation data, that is, generated bridges will be placed

in the physics simulation and evaluated using that instead of trying to infer the score of it based on its static structure. This simulation will be visible to the user; however, this can be turned off to speed up the process or paused when an interesting solution has been achieved. To further increase the convergence speed, the user will be able to choose how many parallel simulations will run (in a tiled layout) before the GA process starts.

From the simulation data that is gathered through this process, the following variables will be used: if the bridge collapses then the score will be based on the time it takes for that to happen. Otherwise, if the bridge is stable and the van can pass through it, the maximum load will be used to calculate the fitness. Initially only these two variables were used, however, later many others were added to the equation, such as percentage of broken constraints, the distance the van travelled until the bridge collapsed, etc. to improve the GA's performance.

Based on this score a leader board will be available showing the best bridges. Furthermore, the bridges shown in the leader board will have an option to show their ancestors.

For the GA not only the world parameters will be adjustable but the control variables for the algorithm as well. Originally, in the requirements this was limited to the following parameters:

- ☐ mutation rate
- ☐ mutation types
- ☐ crossover rate
- ☐ crossover types
- ☐ selection rate
- ☐ population size
- ☐ initial population from saves

As the implementation progressed, the requirements were revised to exclude crossover operators altogether as the population is small and most chromosomes look very much alike, thus it would just introduce another layer of complexity without improving the algorithm. This meant that two of the crossover parameters were not needed. To compensate for this it was decided that much greater customisability would be given to the user by implementing a way to edit the fitness and selection functions directly. Obviously, since not all users are programmers this will be hidden behind a switch for advanced users along with the mutation type selectors. Another implication is that new chromosomes can only be created by mutating another one; therefore, the ancestry of a chromosome would not be a 'family tree' but a 'heritage line'.

Another diversion from the original requirements is the removal of stopping criteria. In most GAs the algorithm can be stopped when some criteria is met, e.g. when a solution is achieved with a given fitness value. However, it was decided to exclude this later as the algorithm is designed to run indefinitely and the user can choose to pause the algorithm at any time.

Yet another change is the exclusion of user interaction in the GA. The initial specification mentioned that users would be able to boost the score of some bridges by selecting one at the end of each generation. This idea was discarded, as there are countless bridges generated that are very low quality and repeatedly looking at unsatisfactory bridges would lead to user fatigue and thus dissatisfaction. The other interaction that was removed is the ability to 'sculpt' the bridges by manipulating them in the simulation. This was discarded, because when multiple simulations are running they tend to be very small to precisely interact with them and since simulations are very short (<3s on normal speed) there is not enough time for the user to react.

Lastly, it was planned to offload some of the simulations to the server side of the application. This decision was made before the game engine was chosen. Only later did it turn out that there are not any game engines that support both the back and front-end of the application, therefore this was not a feasible plan in the first place.

3.2 Non-Functional Requirements

- Performance Requirements:

The app should run reasonably well on an average laptop so that a wide range of users can access it.

- Security Requirements:

Since there is no sensitive data transmitted, there is no need to use HTTPS; HTTP will be sufficient.

- Software Quality Attributes:

The software should be tested according to the highest standards, to avoid any bugs in the final product. If bugs are discovered despite the testing efforts, they should be resolved as soon as possible. Furthermore, due care should be taken to modularise the system to make it easier to maintain the software later on. Comments should be present across the whole software to make it easier to read. To manage the source code of the program git shall be used as the version control tool, and to avoid disasters the repository should not only be local but mirrored to a private GitHub repository.

- Accessibility & Usability Requirements:

Since the application will be web-based, accessibility is not really a concern, as anyone will be able to use the site with an internet connection and a web browser. However since the simulation combined with the GA will be very resource intensive, mobile devices will not be considered when implementing the software. Furthermore, as GAs can be daunting even to programmers who have not learned about it, appropriate help sections should be placed across the UI. Nevertheless, care should be taken to implement the UI in a way that it is easy to understand, by using well-known design metaphors and visual cues.

4 UI DESIGN

4.1 First Iteration

Accompanying the System Requirements Specification, some UI design prototypes were made, to guide the implementation of the software. The first iteration of this design was not considering usability as its top priority; rather, it was just an attempt to organise the requirements discussed earlier into some discrete screens. The reason for this is that the design was made well before any implementation decisions were made, mainly to demonstrate and solidify some of the ideas behind the project. Nevertheless, the basic structure of this design persisted, as the design metaphors that were used, exist in many other pieces of software and therefore it makes it easier for users to learn how to use this app.

The first key decision that was made in designing the UI was to separate the functionality into three distinct screens. These are the 1) editor, 2) GA settings, and 3) GA viewer. Two of the three screens (editor and GA viewer) are there because these are the two main components of the app. Even though they have very different functionality, their layout was decided to be similar to introduce some consistency in the app. The settings screen was introduced for two main reasons. The first one is that, as discussed earlier, GAs require some parameters to be set before they are ran. Even though some of these parameters could be changed while the algorithm is running, it would rarely be sensible. This is because the change in parameters would just make solutions up to that point useless, as they were optimised for a different set of parameters. The second reason for the settings screen is to separate the two core parts of the app. This is to avoid confusion in transitioning between two very similar pages that accomplish very different things.

Another key design decision was to use Google’s Material Design (MD) specification [36] in the project. This design language was introduced by Google in 2014 to replace their somewhat fragmented designs across different products. MD was created with the intention to design a “system that allows for a unified experience across platforms and device sizes” [36]. The core concept behind this language is to mimic some behaviours of paper and ink by taking ideas from print-based design. Moreover, it uses light, shadows and motion to attach meaning to UI elements, which make it easy for the user to understand core intentions behind the design of the UI. Even though the specification is quite extensive, it leaves plenty of space for innovation. Other similar design languages or frameworks were evaluated before choosing this one, such as Twitter’s Bootstrap 3 [37] or Zurb’s Foundation 5 [38], however, MD was chosen in the end for two key reasons. Firstly, because MD seemed to be the framework that potential users might be most familiar with, as this design language is used across most Google products, most of which are market leaders in their respective categories [39-42]. Secondly, because MD has the most customisation options compared to the other mentioned systems.

1) Main Game

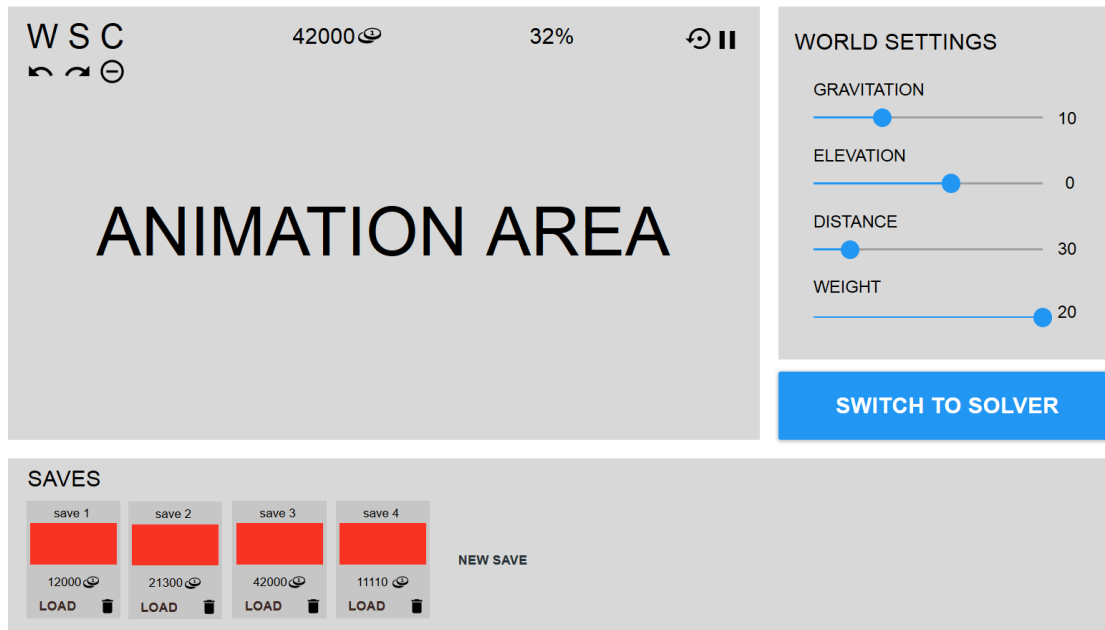


Figure 3 The landing screen – first iteration

- *Animation area:* Since this is the most important element in the app, this occupies most of the screen space. In this area, there are some widgets that make it possible to design bridges.
 - The top left corner is occupied by the material selector (Wood, Steel, Concrete), which would be highlighted when the user selects one. In the final app (if materials were implemented) this would have been replaced by easily distinguishable icons and/or colours.
 - Underneath this, there are some editor tools, for undoing an action, redoing an action, and the edge deletion tool. The undo and redo buttons would appear and disappear as required.
 - In the top middle of the screen, there are two values: the cost of the bridge and its load in percentage. The maximum load will only appear in simulation mode.
 - Finally, on the right side of this area there are some buttons for controlling the simulation. This will be a play button in editing mode and a pause in simulation mode, while a 'rewind' button will also appear in simulation mode that respawns the van at the left side and resets the bridge.
- *World settings:* This simple element controls the four variables that define the simulation. Sliders were chosen as the mutators of these values as they are a very intuitive representation of values that move within a range. This makes it possible to just look at the sliders and have a sense of the expected behaviour of the world. Furthermore, using sliders instead of e.g. a textbox entry makes the implementation easier as well, since a textbox would have to be validated; with sliders, this functionality is built-in.
- *Saves:* This component does what the name suggests; it provides a way to persist the bridge in the editor and load it back later. The creation of a save can be initiated by the 'NEW SAVE' button. Each save would have a name as well which will be editable by the user (as opposed to just being numbered in the picture). Each save will also display a graphical representation of the bridge, replacing the red boxes. Each save will also be able to be deleted by the button representing a bin. There will be no limit on how many saves can be stored. This element would be able to scroll on the y-axis to accommodate for saves that cannot fit on one screen. To persist the saves between sessions the browser's local storage will be used instead of a server-side user management technique.
- *Switch to solver:* This is the entry point to the next screen, the GA settings.

2) GA Settings

The interface is divided into several sections:

- WORLD SETTINGS:** Contains four sliders: GRAVITATION (set to 10), ELEVATION (set to 0), DISTANCE (set to 30), and WEIGHT (set to 20).
- GA SETTINGS:** Contains four sliders: MUTATION RATE (set to 10), MUTATION SIZE (set to 70), POPULATIN SIZE (set to 10), and SELECTION RATE (set to 20).
- CONSTRAINTS:** Contains two sliders: MAX COST (set to 50000) and MAX LOAD (set to 70%). Below these are checkboxes for USABLE MATERIALS: wood (checked), steel (checked), and concrete (unchecked).
- STARTING POPULATION:** A row of four buttons labeled 'save 1' through 'save 4'. Each button has a red square icon and a numerical value: 12000, 21300, 42000, and 11110 respectively.
- LOCK SETTINGS AND START:** A large blue button at the bottom right.

Figure 4 GA settings – first iteration

- *World settings:* Functions identically to the component above.
- *GA settings:* This element is quite similar to the previous, using sliders to adjust values; this time however, these correspond to the GA control parameters that were chosen to be adjustable in the initial requirements.
- *Constraints:* In this area, a user will be able to select the stopping criteria of the GA in terms of target cost and maximal load that the bridge shall have. In the other part of this section, under usable materials, the user will be able to constrain the algorithm to use only specific materials.
- *Starting population:* This area holds the saves previously made by the user, similarly to the one in the previous screen. However, this time they have a different function, which is to be able to select the starting population for the GA. This way the algorithm can be seeded, so that it does not start from randomly generated solutions but improves upon bridges created by the user.
- *Lock settings and start:* This button is the entry point for the next screen.

3) GA Viewer

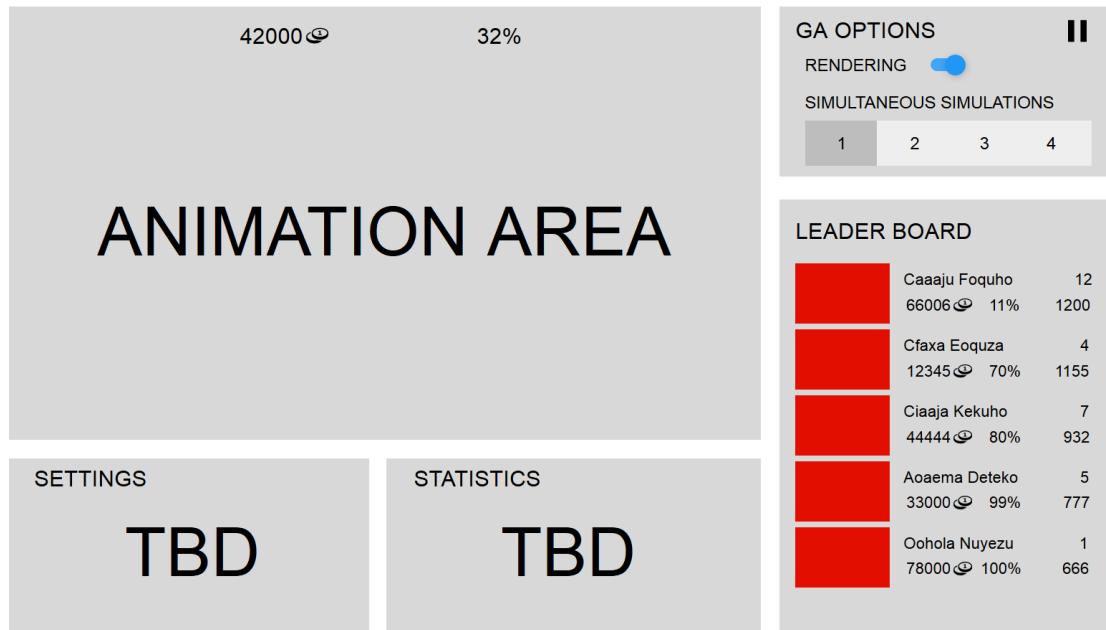


Figure 5 GA view – first iteration

- *Animation area:* This part is mostly identical to the one where a user can edit the bridges, except here the display is lacking the controls that the editor had. When simultaneous simulations are enabled then each game area would be scaled down to fit into a tiled layout.
- *GA options:* In this section, a user will be able to pause all simulations at once. Furthermore, a user will be able to select the number of simulations that run in parallel or turn off the rendering. Note that turning off the rendering does not stop the simulations; it simply saves some CPU time by not drawing the current state of the simulation to the screen.
- *Leader board:* This part of the screen is occupied by the best performing bridges. The red boxes again will be replaced by graphical representations of the bridges. The data displayed next to it is as follows:
 - Top left corner: the name of the bridge. This will be a string generated based on the genes of the bridge. Therefore, bridges with similar names would have similar genes.
 - Top right corner: the generation in which the individual was created.
 - Bottom right corner: score of the bridge.
 - Bottom middle: maximum load of the bridge.
 - Bottom left: cost of the bridge.

This layout was chosen as it indicates a clear hierarchy between high-performance individual and they can be easily compared both visually and based on their statistics.
- *Settings & Statistics:* This section was left mostly blank intentionally as this required some experimenting on what settings and statistics would work.

4.2 Second Iteration

The second iteration of UI design was made around the half point of the project, after the main game had a mostly functional prototype and the GA had its initial skeleton ready. At this point, some usability issues of the initial design became clear and many of the uncertainties of the initial requirements were filled in. Therefore, it made sense to revisit the UI design and adjust it to match the new requirements.

1) Main Game

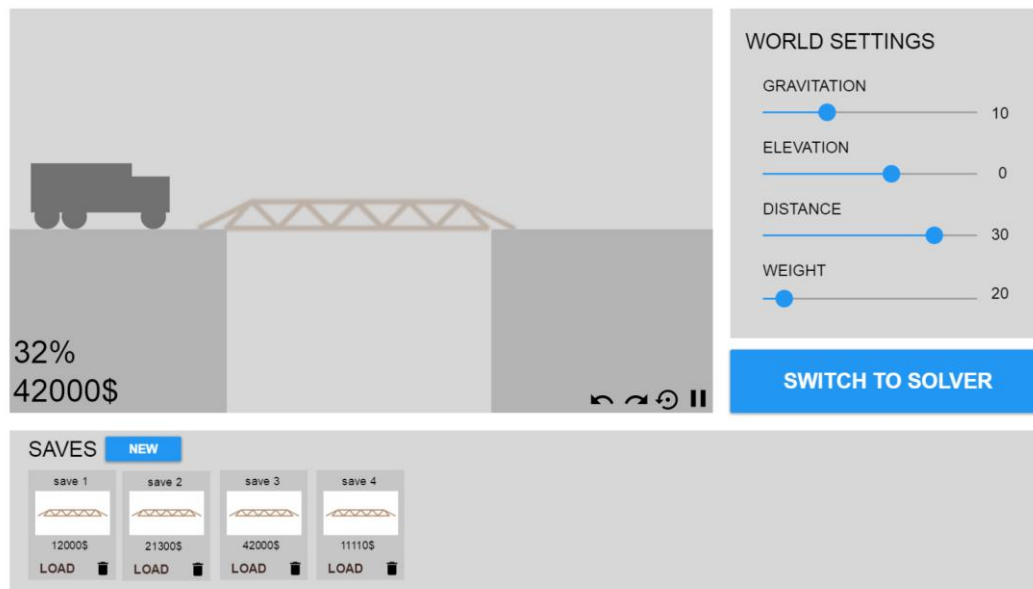


Figure 6 Main game – second iteration

In the initial prototype stage, the platforms were made to be rectangular static structures, which remained the same later on. When elevation for these platforms was implemented, it became clear that the buttons and data displays that were on the top of the game area were distracting, as with high elevations the van would pass by on that area, while the rectangular area under the platforms is completely unused. Therefore, it was decided to move this data and controls to the bottom of the screen.

Another small, but interesting change is that the 'NEW SAVE' button was moved next to the title of the section and shortened to simply 'NEW'. This change was made after implementing a prototype for saves. In testing this prototype, it became clear that if many saves were made the 'NEW SAVE' button would scroll out of view, which was counter intuitive, as making new saves would require the users to scroll the button into view again.

2) GA Settings

The screenshot displays the GA Settings interface, organized into four main sections:

- WORLD SETTINGS:** Includes sliders for GRAVITATION (set to 10), ELEVATION (set to 0), DISTANCE (set to 30), and WEIGHT (set to 20).
- GA SETTINGS:** Includes sliders for MUTATION RATE (set to 10), MUTATION SIZE (set to 70), POPULATIN SIZE (set to 10), and SELECTION RATE (set to 20).
- CONSTRAINTS:** Includes MAX COST (set to 50000), MAX LOAD (set to 70%), and a list of USABLE MATERIALS (wood, steel, and concrete) with checkboxes. Wood and steel are checked, while concrete is unchecked.
- STARTING POPULATION:** A row of four buttons labeled 'save 1' through 'save 4', each with a small bridge icon and a cost value: 12000\$, 21300\$, 42000\$, and 11110\$.

A blue button labeled 'LOCK SETTINGS AND START' is located at the bottom right of the interface.

Figure 7 GA settings – second iteration

This view was not changed at all since at this time in the project there was not enough experimental data that would justify any changes.

3) GA Viewer

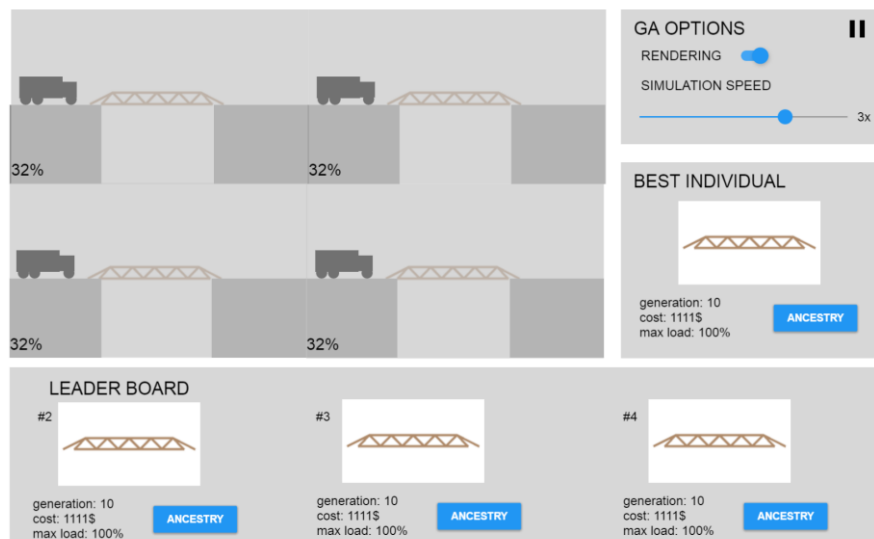


Figure 8 GA view – second iteration

In this view there was only a single, yet fundamental change compared to the first iteration. This is the removal of Settings & Statistics and replacing it with the leader board. This change was made, as it became clear in this stage that bridges would be many times wider than their height, therefore, the bridges would only occupy only a small portion of the space that was assigned for this purpose. Furthermore, the rest of the data that is displayed such as generation and cost would sometimes be too long and overflow. In essence, the design looked good in theory, but it was technically not feasible.

The reason why the leader board would be the replacement for settings and statistics is that it was realised that these other elements were not necessary. For settings, the only option (simulation speed) could be fit in the options panel in the top right corner and statistics was not required as the only pure numerical data could be displayed there, which is the very thing this project is trying to avoid.

An unusual design decision is to separate the first in the leader board from the rest. The reason why this separation is made is that the best individual is always the most important individual in a GA and therefore this separation is only there to show its significance. However, since this is an unusual design, it was decided to include a numbering for the individuals representing their position on the board, to avoid any confusion.

4.3 Evaluation of the UI Design

To evaluate the design of the UI prototype Jacob Nielsen's ten usability heuristics was used [43]. These heuristics provide some rules of thumb that were created by analysing common design mistakes in various systems in order to help developers avoid these mistakes.

1. *Visibility of System Status:* This rule states that users should always be informed about what is happening inside the program through appropriate feedback. The three key states in the app are easily distinguishable and therefore the user is always aware of the status. One shortcoming might be the lack of progress display in the GA view, which should be addressed in the final design.
2. *Match between system and the real world:* This rule is about using a simple language that users can understand. As far as the simulation is concerned, this rule is accomplished; however, describing the various parameters for GAs requires some specific terminology. To aid users in understanding this, help sections shall be provided.
3. *User control and freedom:* This rule states that 'emergency exits' should be available at all times, as users tend to make mistakes. This also includes undo/redo functionality. Undo and redo functionality is supported in the editor, but in other places this is lacking. This as well, should be addressed in the final design.
4. *Consistency and standards:* This rule stipulates to use platform standards. Since the web does not have any standards this might be hard to achieve, but because it was decided to use the Material Design specification, this can be considered complete.
5. *Error prevention:* This rule is involved with preventing the user from making errors. As the app uses sliders that can only move within accepted ranges this is not a concern. In other places, such as when fitness and selection methods are edited directly, appropriate measures will be taken to prevent the user from causing unrecoverable errors.
6. *Recognition rather than recall:* This rule requires that users should not be required to remember information while using the system, but rather enabling them to understand the system immediately by recognising similar structures. Since the design language and its metaphors that will be used in this app are abundant across different apps, it will be easy for users to recognize the functions of the design elements. This is especially true for icon buttons, such as the bin, undo and redo as these are used consistently to represent the same functionality.
7. *Flexibility and efficiency of use:* This rule is about making the program easier to use for experienced users by providing shortcuts. To accommodate for this, all settings shall be stored, including the world settings, GA parameters and selection/fitness functions.
8. *Aesthetic and minimalist design:* This heuristic revolves around placing only the required, most used information on any given screen. This is probably the single rule that the proposed design breaks the most, as it is very information dense. Unfortunately, there is not an easy solution for this issue, as a complete redesign would be required to reduce the information shown in any given screen.
9. *Manage errors:* This rule states that errors should be understandable and they should have a constructive suggestion to recover from them. As mentioned earlier, validations make it hard for errors to happen, but when they do these will be handled appropriately as the rule suggests.
10. *Help and documentation:* The last heuristic is states that it is better if the system explains itself but if it is required, a documentation should be made available. Since GAs are somewhat hard to understand built-in help sections shall be provided.

Even though the second iteration of the UI design falls short on many of these heuristics, most of the problems can be fixed by adding a few extra functions, buttons or other elements. However, to fix these issues correctly it was decided to carry on with the implementation, as the best way to achieve a great user experience is by experimenting with different layouts.

5 SOFTWARE IMPLEMENTATION

5.1 Key Implementation Decisions

5.1.1 Back-end: Node.js & Express

Node.js [44] is a web framework that can be programmed in JavaScript (JS), made possible by Google Chrome's open-source JavaScript engine. Node was designed with a non-blocking I/O model in mind, that is, input and output operations (such as networking or disk access) will not block the program, but rather, a 'callback' will be made when these operations are completed. This is also known as asynchronous programming, which makes it easy to write high throughput web applications without explicitly considering concurrency issues. This paradigm is very different from other web scripting languages such as PHP or ASP.NET, which use a blocking model for I/O.

For many years, developers had to master (at least) two separate languages to make one application, as the front-end of the program is only programmable in JS, while there are countless available languages to choose from when it comes to the server side. This has two fundamental consequences. Firstly, when the two sides want to communicate they have to choose a common ground, a communication protocol. Classically, this protocol was often chosen to be XML, however, this data encoding is not native to any particular platform, and thus had a considerable overhead when it came to parsing it. The second issue is that when the two sides are developed side by side, usually the two platforms would require a different way of thinking, thus making it harder to switch between the two tasks.

Node solves these problems in a very elegant way by making available the client side language on the server of the application, which is the main reason why this platform was chosen. A further reason for this decision is that, initially it was planned to offload some of the simulation to the server in order ease the CPU load on user's machine. However, it was later realised that game engines that support both Node and browsers do not exist. Nevertheless, Node was still kept as the back-end as this way the overhead of learning a new language just for the back end could be avoided.

One issue still remains, which is the fact that Node only has a low-level API, which makes development considerably slower. This is a conscious design decision from developers of the platform. To resolve this issue every Node installation comes with the command line tool npm, the Node Package Manager. This tool is a central repository of libraries, which makes it easy for developers to publish open-source modules that can be installed and reused with a single command line instruction. This infrastructure made it possible for Node to grow rapidly, making npm the biggest ecosystem of open-source libraries.

One of the most commonly used modules is called Express [45], a module that is going to be used in this project as well, makes it significantly easier to make dynamic web applications such as this one. This is made possible by the simple API and expressive routing system.

Even though the back-end was not customised as extensively as it could have been in the end, these are still key components of the system that define the rest of the app's infrastructure to some extent.

```
1.  var express = require('express');
2.  var app = express();
3.
4.  app.get('/', function (req, res) {
5.    res.send('Hello World!');
6.  });
7.
8.  app.listen(3000, function () {
9.    console.log('Example app listening on port 3000!');
10. });
```

Figure 9 Hello World with Node & Express [45]

5.1.2 Front-end: Angular

The design of HTML, the mark-up language of the internet, was made over 23 years ago in 1993 with the intention to make static documents interconnected on the World Wide Web. Hence, it is not surprising that the designers did not have dynamic applications such as this one in mind when writing the specification. Nevertheless, HTML still is the core of the internet, but it carries some shortcomings, that stems from the legacy that it has, which makes it difficult to create rich client applications even after four major revisions of the initial specification.

This is the problem that Angular tries to solve, by extending HTML's syntax to accommodate for modern web applications that look and feel as if they were regular desktop programs. The way this is achieved is by providing a Model-View-Controller like architecture, a common software pattern used for building graphical user interfaces, which automatically manages the propagation of data between the different components of the software. In the next few paragraphs, some of the core features of Angular will be explained in some detail, with examples from this project.

Arguably, the most important feature of Angular is directives. Directives are the means for Angular to extend HTML's syntax. Essentially this feature makes it possible to define new tags that can be used in the HTML template of the site. What makes this feature really powerful is that different directives can pass data between each other freely. Furthermore, a well-designed directive is independent and reusable, which makes it possible to make libraries that consist of reusable components. One such library that was used in the project is Angular Material. This library is an implementation of MD that uses the Angular framework to provide components that adhere to the MD specification. Every slider, button, even icons in the app are displayed by using one of the directives provided by this versatile software package. Moreover, the game uses this feature extensively to abstract away some core features. How this feature is used is explained in more detail in [Section 5.3](#).

Another powerful feature of this library is two-way data binding. Yet again, this feature is made possible by directives. What this it accomplishes is that it keeps the Model (i.e. data layer) in JS synchronised with the View (i.e. the graphical representation) automatically. Take this code snippet for example (from the project's source without styling attributes).

```
1. <span>GRAVITATION: {{settings.gravitation}}</span>
2. <md-slider ng-model="settings.gravitation" step="1" min="5" max="50" />
```

Figure 10 Two-way data binding snippet

This is all the code necessary to make a complex slider element that adheres to the MD standards. In the second line, an Angular Material directive is used (`md-slider`) to create the input component. The `ng-model` attribute of this tag tells the directive that it should 'bind' to the variable `settings.gravitation`. What this makes the bound value in JS always reflect the value of the slider and vice-versa.

Note the unusual double brace syntax in the first line, within the `span` tags. This tells Angular that it should interpolate the code inside the braces, and replace it with the value. What this accomplishes is that the value displayed next to `GRAVITATION:` will always be synchronised with the slider and the internal representation. This notation can also be used to pass data to directives. For example, if the maximum value for the slider was dependent on another value then it could be passed in within the double brace expression. This is only a simple example, but this expression is capable of much more, as it supports a large subset of JS expressions that can be evaluated with it.

Another great feature of the system is Dependency Injection (DI), which battles JS's one key shortcoming, which is the fact that everything is in global namespace within it. DI makes it possible to share common code between different parts of the application without having to declare them globally, making a modular coding style straightforward. To use this feature, one only needs to declare the name of the required dependency in the parameters of the function and the framework will handle the rest; that is, it

will find and inject that code into the function's scope. This feature also promotes reusable code, as this is one of the main ways to use Angular libraries. Furthermore, this also makes it possible to download JS files on demand, which was previously almost impossible, or at least it would require a lot of work.

The project uses this feature extensively, as this is the way code is modularised. Each class is registered as a module with the same name that can be injected in any other part of the code. This can be demonstrated with a short code snippet.

```
1. .service('AddNodeOperator', function (AbstractMutationOperator) {  
2.   class AddNodeOperator extends AbstractMutationOperator {  
3.     constructor(rd) {  
4.       super(rd);  
5.       ...  
6.     }  
7.   }  
8.   return AddNodeOperator;  
9. })
```

Figure 11 Dependency Injection example snippet

In this snippet, a new injectable object is defined called `AddNodeOperator` (one of the mutation operators). To create this class the `AbstractMutationOperator` is injected, which is then in turn extended to form the `AddNodeOperator` class. Note that there is no need to specify where the other module is located, which makes it easy and intuitive to create well-structured code. How Angular handles this is that it registers a so-called service, whose name is the first parameter ('`AddNodeOperator`' string), and its value will be the value that the function in the second parameter returns, in this case the class `AddNodeOperator`. Whenever this service is injected as a variable, it will take the same value as services are treated as singletons.

Angular is not the only front-end framework however, that makes it easy to create dynamic web applications. When it came to choosing this property of the system, many other tools were evaluated, such as React [46], Backbone [47], and Ember [48]. However, the final choice was Angular as it is the most mature system (over five years old), with extensive support available in forms of tutorials and blogs. Furthermore, there are a staggering 164,714 questions on StackOverflow [49] (as of 11/04/16) tagged with Angular, therefore if any problems would arise during the development cycle it is very likely that the issue could be resolved by finding the answer for it in this invaluable resource. There is not a contender in this aspect from the considered tools, as the one with the second most questions is Backbone with only 19,564 questions [50] (as of 11/04/16), with the most voted question being 'What is the purpose of backbone.js?'.

5.1.3 Development Tools: Babel, Gulp & Yeoman

If you are familiar with JS, you might have noticed that classes, as the one presented above, are not widely supported currently. This is in fact a feature from 'future' JS, that is, classes are only part of the ECMAScript 6 (ES6) specification [51], which is the standardised version of JS. The proposal for the 6th version of this specification has been accepted in June 2015 [52], which means that the semantics can be considered final. However, the issue is that browsers are only implementing a subset of the proposed features as of now [53], which makes it hard to reliably use the new syntax.

Since, the current, prototype based inheritance system in today's JS is sometimes confusing, and bug prone, the new class semantics introduced in the proposal is a breath of fresh air, which makes using it very compelling. However, browser incompatibility is something that would greatly diminish the accessibility of the application. To solve this issue, it was decided to use Babel [54] a 'JS of tomorrow to JS of today' compiler (or transpiler), that is, this tool transforms ES6 code to JS that most browsers can understand. This makes it possible to use powerful features such as array destructuring, string templates, classes, arrow functions, etc.

However, this tool introduces a new issue, which is the fact that code needs to be compiled before it can be ran in the browser, which is usually not the case with interpreted languages. Doing this with a shell command every time the code is changed would be rather tedious, therefore a new tool is introduced to

the system called Gulp [55]. Gulp is a build system (much like make or Gradle) which makes it possible to automate repetitive tasks in developing a web app.

This tool was used to automate three key tasks:

- *Development server with watchers:* Executing the shell command `gulp serve` accomplishes the following.
 - ES6 files are transpiled to JS files.
 - Dependencies (such as Angular and Angular Material) and the transpiled project files are injected into the main HTML file.
 - A local development server is launched that serves up these files.
 - Watchers are set on all project files, that is, if any of the files change, the compilation process will be ran again and the browser will be instructed to refresh.
- *Automated build:* This task is quite similar to the previous, but after JS files have been transpiled, they are also concatenated and minified, among other optimisations. This is a common pattern in modern web projects as this minimises the load on both the server and the client. Concatenation accomplishes this by lowering the number of HTTP requests the browser needs to make to load the page, while minification obfuscates and compresses the files, therefore lowering the total number of bytes in transit.
- *Automated deployment:* This task first builds the project files with the above method, and then it takes all the files and pushes them to Heroku, the hosting service of choice. This way the whole deployment process that could be quite tedious sometimes, is reduced to a single command, `gulp deploy`, making the release of the software almost instantaneous as it runs in under 30 seconds.

As you can see, the program passes through a complex transformation pipeline that might take long to implement. However, this is not the case, made possible by yet another tool, namely Yeoman. Most languages have an official or sometimes multiple dedicated IDEs that make writing code on that specific platform easier. One of the key features that make this possible in IDEs is code generation (for example new Class generation in Eclipse, or the ability to generate complete starting projects in Android Studio). Since JS does not have a dedicated IDE that provides this functionality, Yeoman tries to fill this gap by means of a command line tool.

This command line tool makes it possible to create code generation modules. To start off this project, the generator called AngularJS Full-Stack generator [56] was used, which is a Yeoman generator. Using this tool gave the project an initial skeleton, with most of the Gulp automation tasks already built-in. However, Yeoman is not only for kick-starting projects, but it offers help in later stages as well by making smaller code snippets as well, such as creating a separate file for a new Angular service or directive and its unit tests. What this accomplishes in practice is that the whole project will adhere to the same coding standards and best practices as generators enforce this to some extent.

At first, this web of tools, and tools that create tools might seem complicated, but this is not the case. This is only the result of following the Unix philosophy, which is summarised well by McIlroy: “Make each program do one thing well” [57]. Since each of these programs only accomplishes one task, it makes it possible to use them together to achieve complex functionality without requiring the tools to be monolithic. Furthermore, this design philosophy also makes them easy to learn and use since they do not boast with an intricate interface or complex commands and APIs.

5.1.4 Game Engine: Turbulenz

To avoid implementing the physics simulation from scratch, a game engine was required. Several JS game engines were evaluated that could accomplish this task, such as Phaser [58], MelonJS [59], and Turbulenz [60], among others. Even though Phaser and MelonJS are much more popular than Turbulenz (based on StackOverflow questions), it was still decided to use Turbulenz to implement the game. The reason for this is that originally, it was planned to ‘uplift’ the simulation to a 2.5D one in later stages of the

project, and Turbulenz was the only engine that would support both 2D and 3D including a physics simulation in both. This choice might have been the biggest mistake during the project, which will be further discussed in [Chapter 5.6](#).

5.2 Project Management

To manage the creation of such a large project, adapting a software engineering methodology is crucial. The methodology adopted was Iterative and incremental development (IID). The reason for adopting it was that initially very clear requirements were set that were not expected to change considerably. Usually this would warrant a simpler methodology, such as Waterfall. However, some changes were expected that would result from testing the incremental builds of the software. This is the exact scenario that IID excels at, as it can be sometimes quite rigid when there are clear requirements; however, it is capable of adopting changes in these that result from knowledge gained through working prototypes.

The way this worked in practice is the following. Using the initial requirements a checklist was made that is ordered with respect to the priority of the task. To track the completion of this checklist, GitHub's issue system was used. Every week a few of the most important tasks were selected for completion. At the end of each week, the progress was reviewed, and if it was required, the checklist was updated to reflect the changes in requirements. A large part of the changing requirements was due to my peers, as I would ask their opinion of the current state of the game. Using their feedback, I would change the system according to their suggestions and test it again. This process was very beneficial to the software as many usability issues were resolved resulting from this (quasi) peer-review method.

5.3 Internal Design

5.3.1 Simulation

Another crucial step in implementing such a large system is a good internal design. This became very clear when in the early prototype stage everything was contained in a single file, which became unmanageable quickly. However, this stepping-stone was needed as the insights gained through it were used to create the internal design of the software.

Probably the most important part of the system is the physics simulation. Three components were identified where this would be required:

1. *Save display*: This is the most simplistic component only showing a static image of the bridge to provide some context for the saves. It was determined that this component should be designed in a way so that it could be used to display the leader board items as well.
2. *GA display*: This component would be used to evaluate the solutions by means of the physics simulation. A key requirement for this component is that it should be able to run side-by-side other GA displays on the same page, and that it has a simple interface, that can be used in the GA to start simulations using a specific chromosome.
3. *Main game*: This is the most feature rich component, as it has to include the simulation and it has to handle user input to make the drawing of bridges easy.

Clearly, much of the functionality in these components is identical; therefore, replicating code between them would be pointless. To mitigate this issue these tasks were organised into distinct classes that extend the same abstract classes. This class hierarchy is shown in Figure 13, on the next page.

- **BaseGameDriver**: This class is the base for all other Drivers. It includes functionality such as a basic game loop, initialisation of some variables and some parts of the Turbulenz Engine.
- **SimulationDriver**: This class is an extension of the **BaseGameDriver**. It adds functions that handle resource loading (such as textures), and overrides some parts of its superclass to initialise other parts of Turbulenz and extend its drawing capabilities, to make the bridge simulation possible.

- **BridgeDisplayDriver**: This class extends **BaseGameDriver** in a different way, by exposing an interface to set the bridge that is to be displayed, and overriding the main loop to draw only a single time, since the displayed bridge is not updated.
- **EvolutionDisplayDriver**: As the extension of the **SimulationDriver**, this class includes the bridge simulation but it extends it in a way that exposes an interface that allows it to simulate different bridges. It also exposes functions that can change the simulation speed, or even pause it.
- **MainGameDriver**: This extension of the **SimulationDriver** adds all the input handling and buttons to create the editor, as well as an interface that makes it possible to extract the current bridge from the editor to make saves.

However, none of these classes is concerned with displaying their content in the browser. To do this another layer of abstraction was created, by means of an Angular directive called Turbulenz canvas (**tz-canvas**). This directive is capable of creating a **canvas** element that is initialised in a way so that Turbulenz Engine can draw on it as well as linking the engine with one of the **Drivers**. To make this linking process modular a directive was created for each of the concrete classes that just pass the desired **Driver** to the **tz-canvas** directive. How this is used in the code is the following:

```
1. <tz-canvas main-game></tz-canvas>
```

Figure 12 Matching the presentation layer with control logic

What this abstraction achieves is that the HTML template of the app will match the content that is displayed. Furthermore, it separates the control logic of the games from the presentation, which makes the source of the application more manageable.

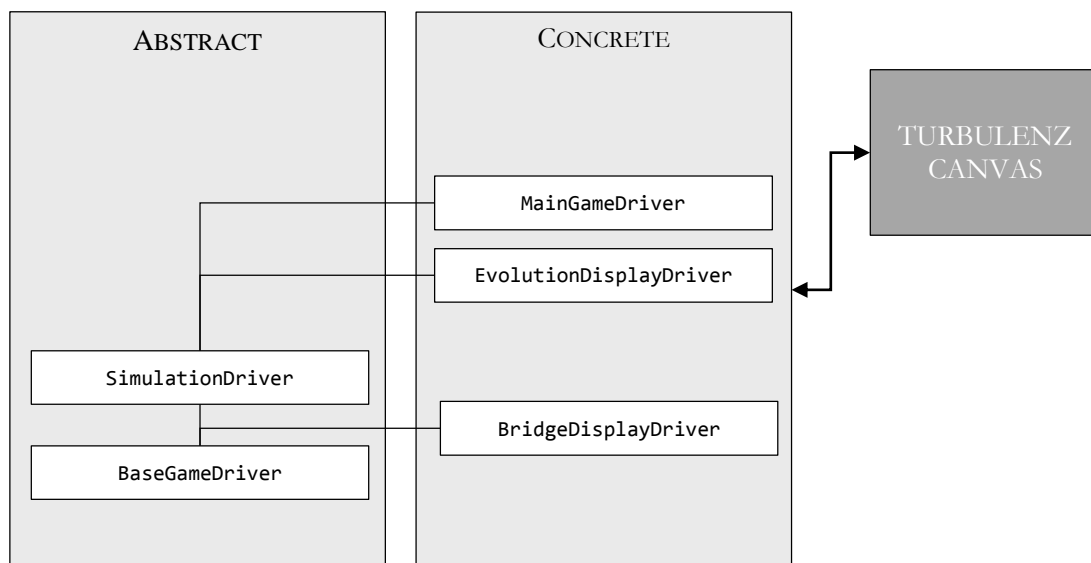


Figure 13 Simple class diagram of display drivers

5.3.2 GA

One of the strengths of GAs is the fact that it has many components that can be changed to better suit the problem at hand. Therefore, it is crucial to design these components correctly to make sure that the algorithm can actually solve the problem. The algorithm that was designed can be best explained using a flow chart:

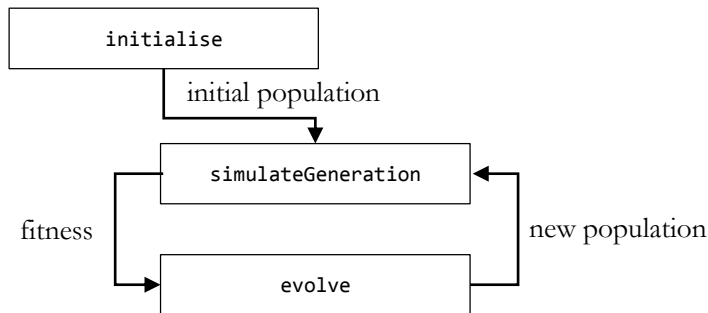


Figure 14 GA flow chart

- **initialise:** In this step, basic requirements for the GA are instantiated, such as mutation rate, population size, etc. After this, it generates an initial population by generating a bridge that connects the two sides and then mutating it 50 times. After the drivers load, it starts the simulations.
- **simulateGeneration:** In this step, the current population is evaluated by the simulation. To do this every member of the population has to wait for a driver to become free, that is, for it to finish simulating the previous individual. In the beginning, all drivers are free. After all simulations finish, the fitness values are calculated (using the user's fitness function or the default one) for each individual and passed to the next step.
- **evolve:** Using the fitness values the best bridges are selected (using the user's selection function or the default tournament selection), and mutated, to produce a new population that is passed back to be simulated again. This loop continues until the user stops it.

There are two critical differences from the simple GA algorithm that was shown in Figure 1. The first one is that the stopping criterion was removed as the algorithm can be stopped at any time by the user. Secondly, that there is no crossover operator. The reason for this is that, since bridges look relatively similar even after a low number of generations, the existence of a crossover operator would only add extra complexity without improving the solutions.

5.4 Implementation of System Components

5.4.1 Simulation

To implement this complex simulation a similar Object Oriented approach was used as with the design of the drivers. To achieve this task the following components were used:

- **Bridge:** This class is the internal representation of the bridge, that holds the nodes and edges that form it. It also keeps track of its load by analysing the constraints that are created while adding the edges to the world.
- **BridgeEdge:** This class represents the pieces that form the bridge. To make it possible that these edges can rotate freely around the nodes where they are connected (also known as revolute joint), two bodies are created in this class. An inner body that only interacts with other edges and is shaped as a parallelogram, and another one that interacts with the rest of the world and is shaped

- as a rectangle. From these two bodies usually only the outer body is visible (made possible by a sprite), however in debug mode (shown in Figure 15) the inner bodies are visible as well.
- **BridgeNode**: This class does not have any visual objects tied to it, it merely acts as a datastructure that holds x and y coordinates.
- **car**: The car or van is made up of four rigid bodies, the main body of the car and three wheels. The body is made up of two smaller rectangles (as shown in Figure 15). The wheels are connected to the main body of the van using a line constraint. To move the car, the angular velocity of the wheels are set on every update instead of applying an impulse to the body of the car. This achieves an acceleration that looks natural as opposed to the former method.
- **AbstractSensor**, **CrashSensor**, **FinishSensor**: To detect the state of the car in the GA sensors are used. These bodies, do not move or interact with the rest of the world, but they are capable of detecting when specific bodies touch them. Since, the only difference between the car finishing or crashing is the place, it made sense to make the sensor functionality abstract, and using concrete classes (**CrashSensor** and **FinishSensor**) to add the position to the equation. The bodies that are created by these are not visible by default, but it is shown in debug mode (**CrashSensor** is the rectangle between the two platforms and **FinishSensor** the thin rectangle on the right side).
- **Platforms**: This class creates the two static bodies on the side, it was designed so that the elevation and distance can be directly adjusted by constructing it with different parameters, to support the world settings panel.

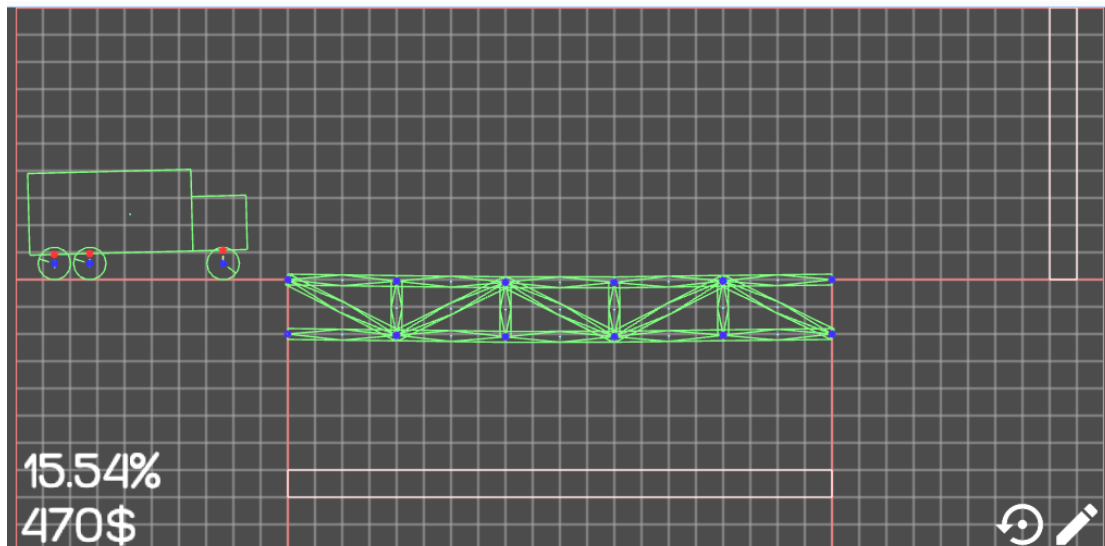


Figure 15 Main game debug mode

5.4.2 GA

A key aspect of a GA is the data structure that is being used to represent the problem. This was chosen to be a structure that is similar to a graph (hence the names **BridgeEdge** and **BridgeNode**). The way this is represented in JS looks like the following:

```
{
  nodes: [
    new BridgeNode(0, 0),
    new BridgeNode(4, 0),
    ...
  ],
  connections: [
    {a: 0, b: 1, e: new BridgeEdge(material, true)},
    {a: 1, b: 2, e: new BridgeEdge(material, true)},
    ...
  ]
}
```

Figure 16 Bridge graph representation

The `nodes` part of the object is quite straightforward; a list of `BridgeNodes` is created where the nodes receive the x and y coordinates as parameters in their constructor. On the other hand, the `connections` part is not as straightforward. The two properties `a` and `b` represent the index of the node in the `nodes` list, while the `e` property of this object contains a `BridgeEdge` that receives a `material` and a Boolean that shows whether the van should collide with it. Therefore, in the example the first `Edge` connects the 0th and 1st index `Nodes` (i.e. (0, 0) and (4, 0)) using some material and the van will collide with this edge.

The reason this somewhat complicated structure was chosen is that this way the connections do not need to obtain a reference to the nodes they need to be connected to, which makes it possible to create these bridge definition objects in one go. However, this choice might not have been the best one, as it caused more problems than it solved; and since this structure was used throughout the software, refactoring it would have taken a considerable amount of time that was not affordable.

As discussed earlier the crossover operator was removed from the algorithm, therefore the mutation operators have to be implemented in a way so that any bridge can be created by only using them. These operators again used an Object Oriented approach, as each operator is a class that extends from the `AbstractMutationOperator` that contains some common functionality used across other operators. Mutations do not always succeed, as there are many constraints on the bridge. If any of these are violated then the mutation is applied again for a maximum of ten times or until it succeeds. One such constraint is that no operator can change the edges that connect the two sides (e.g. removing such an edge), as this would never lead to better solutions. Another constraint that is validated on most operators is that no edges can intersect apart from the ends. The following operators were created:

- **RemoveNodeOperator:** This class mutates the target individual by removing a node, and any edges that are connected to it. If this results in a disjoint bridge, then the mutation fails.
- **RemoveEdgeOperator:** This operator removes a random edge from the bridge and any nodes that are not connected to the rest of the bridge by edges. If this results in a disjoint bridge, then it fails.
- **AddNodeOperator:** This operator adds a node to the bridge that will be connected to a random other node. This operator will fail if the new node is inside the platforms.
- **ConnectNodesOperator:** This operator connects two previously unconnected edges.
- **MirrorNodeOperator:** This operator mirrors a node around the middle of the bridge and connects the new node to an existing other node. The operation fails if the node already exists, if the new node cannot be connected to an existing one.
- **NudgeNodeOperator:** This operator moves an existing node in either direction by at most 2 spaces. It fails if there is a node in that point already, or if any connected edges became longer than the maximum length allowed.
- **MakeTriangleOperator:** This operator connects two edges that already share a node. If the two edges cannot be connected, because the other ends are too far it fails.

Although these constraints introduce bias into the search, experiments (with and without constraints) have shown that it is beneficial, as it avoids exploring solutions that are most likely unviable.

5.5 Detailed Description of the User Interface

5.5.1 Main View

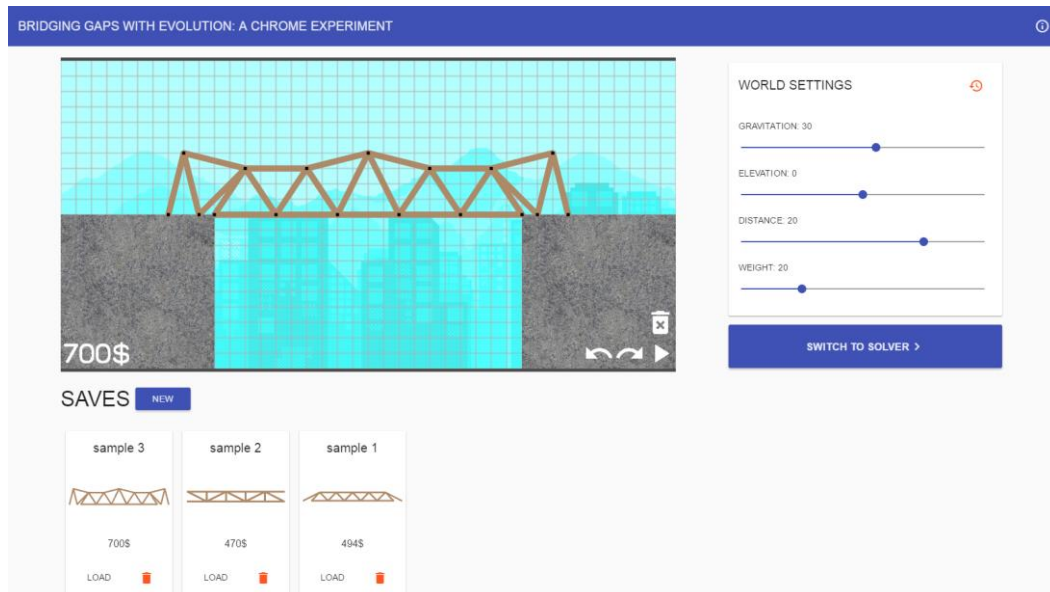


Figure 17 Landing screen

As you can see, the final design of the app follows the second design iteration very closely. However, many usability issues were resolved, that were identified previously in [Section 4.3](#), as well as ones that were identified later by peer-reviews.

- *Visibility of System Status:* It was believed that the screens are easily distinguishable, however, some peers pointed out that this is not the case when they first used the software. To alleviate this, status bars were added on the top that clearly identify the page that is being viewed.
- *User control and freedom:* Users often mentioned that sometimes they want to start building the bridge again from scratch; therefore, a 'clear all' button was added as an 'emergency exit'. This button would clear the bridge, but not the undo stack, therefore if it was accidentally clicked it can be reverted. Furthermore, users also noted that sometimes they want to revert the settings to the defaults after trying out new ones. To make this possible a 'revert settings' button was added in the world settings panel.
- *Consistency and standards:* As Ctrl-Z and Ctrl-Y shortcuts mean undo and redo respectively in most (if not all) software, these shortcuts were added to the editor.
- *Help and documentation:* To provide a quick access help section on every screen, the newly added status bar was used. On every screen, a small 'information' button was added that creates a small dialog displaying a short explanation of the features that area available on the given screen. This button is always located on the right side of the bar to make its access consistent.

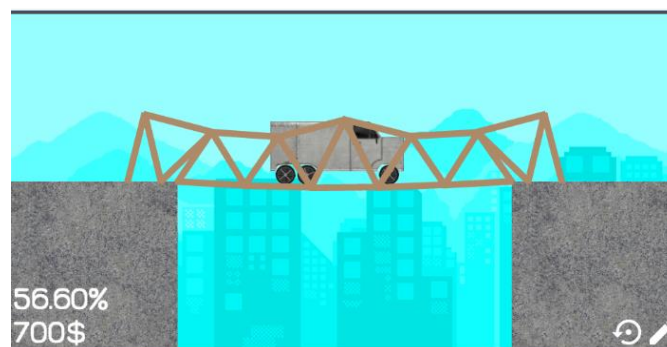


Figure 18 Running simulation

5.5.2 GA Settings Dialog

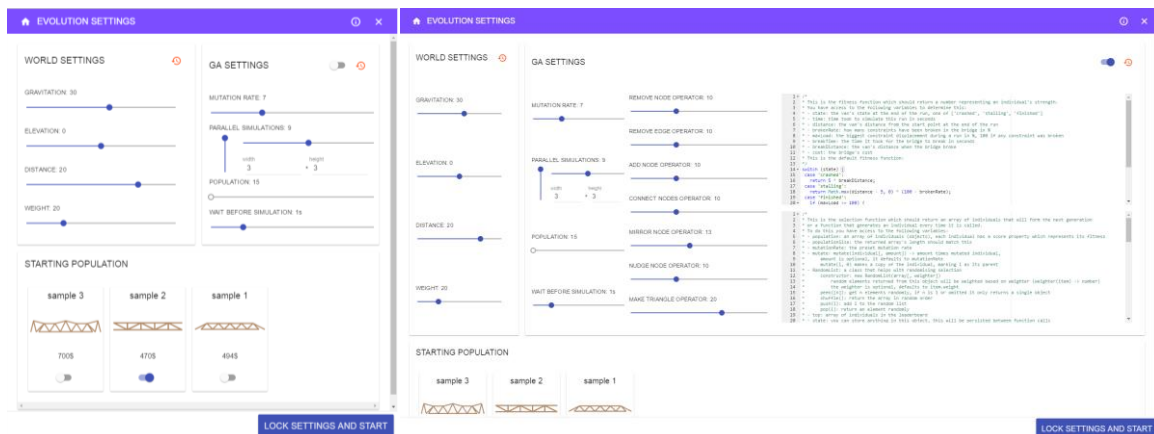


Figure 19 GA Dialog with and without the advanced options enabled (different aspect ratios)

Even though the world settings and the starting population panel did not change all that much, the GA settings panel has been completely redesigned. The only two things that remained the same are the population size and the mutation rate slider. The added new features include the ability to select how many rows and columns of simulations there will be as well as adjusting the wait time before simulations start, which is useful if the user wants to evaluate the bridges before the simulation begins.

When the advanced mode is enabled (toggle next to the ‘revert settings’ button) an individual slider appears for each mutation operator that adjusts their respective weight. The advanced mode also includes two code editor panels on the right (with syntax highlighting) where the user can enter their own selection and fitness functions. The default text in these editors includes comments, which show the variables that can be accessed, and it explains the requirements that functions should fulfil. Since the selection function is slightly more complicated, there are four examples of simple selection methods.

Lastly, some escape routes have been added here as well. Since this screen appears as a dialog, a close button was necessary, which was placed on the top right side like on most platforms and applications. As in the main view, the information button next to the close button will display a modal that shows information about the features of the current screen. Furthermore, a home button was added to the left side of the status bar, to provide an easy way back to the main view, to comply with the *User control and freedom* heuristic. The status bar this time was chosen to be a slightly different hue to indicate that this is not a full-fledged view but only a dialog that can be closed to return to the previous view.

Since the user’s input decides how well the GA will perform, some validations are necessary. Firstly, all sliders have been set so that only valid values can be entered with them. However, if some values are likely to cause bad performance a warning will be shown when the start button is hovered. Secondly, the number of parallel simulations can be entered in a textbox as well; therefore, these have to be validated. Probably the most error-prone input field is the function editors. To validate these, the functions are continuously compiled in the background and they are executed using some test values. If any of these validations fail, the start button will be disabled, showing a message explaining the exact reason it was disabled so that the user can easily recover and continue to the GA view. This error handling method was designed so that it follows the guidelines laid out in the *Manage errors* heuristic.



Figure 20 Input validation

5.5.3 GA View

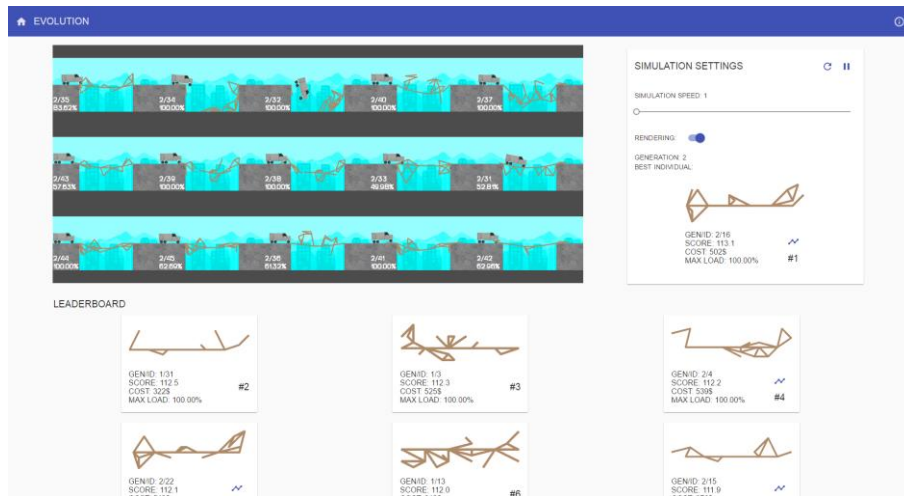


Figure 21 GA View

There were a few notable changes made in the design of this screen. Firstly, a status bar was added here as well. It includes both the home button to return to the main game (complying with *User control and freedom*), as well as including a standard information button, that the same way as before creates a modal that explains the UI elements. Another change that was made to accommodate for the *User control and freedom* heuristic is that a refresh button was added next to the pause button, which initiates the GA settings dialog. However, it only pauses the simulations (if necessary), so if the user would rather keep the GA settings and return to it they can do that by just closing the dialog.

Another important visual difference compared with the design is that the panels that make up the leader board has been slightly reorganised. Firstly, the big ‘ancestry’ button was replaced with a much smaller ‘timeline’ icon button, which only appears if the individual actually has ancestors. Secondly, the numbering has been moved next to this button as it fits much better next to the smaller button.

When the user clicks on one of the ancestry buttons, the dialog below is displayed. This shows a bigger picture of the individual, as well as showing how it evolved over time. The user can use the slider or left and right arrow buttons to navigate between different generations. Why this is particularly interesting is that parent and child can be compared with each other in fast succession in a visual way. This makes it possible to see what individual mutations do (especially if the mutation rate is set to 1), and how sometimes bad solutions lead to top-scoring ones.

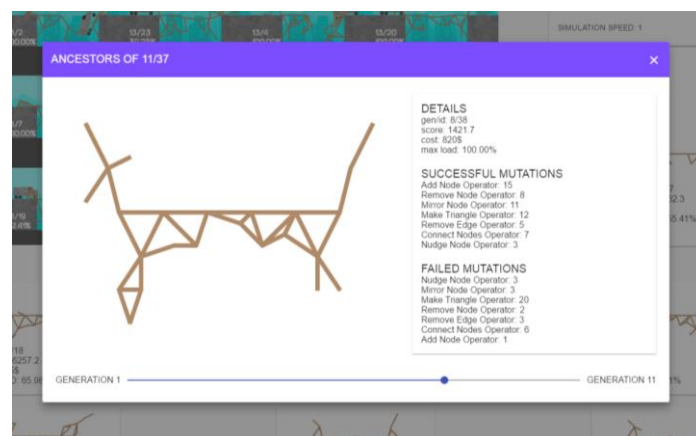


Figure 22 Ancestry dialog

5.6 Problems Encountered

5.6.1 Turbulenz

The earliest problem that was encountered was concerned with the game engine, Turbulenz. The core of the issue is that this engine was chosen based on its features and not its API and support. At first, I thought that it was just a matter of getting used to the library. It was only much later, when I realised that the API was designed with (a team of) expert game programmers in mind. There were many complex concepts about game programming that I had to learn to accomplish the most basic functionalities. This issue would have been much less impactful if the documentation was at least somewhat more detailed or if there was some other online support. Unfortunately this was not the case, therefore countless hours had to be spent stepping through the internals of the engine to understand how some functions work.

Another problem was that the engine came with an SDK (software development kit) that at first seemed very promising, as it included a build system. It took some time to learn how to use it, but I was convinced that this would be useful later on. Only when it came to adding other components to the system, such as Angular, did I realise that the provided build system is barely customisable as it is optimised for games that are to be released through the gaming platform of the creators of the engine. This led to discarding the use of the SDK, changing the complete structure of the project to the Angular Full-Stack generator's, and changing its Gulp build system so that it supports the engine as it has special requirements. This took well over two weeks, which could have been spent on working on the actual implementation of the game. Looking back, I do not regret switching to Gulp, as it allowed me to use and learn advanced ES6 features, which resulted in code that is much easier to understand than regular JS.

Despite these difficulties, the engine was kept, as I was too stubborn to give up using it. If I had the chance to start again, I would pick Phaser as the game engine as it has all the capabilities that were used from Turbulenz. The difference is that Phaser has much more examples, a cleaner API that was designed with newbie game programmers in mind, and it also has a detailed documentation complete with many examples. Furthermore, Phaser is still under active development as opposed to Turbulenz, therefore getting help outside of the documentation would have been easier as well. However, Phaser did not support 3D, which is the reason it was not chosen, as the original plan was to uplift the game to 2.5D at some point.

The last issue was not concerned with Turbulenz, but it is related to the game. It was assumed that any number of canvases could be placed on a page; this is why the saves and leader board items use a full-fledged `tz-canvas` to display their content. Unfortunately, Chrome only allows 16 canvases to be alive at any given time. If a new canvas is created after this, then the oldest canvas is destroyed. This revelation came only at the late stages of the development, and at first, it seemed that this limitation would greatly diminish the quality of the software, as this would mean that saves, the leader board or the ancestry view could not be displayed visually.

This issue was solved by only using one canvas to draw all saves and top-scoring individuals, while limiting the GA displays to only 15 canvases. Whenever a new static bridge display is required, the single canvas is moved into position where the bridge is drawn and finally it is replaced with an image with the same content. This is a rather inelegant way of solving this problem, as it breaks the contract of the `tz-canvas` directive (as it is supposed to create a canvas), but this process is invisible to the user. On the other hand, this workaround has a positive side effect, which is the fact that this way the image of the bridge can be saved as a file and can be shared, which would not be possible if it was a canvas.

5.6.2 GA

After the first stage of the simulation was ready, it was time for creating the GA, which was not without its difficulties. The first big problem arose because of the poor data structure choice. Until this structure was only used to represent the internal state of the bridge, it was fine. However, when it came to implementing mutation operators it was quite hard to change the data so that it stays valid. One of the key reasons for this is that indices were used as opposed to references when describing connections between nodes. This resulted in many hours of finding and fixing indexing bugs.

The second problem with the GA was the fitness function. The main issue was that it was hard to determine what variables would predict a better bridge. After some of these precursors were found, the next difficulty was to create a formula that would map these variables to a single value that represents a bridge's strength. Since a GA's only guidance is the fitness function, determining this formula is crucial. The current fitness function is the result of several days of experiments with different formulae, but I am still not convinced that this is the best possible fitness function.

Likewise, the implementation of the selection function was similarly difficult, taking some days to experiment with different selection methods. On the bright side however, the trouble with these methods gave the idea to allow users to edit these functions themselves, which is a novel feature that gives an unprecedented amount of freedom to users to experiment with the system.

Lastly, even after optimising these two functions as much as possible the algorithm still does not produce results that were expected at the start of the project. Initially, I thought that if the GA was run for a sufficiently long time, with a relatively large population it would be able to produce bridges that are close to what a human would design. Unfortunately, this is not the case, as even high scoring bridges that are developed through hundreds of generations have many useless edges that do not add anything to the overall quality to the rest of the structure. When I was faced with this problem at the first time, I tried to solve it by adjusting the above mentioned functions, but without success. This process took a large amount of time, a large portion of which was spent with watching bridges evolve, which could have been spent more efficiently on writing this report.

6 EVALUATION OF THE PROJECT

To evaluate the finished product, the functional and non-functional requirements ([Chapter 3](#)) were used to determine if the goals of the project were achieved. Furthermore, some volunteers were asked to use the software for a short period of time and then comment on its usability, gameplay and other aspects.

6.1 Comparison with Requirements

6.1.1 Functional Requirements

The aim of this comparison is to make sure that the functions that were planned exist in the system without being concerned about the implementation of the functions. The full comparison listing can be found in [Appendix B](#).

6.1.2 Non-Functional Requirements

- Performance requirements

At every stage of the implementation, the software was tested on an ‘average laptop’ to validate this requirement. Whenever the runtime performance decreased, measures were taken to find and resolve the bottleneck that caused the issue. In the end, it can be said that the app adheres to this constraint.

- Security Requirements

Even though it was determined that HTTP would be sufficient as the transport layer for the software, in the end HTTPS is used in the production version, as HTTPS is enabled by default on Heroku, the service provider of the program.

- Software Quality Attributes

Even though the software was thoroughly tested, there are probably some bugs still left in the software. However, there are many failure handling mechanics built-in the system, so that unrecoverable errors are largely impossible. The software is modularised using Angular’s Dependency Injection feature, which makes it easy to add and modify features. Comments are present across the whole code base, making the code easier to read and maintain.

- Accessibility & Usability Requirements

The final design follows the Material Design specification, which makes it easy to understand, as users have most likely encountered the UI elements before. Help sections are placed across the UI.

To summarise it can be said that all functional and non-functional requirements were achieved, and therefore the final product can be considered a success.

6.2 Evaluating the GA

To evaluate the GA's performance the algorithm was ran for 250 generations, with a population of 50 (over 12500 bridges evaluated). The best bridge that search produced can be seen in Figure 23.

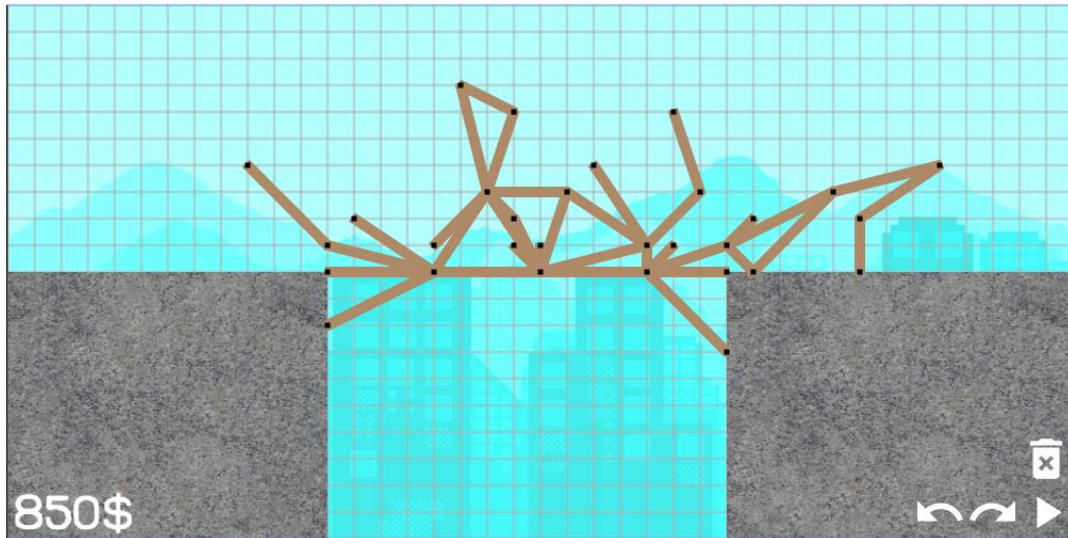


Figure 23 Result of a long running GA (recreated in the editor)

Even though this bridge is stable (~30% worst load), it does not look like one that was designed by a user, as it has many edges that do not contribute to the structure's integrity. This behaviour is not unusual, as this happens consistently on all runs. There are many reasons that can explain why these unnecessary edges are not removed.

- Evaluating 250 generations can be considered a low amount as often GAs are ran for thousands of generations. However, this is not feasible, as evaluating 250 generations took several hours.
- The size of the population is quite limiting, as GAs in the literature often operate with a population consisting of over a thousand individuals. Because of this, the population's diversity decreases too soon, even though tournament selection was used, which delays this effect the most.
- In GAs, mutation operators are often used to retain the population's diversity, as they often change the whole genetic code of an individual. In this case, mutation operators are used to explore the search space, while canonically this is the purpose of the crossover operator.
- Despite my best efforts to create a reliable fitness function, it is not perfect, as the values produced by it do not always reflect the bridge's strength. Furthermore, a larger cost is not as heavily penalised, therefore unnecessary edges do not affect the fitness value that much.
- The physics simulator is not completely deterministic as a stable bridge might collapse once out of a hundred simulations. Since simulations are only ran once for each candidate, and the fitness function uses the data produced by it, it can underrate an otherwise stable bridge.
- Lastly, since the simulation data is queried asynchronously, it might be the case that the values are not extracted at the same point across different simulation. This might lead to inaccuracies in the fitness value, which might in turn decrease the efficiency of the algorithm.

To summarise it can be said that the GA does achieve stable bridges, however, since some unconventional choices were made about its implementation, the bridges that it produces are underwhelming.

6.3 User evaluation

To evaluate the app from the users' point of view some volunteers were asked to play with the app for a short time, while I was watching them. Then they were asked about how they feel about the gameplay, how helpful the help sections were, etc. The reason why I chose this method for analysing user engagement is that at its core, this software is a toy. Toys in general appeal for emotions and therefore it is hard to

evaluate them objectively on some arbitrary 1 to 10 scales. Since emotions can be expressed more freely in a laid-back dialogue than on a dull questionnaire, this alternative methodology was chosen.

Even though most participants liked the gameplay aspect, they still found some issues with it. The most mentioned problem is that bridges are sometimes quite tedious to make. They even suggested solutions, such as drawing multiple edges in one continuous mouse drag. Another solution might be to mirror every drawn edge. This could be turned on or off via a switch that could be introduced to the editor.

Another missing feature that was noted by participants is the lack of sounds. This was quite an interesting discovery, as this was not even considered as an idea, even though this is a crucial feature in most modern games. Clearly adding sound effects to the game would make it more immersive and enjoyable; however, there was not enough time to implement this great suggestion.

As for the GA, the users found this feature of the app quite interesting. It was widely agreed that the help section is concise and helpful for understanding the underlying mechanics and parameters. When observing their behaviour most of them would spend a large amount of the time just looking at the evaluation process, but others were much more interested in the ancestry of chromosomes. However, some of them disliked the fact that when their own solutions were passed in as initial population, the solutions would be degraded, and not improved.

After they finished trying out the software, most users told me that they liked it, and that they would probably play with it again in the future. Moreover, volunteers who had already encountered GAs before found the use of the algorithm quite interesting and they really appreciated the code-editing feature of the app, as they could run their own experiments with the algorithm.

To conclude it can be said that the careful design of the game has led to a success, as most users seemed to enjoy playing with it, while also learning a few things about GAs. Furthermore, the participants have reported that visualisation of the ancestors of individuals is quite intuitive, which aided their understanding of the underlying process to a great extent, accomplishing the goal of this project.

6.4 Personal Evaluation

In the end, I am quite proud of the software that was produced, for several reasons. The user evaluation has shown that the gameplay aspect is intuitive and enjoyable. The GA solver is interesting to watch, and it is capable of producing stable bridges. Furthermore, the visualisation of ancestors is a great success, as users have enjoyed comparing different generations of bridges, which helped them understand how a GA works.

If I had the chance to start again, I would do many things differently. These include but not limited to using a different game engine, a faster build system, adapting Test Driven Development from day one, and choosing a better data-structure for the GA that works with a crossover operator. Finally, the most important thing that I would like to change is starting the implementation earlier. Because this was the first project of this size that I accomplished, it was challenging for me to get started. The challenge that I faced was that every design and implementation decision would affect the rest of the project and I was afraid to make these decisions since making mistakes was not an option.

As it turned out later, one of the very first implementation decisions, the choice of the game engine, was the worst mistake throughout the project, as it caused several delays and therefore, the planned deadlines for milestones in the project were broken one after the other. Towards the end of the project, these delays added up and the project had to be finished in a rush, which was very stressful for me.

On the bright side, I have learned many things about web development and GAs throughout the project that I will be able to use later in my career. Despite the difficulties, I enjoyed working on this app as it proved to be an interesting challenge, and the final results are quite positive.

7 FURTHER WORK

There are plenty of things that could be changed to make the app better. Some features are quite easy to implement since the modular structure of the software allows it to be extended freely. First of all, the gameplay aspect could be improved by adding some tools to speed up the bridge designing process. One such feature could be the automatic mirroring of edges or the ability to draw multiple edges in one drag. Secondly, the game could be made more immersive, by adding sound effects, and possibly some background music. Another feature could be to show which edges are most stressed when simulating by using a colour scale. This could be especially interesting in the GA mode, as this would show how generations improve on the load of individual edges. A further improvement to the GA could be to allow the re-simulation of the bridges, or some way to store developed bridges as saves.

There are other improvements that would require considerably more work. For example, there could be a way to visualise the evolution of whole generations, and not just a single bridge. A simpler improvement could be to compare the evolution of two individuals side by side. In addition, some other visualisations can be considered, for example, a convergence graph of overall fitness, average fitness, price or max load, or a combination of these. A graph could be made that shows how the individuals are selected into the next generations, however, this would be quite hard to generalise in a way that the selection function can be still edited by the user.

Some improvements could only be made by redesigning a large part of the software. If this would ever happen a new type of GA could be implemented with a different problem representation that could avoid adding many unnecessary edges to the structure. Furthermore, the GA settings screen could be redesigned to add even more customisation options, while separating settings in some way (e.g. tabbed layout), so that there is less information on one screen, and therefore better complying with the *Recognition rather than recall* heuristic.

Lastly, despite the testing efforts, there are probably still some undiscovered bugs, which should be fixed. Even though, the above-mentioned features do not exist in the software, it might be the case that if they were added, the UI would become cluttered, and thus, it would reduce the usability of the software. Therefore, I think that if any further work will be done it will be concerned with polishing existing features, without adding too many elements to it, in order to keep the app as simple as possible.

8 SUMMARY

The aim of this project was to develop a web-based visualisation tool for GAs. To accomplish this task, a highly customisable bridge building game was made, which is very similar to a truss optimisation problem that served as the problem domain for a GA. In the design and implementation of the app, some key software engineering practices were used to mitigate its complexity. It was shown how modern web technologies such as Angular and ES6 were used to make the software modular. Furthermore, it was shown how some adjustments were made to the canonical GA to better suit the problem, however, these adjustments led to some unexpected consequences.

In the end, the main goals of the project were achieved and it can be considered complete, as the final software boasts with an enjoyable game that can be solved with a GA, while providing an intuitive visualisation tool that can show how the solutions were evolving over time. Moreover, the GA has many customisation options as well, which makes it possible to experiment with different configurations of the same algorithm to see how each of the parameters affect the progression of the algorithm.

9 REFERENCES

- [1] John H Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*: U Michigan Press, 1975.
- [2] Wen Ping Chin, "A visual genetic algorithm tool," University of Malaysia, 2006.
- [3] William B Shine and Christoph F Eick, "Visualizing the evolution of genetic algorithm search processes," in *Evolutionary Computation, 1997., IEEE International Conference on*, 1997, pp. 367-372.
- [4] Hartmut Pohlheim, "Visualization of evolutionary algorithms-set of standard techniques and multidimensional visualization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, pp. 533-540.
- [5] Adam Prügel-Bennett and Jonathan L Shapiro, "The dynamics of a genetic algorithm for simple random Ising systems," *Physica D: Nonlinear Phenomena*, vol. 104, pp. 75-114, 1997.
- [6] Tanja Dabs and Jochen Schoof, *A graphical user interface for genetic algorithms*. Universitat Würzburg, Lehrstuhl für Informatik: Report 98, 1995.
- [7] Emma Hart and Peter Ross, "Enhancing the Performance of a GA through Visualization," in *GECCO*, 2000, pp. 347-354.
- [8] Emma Hart and Peter Ross, "GAVEL-a new tool for genetic algorithm visualization," *Evolutionary Computation, IEEE Transactions on*, vol. 5, pp. 335-348, 2001.
- [9] Xavier Llorà, Kumara Sastry, Francesc Alías, David E Goldberg, and Michael Welge, "Analyzing active interactive genetic algorithms using visual analytics," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 1417-1418.
- [10] M Zetakova Mach and M Zetakova, "Z.: Visualising genetic algorithms: A way through the Labyrinth of search space," *Intelligent Technologies-Theory and Applications. IOS Press, Amsterdam*, pp. 279-285, 2002.
- [11] Carlos B Lucasius and Gerrit Kateman, "Application of genetic algorithms in chemometrics," in *Proceedings of the third international conference on Genetic algorithms*, 1989, pp. 170-176.
- [12] James Edward Baker, "Adaptive selection methods for genetic algorithms," in *Proceedings of an International Conference on Genetic Algorithms and their applications*, 1985, pp. 101-111.
- [13] John J Grefenstette and James E Baker, "How genetic algorithms work: A critical look at implicit parallelism," in *Proceedings of the third international conference on Genetic algorithms*, 1989, pp. 20-27.
- [14] David E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Reading, Mass.: Addison-Wesley Pub. Co., 1989.
- [15] Gilbert Syswerda, "Uniform crossover in genetic algorithms," *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 2-9, 1989.
- [16] DE Golberg and R Lingle, "Alleles, Loci, and the TSP," in *Proceedings of the 1st International Conference on Genetic Algorithms* 1985.
- [17] GS Lauer, NR Sandell, DP Bertsekas, and TA Posbergh, "Solution of large-scale optimal unit commitment problems," *Power Apparatus and Systems, IEEE Transactions on*, pp. 79-86, 1982.
- [18] F Zhuang and FD Galiana, "Unit commitment by simulated annealing," *Power Systems, IEEE Transactions on*, vol. 5, pp. 311-318, 1990.
- [19] Z Ouyang and SM Shahidehpour, "A multi-stage intelligent system for unit commitment," *Power Systems, IEEE Transactions on*, vol. 7, pp. 639-646, 1992.
- [20] Spyros A Kazarlis, AG Bakirtzis, and Vassilios Petridis, "A genetic algorithm solution to the unit commitment problem," *Power Systems, IEEE Transactions on*, vol. 11, pp. 83-92, 1996.
- [21] Vassilios Petridis and S Kazarlis, "Varying quality function in genetic algorithms and the cutting problem," in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, 1994, pp. 166-169.
- [22] Andrea Lodi, Silvano Martello, and Daniele Vigo, "Approximation algorithms for the oriented two-dimensional bin packing problem," *European Journal of Operational Research*, vol. 112, pp. 158-166, 1999.
- [23] Andrea Lodi, Silvano Martello, and Daniele Vigo, "Heuristic algorithms for the three-dimensional bin packing problem," *European Journal of Operational Research*, vol. 141, pp. 410-420, 2002.
- [24] Oluf Faroe, David Pisinger, and Martin Zachariasen, "Guided Local Search for the Three-Dimensional Bin-Packing Problem," *INFORMS Journal on Computing*, vol. 15, pp. 267-283, 2003.

- [25] Michele Monaci and Paolo Toth, "A Set-Covering-Based Heuristic Approach for Bin-Packing Problems," *INFORMS Journal on Computing*, vol. 18, pp. 71-85, 2006.
- [26] José Fernando Gonçalves and Mauricio GC Resende, "A biased random key genetic algorithm for 2D and 3D bin packing problems," *International Journal of Production Economics*, vol. 145, pp. 500-510, 2013.
- [27] James C. Bean, "Genetic Algorithms and Random Keys for Sequencing and Optimization," *ORSA Journal on Computing*, vol. 6, pp. 154-160, 1994.
- [28] Rafael Matsunaga. (2014, Accessed: March 2016). *HTML5 Genetic Cars*. Available: http://rednuht.org/genetic_cars_2
- [29] Rafael Matsunaga. (2014, Accessed: March 2016). *HTML5 Genetic Cars - GitHub source*. Available: http://github.com/red42/HTML5_Genetic_Cars
- [30] S Rajeev and CS Krishnamoorthy, "Discrete optimization of structures using genetic algorithms," *Journal of structural engineering*, vol. 118, pp. 1233-1250, 1992.
- [31] GIN Rozvany, "Exact analytical solutions for some popular benchmark problems in topology optimization," *Structural optimization*, vol. 15, pp. 42-48, 1998.
- [32] M. Fenton, C. McNally, J. Byrne, E. Hemberg, J. McDermott, and M. O' Neill, "Discrete Planar Truss Optimization by Node Position Variation using Grammatical Evolution," *IEEE Transactions on Evolutionary Computation*, pp. 1-1, 2015.
- [33] Conor Ryan, JJ Collins, and Michael O Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Genetic Programming*, ed: Springer, 1998, pp. 83-96.
- [34] Peter Naur, John W Backus, Friedrich L Bauer, Julien Green, Charles Katz, John McCarthy, *et al.*, "Revised report on the algorithmic language Algol 60," *Communications of the ACM*, vol. 6, pp. 1-17, 1963.
- [35] Boris Delaunay, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, pp. 1-2, 1934.
- [36] Google. (2014, Accessed: April 2016). *Material Design specification*. Available: <http://www.google.com/design/spec/material-design/introduction.html>
- [37] Twitter. (2011, Accessed: April 2016). *Bootstrap*. Available: <http://getbootstrap.com/>
- [38] Zurb. (2011, Accessed: April 2016). *Foundation*. Available: <http://foundation.zurb.com/>
- [39] Statista. (2016, Accessed: April 2016). *Number of apps available in leading app stores as of July 2015* Available: <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>
- [40] Statista. (2016, Accessed: April 2016). *Market share of the leading search engines in Germany from 2014 to 2016* Available: <http://www.statista.com/statistics/445002/market-shares-leading-search-engines-germany/>
- [41] Statista. (2016, Accessed: April 2016). *Most popular internet browsers in Canada in 1st quarter 2016, by market share* Available: <http://www.statista.com/statistics/186150/most-popular-browsers-in-canada-by-market-share/>
- [42] Statista. (2016, Accessed: April 2016). *Worldwide desktop video viewer market share as of September 2014*. Available: <http://www.statista.com/statistics/187001/online-video-viewer-worldwide-distribution/>
- [43] Jakob Nielsen and Rolf Molich, "Heuristic evaluation of user interfaces," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1990, pp. 249-256.
- [44] Node.js Foundation. (2016, Accessed: April 2016). *Node.js*. Available: <http://nodejs.org/en/>
- [45] Node.js Foundation. (2016, Accessed: April 2016). *Express: Fast, unopinionated, minimalist web framework for Node.js*. Available: <http://expressjs.com/>
- [46] Facebook. (2013, Accessed: April 2016). *React*. Available: <http://facebook.github.io/react/>
- [47] Jeremy Ashkenas. (2010, Accessed: April 2016). *Backbone*. Available: <http://backbonejs.org/>
- [48] Tilde Inc. (2011, Accessed: April 2016). *Ember*. Available: <http://emberjs.com/>
- [49] StackOverflow. (2016, Accessed: 11 April 2016). *Questions tagged with Angular*. Available: <http://stackoverflow.com/questions/tagged/angularjs>
- [50] StackOverflow. (2016, Accessed: 11 April 2016). *Questions tagged with Backbone*. Available: <http://stackoverflow.com/questions/tagged/backbone.js>
- [51] Ecma International. (2015, Accessed: April 2016). *ECMAScript® 2015 Language Specification*. Available: <http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf>

- [52] Paul Krill. (2015, Accessed: April 2016). *The new version of the specification underpinning JavaScript brings improvements to syntax and structural issues*. Available: <http://www.infoworld.com/article/2937716/javascript/its-official-ecmascript-6-is-approved.html>
- [53] Juriy "kangax" Zaytsev. (2016, Accessed: April 2016). *ES6 compatibility table*. Available: <http://kangax.github.io/compat-table/es6/>
- [54] Babel. (2016, Accessed: April 2016). *Babel: The compiler for writing next generation JavaScript*. Available: <http://babeljs.io/>
- [55] Gulp. (2016, Accessed: April 2016). *Gulp: the streaming build system*. Available: <https://github.com/gulpjs/gulp>
- [56] Andrew Koroluk. (2016, Accessed: April 2016). *AngularJS Full-Stack generator*. Available: <http://github.com/angular-fullstack/generator-angular-fullstack>
- [57] M. D. McIlroy, E. N. Pinson, and B. A. Tague, "UNIX Time-Sharing System: Foreword," *Bell System Technical Journal*, vol. 57, pp. 1899-1904, 1978.
- [58] Phaser. (2016, Accessed: April 2016). *Phaser*. Available: <http://phaser.io/>
- [59] melonJS. (2016, Accessed: April 2016). *melonJS*. Available: <http://melonjs.org/>
- [60] Turbulenz. (2016, Accessed: April 2016). *Turbulenz*. Available: <http://biz.turbulenz.com/>

10 APPENDIX

A. Original System Requirements Specification

Functional Requirements

1. Physics Simulation
 - 1.1 The World
 - 1.1.1 ...will be based on a grid system.
 - 1.1.2 ...will have two sides: start and finish.
 - 1.1.2.1 ...that will be placed on the left and right side of the rendering, respectively.
 - 1.1.2.2 The distance between the sides will be adjustable.
 - 1.1.2.3 The elevation between the sides will be adjustable.
 - 1.1.3 The gravitation will be adjustable.
 - 1.2 The Bridge
 - 1.2.1 ...will consist of straight lines, called blocks.
 - 1.2.1.1 ...which will be one of three easily identifiable materials (names and properties TBD)
 - 1.2.2 The starting and end point of blocks can only be grid points.
 - 1.2.3 If the block's start or end point touches one of the sides or another block, then it is constrained in that position.
 - 1.2.4 Blocks can rotate around constraints with some freedom.
 - 1.2.5 Each block has a maximum length, determined by the material type.
 - 1.2.6 The added cost of the blocks will make the cost of the bridge.
 - 1.3 The Editor
 - 1.3.1 In the editor, the user will be able to place blocks in the grid system.
 - 1.3.1.1 ...in a drag and drop fashion.
 - 1.3.1.2 The material for the block will be easily selectable.
 - 1.3.1.3 When a material is hovered, some details will be displayed.
 - 1.3.2 ...will have undo and redo functionality.
 - 1.3.3 ...will show the cost of the bridge.
 - 1.3.4 Built bridges will be able to be saved with a name and reloaded into the editor.
 - 1.3.4.1 ...that shall be stored in the browser's local storage.
 - 1.4 The Simulation
 - 1.4.1 ...can be paused or resumed.
 - 1.4.2 ...shows the current load of the bridge.
 - 1.5 The Vehicle
 - 1.5.1 ...will be placed on the start side, when the simulation is started.
 - 1.5.2 ...will try to move towards the finish side.
 - 1.5.3 The bridge will be the only means for the vehicle to cross the two sides.
 - 1.5.4 The vehicle's weight will be editable.
2. The GA
 - 2.6 Scoring
 - 2.6.1 ...will be based on whether the bridge collapses,
 - 2.6.1.1 If it does then it will be based on the time it takes to do so.
 - 2.6.2 ...otherwise
 - 2.6.2.1 The maximum load will be used to calculate the value.
 - 2.6.2.2 Among other parameters.
 - 2.7 Settings
 - 2.7.1 The population size will be editable.
 - 2.7.2 The mutation rate will be editable.
 - 2.7.2.1 The types of mutations will be selectable.
 - 2.7.3 The crossover rate will be editable.
 - 2.7.3.1 The crossover types will be selectable.
 - 2.7.4 The parameters of the world will be locked in before starting the GA.

- 2.7.5 The selection rate will be editable (how many of the last generation will be carried over).
- 2.8 Constraints can be set.
 - 2.8.1 ...on the bridge cost.
 - 2.8.2 ...used materials.
 - 2.8.3 ...maximum bridge load.
 - 2.8.4 ...maximum number of blocks.
 - 2.8.5 Some of the starting population can be selected from the previously saved bridges (made by the user).
- 2.9 Rendering
 - 2.9.1 Can be turned off.
 - 2.9.2 Or multiple simulations can be rendered at the same time.
 - 2.9.2.1 In a tiled layout.
 - 2.9.3 Some of the simulation might run on the server (TBD).
 - 2.9.4 Can be sped up.
- 2.10 User interaction
 - 2.10.1 When the simulation is running, the currently simulated bridge can be “sculpted”.
 - 2.10.1.1 With the mouse nodes can be moved.
 - 2.10.1.2 With the mouse, constraints can be broken.
 - 2.10.2 At the end of each generation, the user can choose between the best three scoring bridges.
 - 2.10.2.1 The chosen bridge’s score will be boosted, thus making it more likely that its genes will be carried over to the next generation.
- 2.11 Leader board
 - 2.11.1 The highest scoring bridges shall form a leader board.
 - 2.11.2 Bridges will have a name that is generated using their genome.
- 2.12 Statistics
 - 2.12.1 A family tree of bridges will be available.
 - 2.12.2 Other statistics of the running search will be available (TBD).

Non-Functional Requirements

Performance Requirements

The app should run reasonably well on an average laptop.

Security Requirements

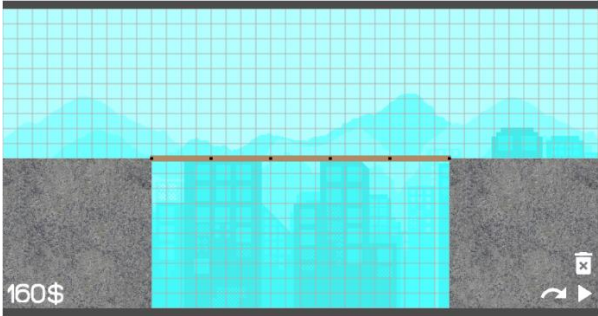
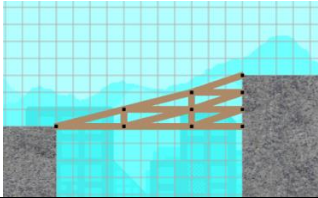
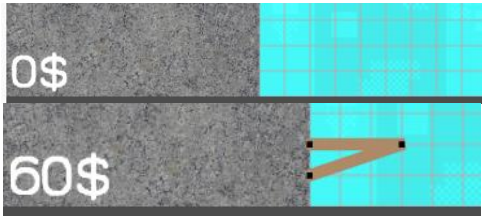
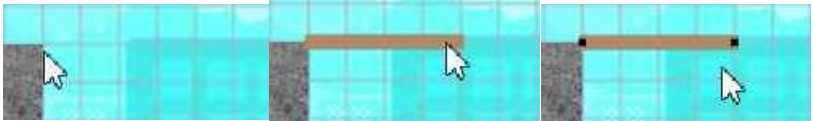
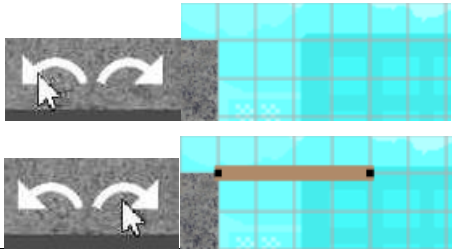
Since there is no sensitive data transmitted, there is no need to use HTTPS; HTTP will be sufficient.

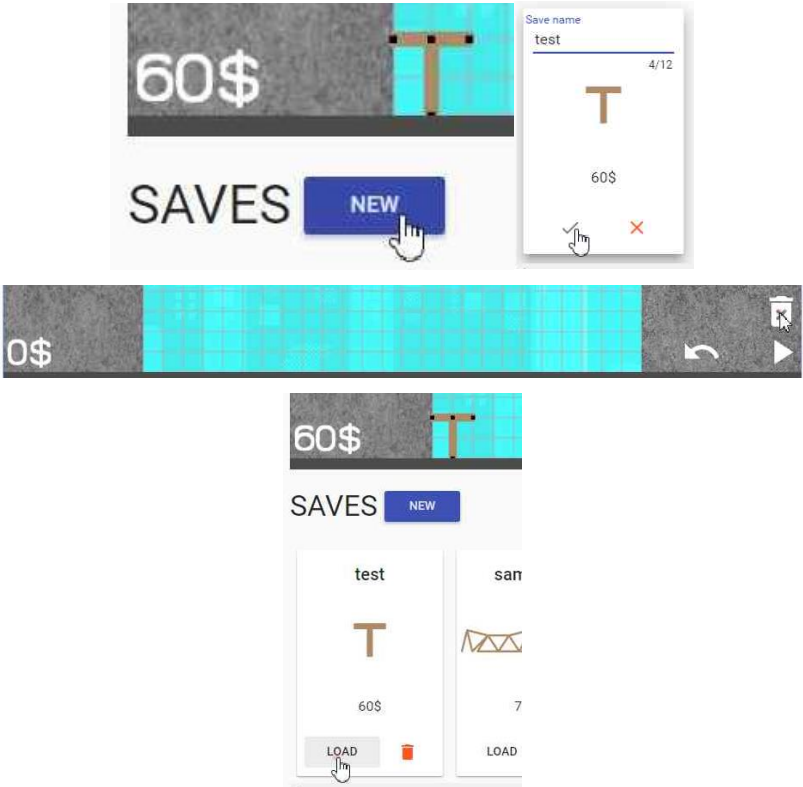
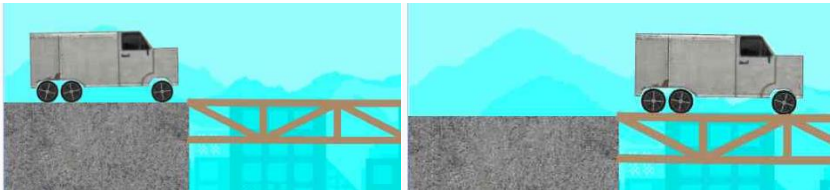
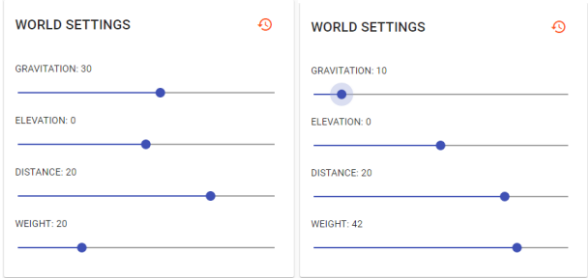
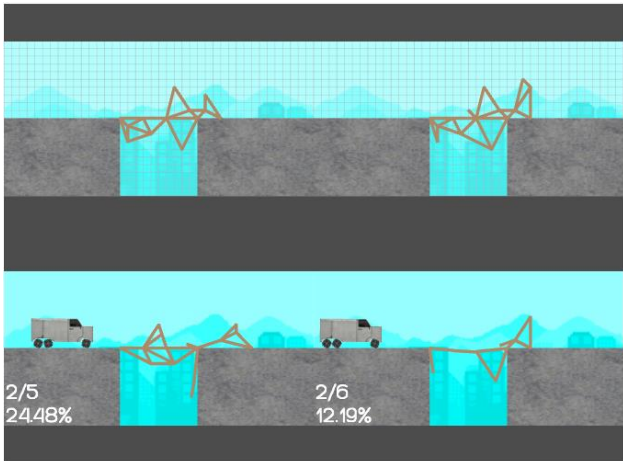
Software Quality Attributes

To avoid bugs, unit tests shall be written to cover most (at least 90%) of the code base.

B. Comparison with Functional Requirements

Passed tests

Requirement	Evidence
<p>Grid based physics simulation with two platforms on two sides.</p> <p>Bridges can be built from edges with limited length that connect on their ends.</p>	
<p>Distance and elevation are adjustable.</p>	
<p>The cumulative cost of nodes and edges will be the cost of the bridge.</p>	
<p>Design bridges by clicking and dragging.</p> <p>Nodes are generated automatically at the ends.</p>	
<p>Has undo/redo functionality (following the previous test).</p>	

<p>Store and load bridges.</p>	
<p>The van is placed on the left side and it moves towards the right side.</p>	
<p>Weight and gravitation is adjustable.</p>	
<p>Bridges are evaluated by a physics simulation.</p>	

World parameters and GA parameters (mutation rate, population initial population) can be edited before running the GA.

WORLD SETTINGS

GRAVITATION: 30

ELEVATION: 0

DISTANCE: 10

WEIGHT: 20

GA SETTINGS





MUTATION RATE: 7

PARALLEL SIMULATIONS: 9

POPULATION: 15

WAIT BEFORE SIMULATION: 1s

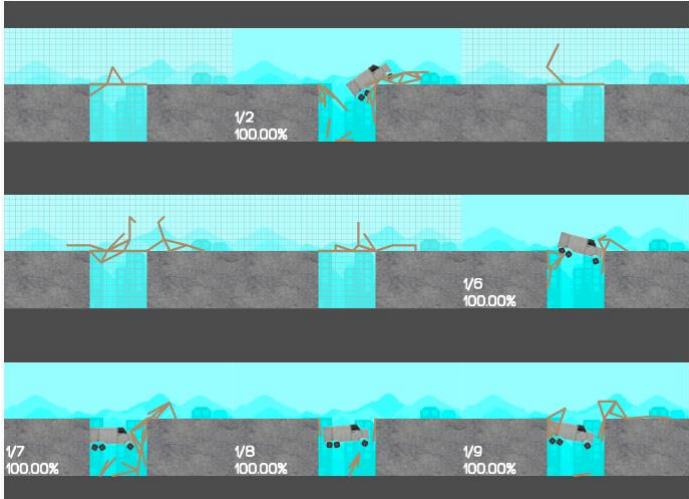
STARTING POPULATION

test	sample 3	sample 2	sample 1
			
60\$	700\$	470\$	494\$
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Simulations run in an editable size grid.

PARALLEL SIMULATIONS: 9

width 3 height * 3



Rendering can be turned off.



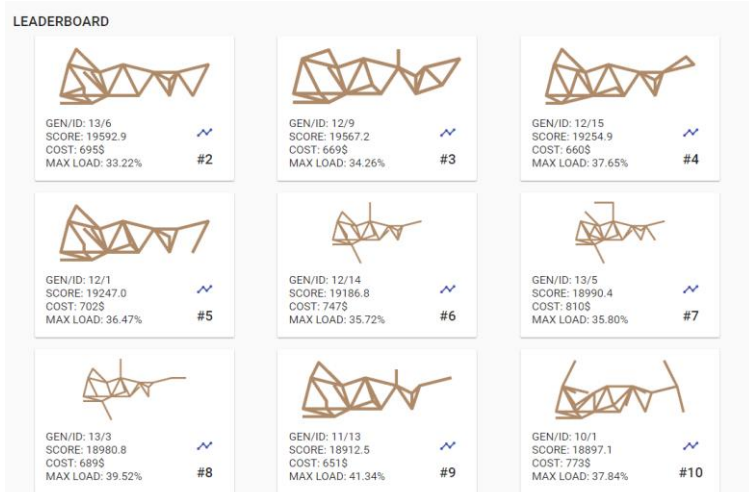
SIMULATION SETTINGS

SIMULATION SPEED: 1

RENDERING: ☐

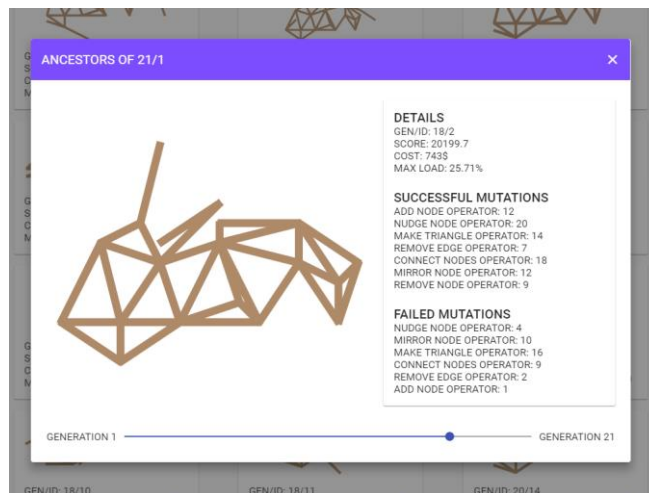
GENERATION: 5

A leader board is available, showing the top-scoring bridges.



The ancestors of leader board items can be shown.

Ancestry is a line instead of a tree.



Edit fitness and selection functions, as well as mutation operator weights; all hidden away behind an advanced settings toggle.



Failed tests

None.