Procedural Generation of Levels with Controllable Difficulty for a Platform Game Using a Genetic Algorithm

Procedurell generering av banor med kontrollerbar svårighetsgrad till ett platformspel med hjälp av en genetisk algoritm

Viktor Andersson
Johan Classon

Supervisor: Aseel Berglund
Examiner: Henrik Eriksson
Upphovsrätt


Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page http://www.ep.liu.se/.

© Viktor Andersson
© Johan Classon
Abstract

This thesis describes the implementation and evaluation of a genetic algorithm (GA) for procedurally generating levels with controllable difficulty for a motion-based 2D platform game. Manually creating content can be time-consuming, and it may be desirable to automate this process with an algorithm, using Procedural Content Generation (PCG). An algorithm was implemented and then refined with an iterative method by conducting user tests. The resulting algorithm is considered a success and shows that using GA’s for this kind of PCG is viable. An algorithm able to control difficulty of its output was achieved, but more refinement could be made with further user tests. Using a GA for this purpose, one should find elements that affect difficulty, incorporate these in the fitness function, and test generated content to ensure that the fitness function correctly evaluates solutions with regard to the desired output.
We would like to thank our supervisors Aseel and Erik who helped test the product during the development and gave valuable feedback throughout the thesis. We also extend our gratitude to the people who took time out of their day to help us test the product, since it provided us with invaluable feedback.

I, Viktor Andersson, would like to thank my mother Maj-Karin and my brother Joel, for supporting me throughout my education and for shaping me into the person I am today. Thanks to all my friends, both old and new for making my life wonderful.

I would like to thank Carl Einarson for being able to hang out at a moments notice when I needed it, and Martin Wiman for the many years of friendship and the time spent together in Linköping. I would also like to thank Amanda Edlund and Linnea Sandgren for the times I spent with you in my hometown between semesters. You both made it feel like I never left and you made me look forward to each visit home. Finally I would like to thank my corridor friend Tove Canerstam for the many conversations we had that put me in a good mood, and for making our corridor feel like not just a place to eat and sleep, but like a collective. You guys are awesome.

I, Johan Classon, would like to thank my mother Carina, my father Lars-Erik, and my brothers Andreas, Tobias and Daniel, for always supporting me and believing in me throughout my whole education.

To my girlfriend Joana for giving me energy to continue, pushing me to study hard, always believing in me and for showing me the wonderful things that life has to offer. To Per Thor, for always being my true friend through all the phases of my life. To all my friends, old and new, making these five years of university studies some of the best of my life. You have all helped shape me into the person I am today, in your own way, and for that I thank you from the bottom of my heart. Special thanks also to Lukas Wingren and Anders Skogsdal for inspiring me to start my university studies.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iv</td>
</tr>
<tr>
<td>Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Aim</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Research questions</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Delimitations</td>
<td>3</td>
</tr>
<tr>
<td>1.5 Background</td>
<td>4</td>
</tr>
<tr>
<td><strong>2 Theory</strong></td>
<td>7</td>
</tr>
<tr>
<td>2.1 PCG</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Genetic Algorithms</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Validation of levels</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Motion based games</td>
<td>13</td>
</tr>
<tr>
<td>2.5 PENS</td>
<td>14</td>
</tr>
<tr>
<td>2.6 Iterative development</td>
<td>14</td>
</tr>
<tr>
<td><strong>3 Method</strong></td>
<td>16</td>
</tr>
<tr>
<td>3.1 Similar work</td>
<td>17</td>
</tr>
<tr>
<td>3.2 Representation of levels</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Implementation of the GA</td>
<td>17</td>
</tr>
<tr>
<td>3.4 Feasibility algorithm</td>
<td>18</td>
</tr>
<tr>
<td>3.5 Identification of relevant variables</td>
<td>18</td>
</tr>
<tr>
<td>3.6 Iterative step</td>
<td>19</td>
</tr>
<tr>
<td>3.7 User tests</td>
<td>20</td>
</tr>
<tr>
<td><strong>4 Results</strong></td>
<td>22</td>
</tr>
<tr>
<td>4.1 Pre-study</td>
<td>22</td>
</tr>
<tr>
<td>4.2 Iteration 1</td>
<td>23</td>
</tr>
<tr>
<td>4.3 Iteration 2</td>
<td>27</td>
</tr>
<tr>
<td>4.4 End phase</td>
<td>31</td>
</tr>
<tr>
<td><strong>5 Discussion</strong></td>
<td>34</td>
</tr>
<tr>
<td>5.1 Pre-study</td>
<td>34</td>
</tr>
<tr>
<td>5.2 Iteration 1</td>
<td>35</td>
</tr>
</tbody>
</table>
5.3 Iteration 2 ................................................................. 38
5.4 End phase ............................................................... 41
5.5 Method ................................................................. 43
5.6 Genetic algorithm .................................................. 44
5.7 Research Question .................................................. 45
5.8 The work in a wider context ..................................... 46

6 Conclusions ............................................................. 48
  6.1 Future work .......................................................... 49

Bibliography .............................................................. 50

A Appendix - Matlab code .......................................... 53

B Appendix - PENS Questionnaire ................................. 55
List of Figures

1.1 Infinite Mario, created by Markus Persson. The game uses a seed to generate new level every time the game is played. .......................................................... 3
1.2 Screen capture of the game with two motion zones activated. The white contour shows the player movement. .......................................................... 5
1.3 Structure as defined by Tim Ziegenbein .......................................................... 5

3.1 Overview of the iterative process ........................................................................ 19

4.1 Easy chunk, Fitness: 1.1633 ........................................................................ 24
4.2 Medium chunk, Fitness: 1.89321 .................................................................... 24
4.3 Hard chunk, Fitness: 2.46408 ........................................................................ 24
4.4 Fitness of level vs. average perceived difficulty of level (starting config) .......... 25
4.5 Fitness of level vs. average perceived difficulty of level with adjusted variables (iteration 1) .................................................................................. 25
4.6 Deviation from average number of deaths vs. average perceived difficulty of level (iteration 1) .............................................................................. 25
4.7 Deviation from average time vs. average perceived difficulty of level (iteration 1) ............................................................................................... 26
4.8 Fitness of level vs. average perceived difficulty of level with new variable and adjusted variables .............................................................................. 27
4.9 Easy chunk (iteration 2), Fitness: 2.79504 ......................................................... 28
4.10 Medium chunk (iteration 2), Fitness: 4.53246 .................................................. 29
4.11 Hard chunk (iteration 2), Fitness: 6.73556 ......................................................... 29
4.12 Fitness vs perceived difficulty. Difference: 0.1735 ............................................ 30
4.13 Fitness vs perceived difficulty, iteration 2. Difference: 7.9223e-05 .................... 30
4.14 Deviation from average number of deaths vs. difference in perceived difficulty .......................................................... 31
4.15 Deviation from average time to complete level vs. difference in perceived difficulty ........................................................................................................ 31
4.16 Percentage of coins picked up in a level vs difference in perceived difficulty ........ 32
4.17 Fitness of level vs average perceived difficulty of level (end phase) .................. 32

A.1 Matlab code, iteration 1. .................................................................................. 54
List of Tables

2.1 Correlation between features and challenge ........................................ 12
4.1 Fitness values and perceived difficulty in first user test ...................... 25
4.2 Results of PENS in iteration 1 .............................................................. 26
4.3 Variables in fitness function ................................................................. 27
4.4 Features and difficulty thresholds ....................................................... 28
4.5 Fitness values and perceived difficulty in second user test .................. 28
4.6 Variables in fitness function, iteration 2 .............................................. 30
4.7 Results of PENS in iteration 2 .............................................................. 31
4.8 Variables in fitness function, end phase ........................................... 32
4.9 Results of PENS in both iterations ....................................................... 32
Developing games is no longer just an activity for large corporations. Using game delivery platforms such as Valves Steam allows for smaller, independent developers to have a bigger chance of being discovered on a growing gaming market. While these games can get coverage and seem interesting, they still need to have a sizable amount of content to keep players interested and motivate a purchase. Creating this content can take up a lot of time, especially if the game is developed by just a few people. Ensuring that this content can be generated automatically would allow for the developers to focus on other parts of the development, while still keeping the amount of content high.

This master’s thesis aims at finding out how algorithms can be used to procedurally generate content for a game, controlled by a motion sensor, that will help make the game a more fun and enjoyable experience. A genetic algorithm, a fitness function, and a way to automatically determine that a level lives up to certain specified criteria, e.g. that it is possible to complete the level, will be created. This is in order to automatically and with minimal user input create levels for a motion-based 2D platform game.

1.1 Motivation

As the gaming industry develops the amount of content required in a game increases. As more content is needed to keep players interested more design work is needed to fulfill these requirements. Creating content in a game, e.g. enemies, levels or items, takes up time and can be expensive. A human usually works quite slow compared to a computer. If such content can be algorithmically generated companies developing games can save a lot of time and money on these tasks. It would allow for even larger and more content rich games to be developed which could be considered a positive effect for players. It would also allow smaller developers to create larger games that would allow them to have a stronger position on the gaming market, at least with regards to the amount of content in games. This automated process is called Procedural Content Generation (PCG).

An example of a platform game utilizing PCG is Infinite Mario, presented in figure 1.1, a java-applet clone of the famous platform game Super Mario. According to the website Indie
Games, the game sets a new seed every time the java applet is loaded, and generates all areas and level selection maps from that seed [27]. This results in a game that, theoretically, never runs out of content.

Figure 1.1: Infinite Mario, created by Markus Persson. The game uses a seed to generate new level every time the game is played.

1.2 Aim

The purpose of the thesis project is to create an algorithm that can procedurally generate feasible and varied levels for a game developed by Active lab, located at Linköping University, to increase the content present in the game, where the difficulty of the created levels can be controlled with input to the algorithm. In this context, feasible means that the created level has a solution, that it can be completed.

The content will be created using a so-called Genetic Algorithm, which is a part of the class Evolutionary Algorithms [9, p. 25]. We wish to use a GA in this thesis because we believe that it will aid in finding solutions that a human might not think of. Depending on the type of GA used, it can be computationally heavy and therefore slow. The algorithm that will be created in this thesis will not be used to create content in run-time, and a slow convergence is therefore not seen as a problem.

The levels that are to be generated will be evaluated during user tests, to ensure that the reality of the levels reflect what the algorithm believes to be true. Because the game is in development, testers will be provided with a questionnaire, see Appendix B, using the Player experience of need satisfaction (PENS) model. The PENS-measure, described in the theory chapter, could help further the development of the game by finding game features that increase the motivation to play the game. This measure will therefore be used in connection to the user tests, to find out if features implemented in between the users tests increased these factors in some way.

1.3 Research questions

The research question that this thesis will answer is:

How can a genetic algorithm be implemented to create levels with controllable difficulty?

1.4 Delimitations

The focus in this project will be on writing an algorithm used to create levels for a motion-controlled 2D platform game, since the game that is used in the project is in this specific
format. This means that it will most likely not be applicable for a game that is in 3D. It will also be special in the regard that the player will be using a motion sensor to play the game, which adds some restrictions to the creation process. It needs to take into account the fact that a motion controller isn’t as precise as using, for example, a keyboard or some other game controller.

The algorithm will only create levels for the game, thus other content that can be procedurally generated will not be included, e.g. enemies, items and graphics. This is because procedural generation of content is a large area and this focus will keep the thesis in a reasonably small scope.

Since the game is in development, elements affecting difficulty may change over time. The algorithm will only take into consideration the elements present at the time of this thesis work. Thus, future elements that may affect difficulty will not be accounted for in the algorithm, and it may require to be changed if new elements are incorporated.

There are many areas of PCG, but the focus in this thesis will be on generation using optimization, that is, the levels are generated and compared to different criteria. Only levels that fulfill the criteria are used. These criteria may change over the course of the thesis.

Since the desired result of the thesis is an algorithm that can produce levels that are of certain difficulties, the algorithm will not be required to find the global optima. Levels can look different, be constructed differently, be of a desired difficulty and still be considered good enough for players. Thus the intention is to create an algorithm that creates levels that are considered good by testers, stakeholders and the writers of this thesis. The intent is not to create the best levels possible. This would be difficult to measure if tried and would probably give less variation between different levels than wanted, as the best level will be a minority in a huge search space, while it may be possible to find a lot more levels by simply lowering the requirements a bit.

1.5 Background

Active lab at the Department of Computer and Information Science (IDA) at Linköping University conducts research of gaming and interaction with regards to games for health and games for learning. One of the projects that they work on aims at creating games that activate the players by making them use their body as the main way to control the game. This is to make gaming not just a fun experience, but also to help people have a more active lifestyle. The games are web-based, meaning that they can run in an Internet browser\footnote{e.g. Google Chrome or Mozilla Firefox}. Because of this, a player only needs a web camera and an Internet connection to be able to play the game.

The game in focus in this thesis was made in an open-source framework called Phaser\footnote{http://phaser.io/}. It was extended upon by Tim Ziegenbein in a previous thesis work\cite{30}.

The game is a motion-controlled 2D platform game. The player controls a character with input generated by creating motions in front of a camera. Generating motion in different zones results in different actions, e.g. generating motion to the left or right will make the character move in the corresponding direction, while generating movement in the middle, e.g. by jumping, will make the character jump. Figure\cite{12} shows a screen capture of the game with the left and right motion zones activated.

The objective of the game is to navigate through the levels and reach the goal while avoiding obstacles such as spikes, killing enemies and gathering collectibles such as coins and di-
amonds. Killing enemies yield rewards in the form of coins which can also be found spread out in the levels, or in special boxes that the player can break. Special tokens in the shape of diamonds are spread out in the level, and collecting all in one level gives the player a bonus in the form of bonus levels.

The objects that can hurt the character are: *Spikes*: a small object attached to the ground that hurts the character when walked upon. *Spear pendulums*: a spear that periodically comes up from a small box, covering about three tiles. Walking trough any of these tiles will injure the player if the spear is up. After a specific time interval, the spear retracts into the box and the tiles can be traversed safely. *Enemies*: characters that hurts the character if collided with in any direction other than landing on top of them. Enemies can also walk along the level, but cannot jump. In the version worked on in thesis the only enemy available was a yellow bird, but more will be added during the games development.

Other objects of note are: *Moving platforms*: platforms that move in a horizontal direction until they collide with a physical object, such as a ground tile, and changes direction to go in back along the same path it came from, again, until it collides with something. *Unstable platforms*: platforms that are stationary until the character steps on them. When stepped upon, a hidden timer counts down a certain time interval, after this the platform falls until it hits a physical object, such as ground. After a short duration it resets to it’s position in the air.

In this thesis south and north going exit points are defined as *pits*. These enable the player to transition from one area to another one, located directly under or above the exit points.
A type of procedural generation already existed, developed by Tim Ziegenbein. A level structure was defined, displayed in figure 1.3 [30, p. 43], in which the level was divided into several areas called chunks. These chunks would be distinguished by type, for example a start chunk or an end chunk, and these had been constructed beforehand using rules, i.e. a start chunk needs to have a spawn-point, a top-right chunk needs to contain a pit to enable the player to reach the bottom-right chunk, and so on. Every chunk-type had a pool of pre-made chunks to choose from. This helped introduce some variation into the levels, however, with a limited amount of chunks to choose from, it would not take a long time for a player to encounter situations that they’ve seen before. The chunks also had to be generated manually beforehand, and the player path never changed.
This chapter entails important theory needed to understand the thesis work. Here, PCG, GA’s, and motion based games in general are discussed.

2.1 Procedural Content Generation

The levels in this thesis are created using Procedural Content Generation, or PCG for short. In this thesis, PCG in games is defined by Togelius et al. as “the algorithmical creation of game content with limited or indirect user input” [24]. Some examples of games using PCG is Spelunky, in which layout and contents of a dungeon is generated [15], and Dwarf Fortress, which generates an entire world (including, but not limited to, villages, events, fauna and poetry) [11]. This section provides the information necessary to understand PCG within the scope of this thesis.

Online generation

In PCG, there is a need to distinguish between between offline and online generation. Online generation is generally when content is generated when the game is actually played, while offline generation entails content being generated during the development of the game [23]. An example of online generation could be if a game contained a building, which the player needs to enter. The interior, layout and detail of the rooms in the building could then be generated in the moment the player opened the door to the building. Another example is in generation based on user experience, i.e. where the game changes during the play session to become e.g. harder or more varied, depending on how the game is being played. This means that the game is being tailored based upon the experiences of the user supplying the input to the game, meaning both how the user performs and her responses to stimuli in the game [29].

A game that uses this technique is Valve’s Left 4 Dead, which continually analyses player’s performance to see whether it should add or remove e.g. health packs or enemies [5]. Another game using this method is Cloudberry Kingdom, which uses an AI that creates a level, ensures it is feasible, and shapes the next level depending on how well the player has performed
previously. Jordan Fisher, one of the developers of the game, claims that their algorithm uses "thousands of parameters" to control the difficulty of a generated level [10].

One of the most famous examples of procedural generation and its benefits is the game Elite (1984). At the time of the game's creation, the computers in use didn't have enough memory to store the game's world space. Elite therefore generated the whole world procedurally using seeds and tables, which greatly reduced the amount of memory the game needed. [2] It's important to note that most high-profile games today only use procedural generation for small parts of the game, e.g., for creating vegetation, rather than creating full levels. There are exceptions, however, as with the game No Man's Sky, featuring an "infinite procedurally generated galaxy" [12], thereby making the game revolve around the procedural generation.

**Offline generation**

Offline generation is when content is generated beforehand and then selected and refined by a human designer before release. Using the example from earlier with the building that a player needs to enter, the inside of the building would be created with the help of an algorithm before the game is released. A designer could then look at what the algorithm created and make changes, if desired. Another example, being used in, e.g., The Witcher 3, is in creating shapes for the vegetation that is used in the game world [13].

The game used in this thesis is supposed to have a fixed set of levels, so that players can compare things such as how many levels they have completed and how much time a certain level took to complete. For this reason offline generation has been chosen for the algorithm. In online generation an algorithm should be fast and produce predictable results. These criteria are of less importance in offline generation [26]. The choice of offline generation also allows for the use of a GA since these are typically quite slow.

**Different kinds of PCG**

Togelius et al. makes a distinction between constructive algorithms and Generate and Test-algorithms. Constructive algorithms generate content using, e.g., operations that are guaranteed to lead to a solution that is considered "good enough". A Generate and test-algorithm on the other hand generates candidate content and checks it versus some sort of criteria. For example, is there a path between the starting point of the level and the end point? If this is not the case, the candidate is thrown away and the generation starts over. There is a special case of Generate-and-test algorithms called Search-Based PCG (SBPCG) algorithms, in which the solution is evaluated using certain criteria, e.g., a mathematical formula. This is the type of algorithm used in this thesis. [25]

As mentioned in section 2.1, procedural generation can be done based on user experience. A framework developed by Togelius and Yannakakis called Experience-Driven Procedural Content Generation, or EDPCG, has the purpose of coupling user experience and PCG, and describes a "generic and effective approach for the optimization of user (player) experience via the adaptation of the experienced content." [29]. The framework consists of four key components: Player experience modeling, content quality, content representation and a content generator. These are explained below.

Player experience modeling can be divided into three main classes. Subjective, objective and gameplay-based. Subjective modeling means building a model by asking the players about their experience. Objective modeling means looking more at a player's physical and emotional responses to events in the game e.g., by using sensors measuring bodily responses. Lastly, gameplay-based modeling means looking at the interaction between the player and the game, and at how the player responds to elements in the game. This modeling can be
2.2 Genetic Algorithms

Looking at quantitative measures from play sessions of the game, e.g. how many coins were collected in a level, or time spent in a level. While this class is described as being the least intrusive, it may require lots of assumptions as to what the metrics actually mean, and this may result in a faulty model.

Content quality is a measure of how well-suited certain items may be in a game's current context, with regard to the modeled player experience. Content representation means how the generated content should be represented, e.g. a vector of bits or a list of desirable features. The content generator is the part of the framework that searches for a solution, using the player experience that has been recorded. This can be done by creating or modifying a function that gives a quantitative measure of how well the generated content matches the desired content quality. \[29\]

2.2 Genetic Algorithms

In this thesis work, a genetic algorithm is used to produce levels. This section will describe the general idea of genetic algorithms.

GA’s is a version of evolutionary algorithms[^1], which draw inspiration from the notion of natural selection in the creation of a product. According to John Holland, the idea is to create software that simulates real world evolution by means of reproduction and mutation, thereby exploring a larger number of potential solutions than with conventional programs \[14\]. The basic steps for a GA are as follows \[18\]:

1. Generate a set of starting solutions, the population.
2. Calculate the fitness for each individual in the population.
3. Select a subset of the population for reproduction. Individuals with a higher fitness value have a higher chance of being selected and an individual can be selected more than once. Those that are not selected die out.
4. Select pairs of individuals to reproduce with probability $p_c$.
5. Perform crossover on selected individuals. Individuals that did not go through a crossover are passed to the next generation without modification.
6. Mutate each bit in the offspring with probability $p_m$.
7. If the number of iterations or other criteria are not met: repeat from step 2.

While there are many different kinds of algorithms that can be used to achieve procedural generation, a GA was chosen for the following reasons:

1. A GA can produce a set of solutions instead of just one. Generating a perfect level is a difficult task to accomplish. If the algorithm instead produces a set of levels, these can be analyzed to improve either the fitness function or to choose a set of levels that the game producer wishes to use.
2. Since a level in this thesis is represented as a string, the GA can operate directly on this string, thereby operating directly on the level. Since the levels generated are in 2D, a level can be represented as an array of tiles, which can be represented as a vector. Another way to represent a level could be a set of variables, for example: number of gaps in the floor, number of enemies and size of gaps, etc.

[^1]: Which is an area of evolutionary computing, see [9] for more information
3. A solution given by a GA might include designs that would not otherwise be used, even though they are valid and could be seen as good.

**Fitness**

A GA takes a population where each individual consists of several granular parts, e.g. a sequence of zeroes and ones. All of the individuals in this initial population are seen as possible solutions, but since they are generated randomly, it’s easy to imagine that some parts could be randomly generated better than others. All of the individuals in the population are evaluated with a fitness function (a.k.a evaluation function, see [19, p. 20]). This function will be different depending on what goal is to be achieved with the algorithm, i.e. in which direction the population is to be shaped over a number of generations. A sequence that is awarded a high fitness value by the function will be seen as a strong member of the population, and has a higher probability of being used in the next step, the crossover. The sequences that get a low value from the fitness function are considered weak and have a higher probability of dying off.

Explaining with an example, one might be wanting to generate a bit sequence consisting solely of ones. A sequence with only zeroes would get a low fitness value, while a sequence with many ones would get a higher value, therefore having a higher probability of being used in the crossover step.

**Method of selection**

When fitness values have been calculated for the individuals in a population, a method is used to determine which individuals should be used to create offspring. Simply choosing the individuals with the highest fitness in every iteration could cause the algorithm to end up in a local optimum. This is due to the possible presence of so-called super individuals, whose fitness are much higher than the average individual fitness [19, p. 59]. This high fitness can cause the individuals to be chosen more often for re-population, and might lead to low genetic diversity after just a few generations. To try and avoid this, a probabilistic method is used that selects the individuals to breed and guarantees that all individuals have a chance of being chosen, even the ones with a low fitness value (albeit a lower one).

The method used in this thesis is called tournament selection, which randomly chooses a subset of individuals to compete in a tournament [19, p. 61]. The individual that has the highest fitness value in the tournament is selected as a winner and will be a candidate for reproduction. This is done as many times as needed to keep the population at the right size. This method of selection means that good solutions are more favored to be kept while still allowing solutions with lower fitness to stay alive (if they are compared to even worse solutions). This helps with mitigating the risk of getting stuck in a local optimum.

**Crossover**

A crossover, or reproduction, occurs between two items in the population with a certain probability $p_c$, creating offspring that will replace the parents [19, p. 17]. In the case of the bit-sequence example, the crossover between two individuals will occur in some point of the sequence, potentially selected randomly. This crossover will create two different sequences. One will have the part of the first item before the crossover point, and the rest of the sequence from the second item. The second will have the part of the second item before the crossover point, and the rest of the sequence from the first item. If the crossover point is after the fourth bit, we would get: (1110 0110) X (0001 1101) = (1110 1101, 0001 0110) as the offspring.
2.2. Genetic Algorithms

Mutation
The next step is the mutation step, where each part of the offspring is mutated with a certain probability, \( p_m \). In the bit-sequence example, the mutation could be to simply invert one bit in the offspring’s sequence. These new possibly mutated sequences will make up the new population, and be used in the same manner as the previous generation (i.e. the algorithm restarts, but with the new generation as the population).

Variables
For a GA to function well it needs a fitness function, i.e. a mathematical formula that in some way describes how well a solution fits to the problem it’s trying to solve. In this thesis, the problem is to create levels where the difficulty rating can be controlled. A set of controllable features that correlate with players experience of challenge were found in a paper by Pedersen et al. [21], where a modified version of Markus Persson’s Infinite Mario Bros was used as a testbed. These are:

- C: Whether or not a level was completed.
- \( n_p \): Number of blocks the player pressed over the total number of blocks existent in a level.
- \( d_j \): Number of times the player died by jumping into a gap over the total number of deaths.
- \( d_g \): Number of times the player died by jumping into a gap.
- \( J_d \): A jump difficulty heuristic.
- \( E_{Gw} \): Average width of all gaps in a level.
- \( n_d \): The number of times a player ducked with Mario.
- \( t_{ll} \): Time spent on last life over the total time spent on a level.
- \( n_c b \): The number of coin blocks pressed by the player over the total number of existing coin blocks in a level.
- G: The number of gaps in a level.
- \( H_g \): Spatial diversity of gaps placed in a level.

The correlation coefficients are presented in table 2.1 defined by Pedersen et al. [21]. Note that the feature \( H_g \) is not included in the table since no correlation coefficient was presented for this feature. It was however noted in the paper that this feature had a smaller but still significant correlation than the features presented in the table.

There is a classical saying that correlation does not imply causation. Practically, this means that just because one variable seem to cause a change in the other, this is not necessarily the case. Therefore it can not be stated that these variables will affect the challenge of a level, but they may. During the course of the thesis work, tests will be performed to measure if the fitness function, which will rely on some of these features, actually affects the perceived challenge of created levels. The goal of this thesis is not to prove causality between challenge and these features, but to create an algorithm and a fitness function that controls challenge. Therefore these features should be a good starting point.
### 2.3 Validation of levels

#### Table 2.1: Correlation between features and challenge

<table>
<thead>
<tr>
<th>Feature</th>
<th>Correlation coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>-0.600</td>
</tr>
<tr>
<td>(n_p)</td>
<td>-0.480</td>
</tr>
<tr>
<td>(d_j)</td>
<td>0.469</td>
</tr>
<tr>
<td>(d_g)</td>
<td>0.447</td>
</tr>
<tr>
<td>(j_d)</td>
<td>0.439</td>
</tr>
<tr>
<td>(E(G_w))</td>
<td>0.409</td>
</tr>
<tr>
<td>(n_d)</td>
<td>-0.368</td>
</tr>
<tr>
<td>(t_l)</td>
<td>-0.312</td>
</tr>
<tr>
<td>(n_c,b)</td>
<td>-0.292</td>
</tr>
<tr>
<td>(G)</td>
<td>-0.287</td>
</tr>
</tbody>
</table>

#### Curse of dimensionality

The curse of dimensionality \[6\], refers to the exponential increase in search space that occurs when adding dimensions to a problem. If each variable has many possible configurations, the amount of different configurations will grow in an exponential fashion. This will quickly lead to a configuration space that is too big to test. Depending on the amount of variables found in this thesis, this may or may not become a problem. Therefore it would be wise to keep the number of introduced variables as small as possible to mitigate this.

#### Fitness function

An important part of the GA is the fitness function. The fitness function helps to decide which levels will live on to the next generation and which levels will be thrown away. Therefore a fitness function that keeps the goal of the thesis in mind is needed. This goal is for the fitness function to be able to evaluate the difficulty of the levels it produces. Thus a higher fitness should mean higher difficulty.

In a paper on level design, Hector Adrian and Ana Luisa proposes using the difficulty of a level as the fitness function in a GA. They propose using a function that calculates the difference between the wanted difficulty and the actual difficulty of generated content to achieve a fitness function that is independent of the type of game for which it is used. \[8\] The actual difficulty in this context will be the perceived difficulty reported during user tests. The values calculated by the fitness function will thus have no meaning until the variables have been adjusted to correlate to the perceived difficulties. This also means that the fitness calculated will change when the variables are adjusted, meaning that a certain fitness in one iteration may closely correlate to a certain difficulty while, in another iteration, the same fitness may correlate to an entirely different difficulty.

Therefore a fitness function will be used that takes the difficulty into consideration, calculated with the help of the variables in subsection 2.2 (along with variables that may be discovered later) as well as whether the level is possible to finish or not. The weight of these variables will be adjusted to improve the results. By incorporating whether or not a level is feasible in the fitness function, it will be possible to increase the probability of feasible levels to be chosen for the next generation.

#### 2.3 Validation of generated levels

The levels will be divided into chunks of a certain size, and these will be generated individually. The chunks will have start- and endpoints, can refer to the previous and next chunk
2.4 Motion based games

(if they exist) and be connected to a specific physical layout, fitting for the type of chunk (depending on the direction the player should move in).

**Drunkard walk algorithm**

The position of the chunks relative to each other will be generated before the GA using the drunkard walk algorithm. The algorithm imagines a grid-system, randomly picks a cardinal direction and moves there, marking the space it moves into as occupied. It then chooses another random direction and will move there, provided that it has not occupied that space previously. The procedure is repeated until a set amount of spaces are marked as occupied. Since the player uses an arm to generate movement in a certain direction, the levels created should have a randomness in direction to them in order to ensure that the player uses both arms. The result of this step will be used to determine what type of physical layout the chunks will have.

**Finding layouts**

When the chunk has been classified, a population of physical layouts are generated. This population is seen as the first generation and are used as input to the GA. The algorithm refines the population and when it finds a chunk to have a fitness close enough to the desired fitness it returns the chunk. This individual will represent the physical layout tied to the chunk. If the desired fitness is not achieved the algorithm will stop after a predetermined number of iterations and run again. A problem can arise if the chunk is seen as good enough by the genetic algorithm, but it contains some elements that should be removed, e.g. misplaced spikes or unfair spikes. Spikes could be considered unfair if the character is able to be hurt or even killed by them without warning. Ernest Adams, in his book *Fundamentals of Game Design*, calls this *learn-by-dying*-designs, meaning a game being designed in a way such that the character has to die to know what to do (or rather, what not to do). Repeated death is not seen as fun and elements that can lead to this type of game design should be removed. [3, p. 417] If these elements are present after the GA is finished, they elements will be removed. This can lead to a good solution being turned into a bad one, e.g. if the removal of a spike suddenly makes the level too easy. If this happens, the search starts over.

The SBPCG part of the algorithm is prominent in that we want to ensure that when a chunk is chosen to be used in the level, it should be possible to finish, and we rather try to refine the candidates than simply throwing them away. This allows for the incorporation of this important characteristic into the fitness function. Since the test to check whether a chunk is feasible or not is executed many times, this check should be as efficient as possible, so as not to become a bottleneck in the algorithms execution time.

2.4 Motion based games

Motion based games, refers to games controlled by the player by movement of their body instead of for example using a controller.

If these games can provide exercise for players, it stands to reason that they will have a positive effect on e.g. obesity and sedentary related diseases. In a study by Whitehead et al. which surveyed studies made on exergames, it was found that games that promoted full-body movement resulted in more exercise. [28] Because of this, there are certain design aspects that one needs to take into consideration when designing levels for this type of game. Many popular platform games tend to move in one direction e.g. Super Mario Bros in which the player mainly moves from left to right. In a motion based game, this would mean that the player would use the right side of their body to a larger extent than the left. This may
make the game boring or make the player tired, since the movements would be repeated for a long time. It would be better to create levels that move in all directions in order to provide a natural pause for body parts of the player, while also attempting to engage as much of the players body as possible.

Reaction time when playing motion based games can differ from controller-based games, depending on the input mechanism and the players having low experience with the unusual form of game control. Thus the difficulty of a motion based game may be considered harder when comparing the same level layout compared to a game played using a controller.

2.5 Player Experience of Need Satisfaction

A problem that can arise when developing games is knowing what makes the game "fun". Richard Ryan and Scott Rigby argue that there are many factors that control if a game is perceived as fun or not, and that ultimately it is down to satisfying the player’s psychological needs, no matter who the player is. [23] In a report, Richard Ryan, Scott Rigby and Andrew Przybylski uses something called Self-determination Theory, or SDT, that can be used to evaluate players motivation to play a game. [7] A distinction is made between intrinsic and extrinsic motivation. Intrinsic motivation being when someone is motivated to act simply because the act is found to be satisfying, for example going out for a jog. Extrinsic motivation is, opposite intrinsic motivation, when an act is performed to get some external reward, e.g. if the only reason for going jogging is to lose weight. [22] Looking at computer games, it is, according to SDT, clear that the motivation is mainly intrinsic. Players (usually) need to pay to be able to play games and usually don’t get rewards or approval for playing. [7]

To measure need satisfaction in game play, Ryan, Rigby and Przybylski came up with a measure called Player Experience of Need Satisfaction, or PENS. This measure consists of several factors that account for different psychological needs. The PENS variables are: Autonomy, measuring the amount of choices a player has. Competence, measuring if the game presents a challenge while not being overwhelmingly difficult, and the perceived efficiency of the user. Relatedness, a persons connection to other players. Presence/Immersion, the sense that the player is actually inside the game world and finally Intuitive Controls, which measures the user interface with regards to moving through the game world. [23] The theory is that high reported values of these needs may indicate that a game is fun and motivating to play, and that a lower value in a specific need may help pinpoint a flaw in the game design, prompting a change in some particular area.

2.6 Iterative development

In the iterative enhancement model, which is to say, an iterative method of development, the design team makes several iterations of the same product while constantly getting feedback from users. This allows the developers to start with an initial idea of how the system should work (e.g. developed in meetings with a client), create a product using this idea, and then refining the product over several iterations. Victor R Basili et al. states that using this model "allows the developer to learn through each cycle of development and the user to provide timely essential feedback, improving each version until the final version of the system is produced” [4, pp. 6-7]. Since the end user in this case is a player of a game, and, as described in section 2.5 the user will make the choice of playing the game because they are motivated to do so, the product needs to take the end users’ feedback into consideration.

An iterative process has a lower risk factor compared to methods such as the waterfall method, since possible risks and problems with the software used are identified earlier [17].
A case study by Jorge Osorio et al. showed that planning a project on an iterative basis made it simpler to make changes to the process, should the need arise [20, p. 455].
The method used to implement and refine the GA as well as evaluating the results is proposed in this chapter. The method contains a set of tasks to be completed first, followed by an iterative step. An iterative method was chosen since it was not known what the features of the fitness function should be beforehand, which lead to having to try out different settings and seeing what worked and what didn’t. The iterative approach allowed for small changes in the design until a setup was found that matched the results sought. Any issues that were discovered resulted in tasks for the next iteration.

Before the iterative step of testing the algorithm and its’ fitness function could start, the stage needed to be set for this to work. Because of this, there was a pre-study where the GA and the algorithm for checking if a level is feasible were implemented. A set of variables were identified from other, similar work (described in the theory chapter), that were believed to impact the difficulty of the resulting levels. Examples of variables could be e.g. the average width of all spikes placed in the level or the number of spikes placed in the level.

When these steps were completed the iterative step was started. This step consisted of the following tasks:

- Test new variables identified in previous iteration.
- Update fitness function to include positive variables and exclude those that made the results worse.
- Adjust variables of fitness function with Matlab.
- Create levels for user tests.
- Conduct user tests.
- Adjust current variables to fit results from the user tests.
- Identify new variable candidates
3.1 Research methods in similar work

Since the algorithm that was designed and implemented in this thesis work should be able to control the difficulty of a created level, it was important that the difficulty of generated levels were correctly translated to the difficulty that would be perceived by the end user. Because of this user tests were conducted on the algorithm at the end of the iterations, to ascertain that the difficulty is controllable in way that satisfies the end users need for challenge. “User-centered design’ (UCD) is a broad term to describe design processes in which end-users influence how a design takes shape” [1], and user tests are a part of UCD. By comparing the perceived difficulty by users and the difficulty the algorithm believed a level to be, it was possible to adjust the algorithm to fit the findings in the user tests. The same tests also measured the users ability to control the game as well as immersion etc. which may be used to measure the amount of variation perceived by users.

The method used to create and refine the GA is similar to the framework described in section 2.1 i.e. that the GA was shaped with player experience. The player experience was modeled using subjective data gathered from the user tests. The content quality was measured using the fitness function, since it looked at elements in the levels and gave subjective measure. Content representation can be described as how the levels were represented in the GA, i.e. a vector of integers. Lastly, the content generator was the genetic algorithm that was updated according to the player experience model. During the tests, a gameplay player experience model was also constructed using the metrics collected by the game during the play session. These metrics were e.g. coins collected or time spent in the level.

3.2 Representation of levels

Before this thesis work, a map editor called Tiled was used to create levels. It allows a designer to load graphics and use these to create levels by hand. The output of the algorithm created in this thesis is generated to match the output of Tiled. This makes it possible to open generated levels in Tiled so that human designers can refine them, which is a desirable part of offline generation. Offline generation also allows for the combination of parts of levels. For example if a designer likes certain parts of a generated level these can be combined with other parts in Tiled, making the generated levels more versatile in their use.

3.3 Implementation of genetic algorithm

As explained in the theory chapter, a GA consists of a fitness function, a selection step, a reproduction step and a mutation step. The selection step chooses which solutions will be used in the reproduction step based on their calculated fitness. Tournament selection, described in 2.2 was implemented for the selection step.

The reproduction step chooses a set of the solutions gathered from the tournament selection with a probability $p_c$ and performs crossover operations on these solutions by dividing them into two halves of a level and merging these halves with halves of other solutions.

Lastly, the mutation step iterates through each position in each solution and mutates the position into either a spike or an air tile with probability $p_m$. As the fitness function only regards spikes is was decided that the mutation step should have the ability to mutate positions into spikes and spikes into air.
3.4 Implementation of feasibility algorithm

An algorithm was implemented to check whether a generated chunk was feasible. This algorithm was inspired by the flood fill algorithm. The algorithm looks for the starting position of a chunk and marks positions depending on how far the character is able to get from the position. Tiles that the character can stand on and get to are pushed to a vector and will be used to continue the check of the chunk, i.e., they will be used as new starting points to mark from. This is only done if the tile has not been used as a starting point in an earlier iteration. If the position in the chunk marked as the goal is reached, the algorithm returns that the chunk is feasible. If all reachable positions in a chunk have been marked and the goal remains unmarked, the chunk is seen as impossible.

The in-game physics allow the character to jump four tiles high. If there is a spring on a reachable position, the character can reach up to eight tiles high when jumping from this position. This is also reflected in the level-checking function.

Each chunk in the level is tested individually. Firstly during the process of the GA, since the possibility to finish a level has an impact on fitness. This means that all individuals in the population used to select the chunk will be tested, every generation. Secondly, a chunk that has been selected by the GA is tested along with its neighbour, to see that it is possible to make it through the combination of chunks.

3.5 Identification of relevant variables

In a paper by Barbara Kitchenham et al., a set of guidelines are proposed for use when designing an experiment similar to the one that was conducted in this thesis. The first guideline is: "Identify the population from which the subjects and objects are drawn" [16]. In this case, the subjects are the variables that are believed to be relevant to achieving difficulty in procedurally generated levels for a 2D platform game. As such, all elements of a 2D platform game can be viewed as the population from which the selected variables should be drawn. It would be impossible to list all possible factors that may effect the result in the desired manner.

In the theory chapter a set of variables were identified and used as a starting point. However, many of these are not relevant to the game used in this thesis, e.g., $n_d$: the number of times a player ducked. It is irrelevant as there is no way to duck in the game used in this thesis. The variables that were seen as relevant and therefore used in the fitness function for iteration 1 were:

- $E(G_w)$: Average width of all gaps in a level.
- $G$: The number of gaps in a level.
- $H_g$: Spatial diversity of gaps placed in a level.

These variables were identified in a study using a Super Mario game. In Mario, a gap is an obstacle that the player needs to overcome in order to survive. If Mario falls into a gap, he dies and you have to restart the level. In the game used in this thesis there are no such gaps. Instead there are spikes that inflict damage on the character if they are stepped/landed upon. These spikes were used instead of gaps to calculate this variable, where a spike one tile in width is considered equivalent to a gap of width one. The variable $G$, as shown in table 2.1, has a negative correlation value. It is argued in the cited report that an increased number of gaps imply a linear decrease in challenge. However this did not seem to be case in the

---

1Flood fill works similar to the bucket fill tool in your favourite painting application
game used in the thesis. Because of this, the correlation for number of gaps was started off as positive. Tests were then run to find the best value of the variable and, depending on the results of the user tests, the correlation values could change between iterations.

### 3.6 Iterative step - Test, refine and evaluate the algorithm

The thesis work was done in an iterative process. First a pre-study was performed where the algorithm was implemented. After this the iterative process started and user tests were conducted. These user tests provided data needed to improve the algorithm. In each iteration any new variables that might have an impact on difficulty were implemented. Lastly the variables currently included in the algorithm were adjusted and fitted to the perceived difficulty curve found in the user tests. All steps are described in more detail below and an overview is shown in figure 3.1.

![Iterative process](image)

**Figure 3.1: Overview of the iterative process**

#### Evaluate variables

With the results of the previous step the variables and their effect on the difficulty were evaluated. If the temporary removal of a variable did not impact the resulting difficulty significantly, or if inclusion of the variable was found to make the results worse, it could be considered ineffective and dropped from the set. Conversely, if the addition of a new variable made the results better, it was added to the fitness function, and subsequently used when creating the new set of test-levels.

#### Evaluate results

The set of variables were adjusted over two iterations and an end phase. The tests were initially run with the goal of finding out if the magnitude of change in difficulty reflected the numbers in table 2.1. After testing with the levels resulting from the use of these coefficients, the goal was to find out whether some variable should be removed or added, finding a new configuration using the new set of variables, and creating new levels using the new configuration. The configuration tests were set up in such a way that when one variable was changed, the others were kept constant. Sets of levels were then created with the best configuration and used in the user tests of the next iteration to see if the perceived difficulty matched the one measured by the fitness function.
3.7 User tests

Adjust the fitness function

With the knowledge of how the variables effect the outcome the fitness function of the GA was adjusted to reflect the new configuration. By adjusting the fitness function to include the effect the variables have on the result, as many variables as possible could be excluded. The desired result was an algorithm where the only variable used as input is difficulty. This was to keep the algorithm as generic as possible while at the same time minimizing the impact of the curse of dimensionality.

The variables were adjusted by using Matlab. First, the result of a configuration was normalized to values between 0 and 1. A straight line between 0 and 1 was created and the difference in area between these two was the result measured. If the fitness function for a certain configuration differed by 0 from a straight line, it could be considered perfect, since the relationship between difficulty and fitness could then be described in a linear fashion.

Iteration through all possible values for all variables between -2.0 and 2.0. with a precision of two decimal points was conducted for iteration 1, the limits were for iteration 2 and the end phase changed to -1.0 and 1.0, and the difference for each configuration was calculated, finally choosing the best one as the start for the next iteration. The code used for this is presented in Appendix A.

Evaluation of the iterations

Iterations were evaluated using three sources of data. By evaluating the fitness functions effect on challenge and how well it achieved desired grades of challenge, personal reflection and testing with stakeholders. By comparing these three sources to the planned levels of challenge, the results of the evaluation acted as a foundation for the next iteration, meaning that information was extrapolated from both the tests and the measurements to set up tasks for the next iteration.

End criteria for the iterative step

Since there is no clearly defined limit as to how good an algorithm can be at creating variation or difficulty in a level, the end criteria of the final iteration was decided to be in the end of the thesis, with regards to the time limit of the thesis.

3.7 User tests

User tests were conducted to ensure that the algorithm produced levels that were equivalent to user expectations, i.e. to see if the levels that were created by the algorithm were as hard as the algorithm claimed. This was crucial, since difficulty of a level is supposed to be the only input to the level generator. Six levels were generated and used for all user tests in one iteration.

The test was designed to take about 15 minutes, to ensure that the subject did not grow bored or tired of playing, and were conducted as follows: The subject filled in a special consent form, along with a demographic questionnaire. They were informed about what the thesis was about, along with some information about the game. The player got to play two minutes on a test level, one of the levels that was part of the original game. This was to get a feel for the controls in the game. After the two minutes were up, the player was informed that they would play six short levels. These levels were, on completion, rated on a scale of 1-10, depending on how hard the player perceived the level to be. All testers in an iteration were presented with the same levels and in the same order. Aside from the rating, some additional
information was recorded after each level; amount of deaths in the level, the time spent in the level, number of coins picked up and possible comments on the level.

After all the levels had been played, the player filled in a questionnaire made using the Player Experience of Need Satisfaction model (PENS). The model was used to measure some metrics in the game, like Competence, Autonomy, Intuitive Controls, etc.

**PENS**

To measure involvement in the game, the players were presented with a questionnaire created with specific guidelines and administered post-play. It was presented such that the user rated their level of agreement to items using a 7-point Likert scale, ranging from 1, Do Not Agree, to 7, Strongly Agree (with some items having the scale flipped, something that was kept in mind when calculating the average). The Cronbach alpha of the items along with a confidence interval is shown in the presentation of the results. The items were presented in a randomized order.

The PENS-variables, explained in more detail in section 2.5 were: Autonomy, Competence, Presence/Immersion and Intuitive Controls. Relatedness was seen as irrelevant, since it measures e.g. relatedness to other players or characters, something that was not yet a part of the game, since it was in development.

Because of a mistake when creating the questionnaire for the first user test, an item in the Intuitive Controls variable was missing ("Learning the game controls was easy"). This means that the variable may be less accurate, since quantitative information is missing. It was still presented, using the questions that were present, however more weight was put on the observations that were made during the test. The question was added to the second user test.

After the questionnaire was filled in, possible comments on what was regarded as hard/easy were written down. This was to find out if there were things that were missing in the game, or if something monumental should be changed in the algorithm.
This chapter describes the results achieved in the pre-study, the iterations that followed and the end phase. For each iteration there was a set of goals. These goals are presented here and the results compared to them.

4.1 Pre-study

The following goal was set for the pre-study: Implement a GA that can change the structure of a level. At the end of the pre-study, the goal had been achieved.

The program started by first creating a number of chunks and putting them in a kind of virtual grid. This was used to create levels with chunks that can go in any of the cardinal directions relative to each other. The virtual grid was used to get an idea of how the layout of the level would be, and was generated randomly.

Each chunk was then connected to a population of physical layers, each consisting of a possible player path based on a block structure as might be seen in e.g. Super Mario. These physical layers were put in the GA, and the output of the GA (the "winner") decided how the chunk would look. Each chunk was tested to see that it was feasible, i.e. it is possible to get from the start to the end of the chunk. If this was not the case, the layer would be regenerated. For all chunks, the algorithm checked that it was possible to get from the start of the current chunk to the end of the next chunk. If this was not possible, it meant that there was something wrong with the current layer and that the GA should start over with this chunk.

After all chunks were connected to a layer, they were put together into a big level and written to a json-file. It was decided that this implementation should be kept, since it enabled the possibility to open the levels in the level editor Tiled. If a level was generated and a change was desired, the level could simply be loaded into the map editor and changes could be made with ease.
4.2 Iteration 1

Evaluation

The pre-study was evaluated by discussion and testing. All included agreed that the goal set for the pre-study had been met. The goal in itself needed to be completed in order to start adjusting a fitness function to achieve controllable challenge. However, the pre-study itself had no direct connection to the research question and therefore the evaluation mainly focused on extrapolating goals for iteration 1. During the evaluation together with stakeholders the current state of the generator was discussed and the stakeholders mentioned a few things that they would like to see in the next iteration. These where: springs, checkpoints and enemies. They also commented on certain levels where the player could get into situations that were impossible to get out of, as well as spikes that were placed where the player could not be hit by them. Both of these were seen as unwanted. From this meeting a set of tasks were decided upon.

Tasks for iteration 1

The following goals for iteration 1 were agreed upon:

- Make sure that no level generated could result in an impossible situation.
- Introduce springs and checkpoints into the generator.
- Ensure the removal or repositioning of spikes that cannot be reached.
- Introduce enemies into the generator.

4.2 Iteration 1

Following up from the tasks decided upon when evaluating the pre-study, the first task to tackle was generation of levels where the player would end up in an impossible situation. This was remedied by creating functionality to find impossible situations as well as functionality for "fixing" them in a randomly selected way, making it impossible to get stuck. This was done by e.g. putting down springs to allow higher jumps or creating a staircase out of ground tiles.

Checkpoints, coins and enemies were also introduced into the generator. Including more items meant changing how objects were represented in the program, since some of the objects need to have properties (e.g. spikes have different angles depending on placement). The sprite sheet used in the game was found to be outdated and did not contain all enemies that were planned to be used in the game. Therefore the only enemy implemented was the bird enemy.

As per the task described above, functionality for removing and/or moving unreachable spikes was implemented.

Evaluation

Iteration 1 was evaluated partly by the same means as the pre-study, i.e. by discussion and testing with the stakeholders. Aside from this, a user test was conducted with twelve testers (ten male and two female) from the campus of Linköping University. Testers were mostly students working with Active lab. No incentives were offered for participating.

The user tests were conducted with six generated levels with fitness values ranging from 1.107 - 2.367. Figures 4.1, 4.2 and 4.3 are examples of how easy, medium and hard chunks looked in the tested levels.
The average perceived difficulty is presented in table 4.1. The order in which the levels were presented to the subjects was determined beforehand. The level with the highest fitness was placed last, and the level with the lowest fitness was placed first, while the order of levels two through five was randomized.

Players reported that small landing spaces between spikes raised the perceived difficulty. Therefore this was seen as a candidate to become a new variable in the fitness function. Checkpoints were found to be placed too close to each other. Players were found to often die from falling down on spikes that were impossible to see beforehand. Lastly, when demon-
4.2. Iteration 1

The results of the stakeholders, a desirable feature that was missing was decorations, e.g. trees in the foreground and the background.

A plot showing average perceived difficulty versus reported fitness value is shown in figure 4.4. The difference calculated for the starting configuration was 0.1393.

![Figure 4.4](image)

Figure 4.4: Fitness of level vs. average perceived difficulty of level (starting config)

First the existing variables were examined and changed according to the information gained from the tests. This resulted in the graph presented in figure 4.5 with a calculated difference of 0.0392.

![Figure 4.5](image)

Figure 4.5: Fitness of level vs. average perceived difficulty of level with adjusted variables (iteration 1)

The results of the PEN5-questionnaire is shown in table 4.2.

1*One question from the Intuitive Controls category was missing due to a mistake.

Table 4.1: Fitness values and perceived difficulty in first user test

<table>
<thead>
<tr>
<th>Level</th>
<th>Fitness</th>
<th>Perceived difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.107</td>
<td>2.792</td>
</tr>
<tr>
<td>2</td>
<td>1.493</td>
<td>4.583</td>
</tr>
<tr>
<td>3</td>
<td>1.274</td>
<td>3.500</td>
</tr>
<tr>
<td>4</td>
<td>1.880</td>
<td>5.875</td>
</tr>
<tr>
<td>5</td>
<td>1.561</td>
<td>4.625</td>
</tr>
<tr>
<td>6</td>
<td>2.367</td>
<td>6.333</td>
</tr>
</tbody>
</table>
Table 4.2: Results of PENS in iteration 1

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
<th>Cronbach’s Alpha</th>
<th>95% Confidence Interval (±)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intuitive Controls</td>
<td>5.665*</td>
<td>0.90</td>
<td>0.60</td>
</tr>
<tr>
<td>Presence/Immersion</td>
<td>3.42</td>
<td>0.85</td>
<td>0.35</td>
</tr>
<tr>
<td>Autonomy</td>
<td>4.08</td>
<td>0.88</td>
<td>0.53</td>
</tr>
<tr>
<td>Competence</td>
<td>3.75</td>
<td>0.75</td>
<td>0.52</td>
</tr>
</tbody>
</table>

The relationship between perceived difficulty, number of deaths and average time to complete a level is shown in figures 4.6 and 4.7.

![Figure 4.6: Deviation from average number of deaths vs. average perceived difficulty of level (iteration 1)](image1)

![Figure 4.7: Deviation from average time vs. average perceived difficulty of level (iteration 1)](image2)

Tasks for iteration 2

After inspecting the results and reviewing comments made by test subjects a set of tasks were decided that would be the main focus of iteration 2:

- Ensure spikes are not placed directly under a pit.
- Include new variable into the fitness function that takes the landing space after spikes into account when calculating the difficulty.
- Introduce decorations into the generation of levels.
4.3 Iteration 2

Functionality was implemented that made sure that spikes no longer could be placed under pits as well as functionality for putting trees into the decoration layers.

A way to measure distance between spikes was added and incorporated into the fitness function. Two ways were tested. The first was calculating the average width of landing spaces after spikes, and the second was finding the smallest landing space after spikes. Both only checked inside a single chunk. After testing, the second way was chosen. The new variable was calculated using the following formula:

\[
\frac{1}{\log N(\text{SmallestLandingSpace})}
\]

The new variable was introduced and testing in Matlab was conducted to see which configuration was the best with the new variable. The result of running this configuration on the test levels from iteration 1 is presented in figure 4.8. The difference calculated with the new variable and adjustments made was 0.0315. The new configuration of variables is shown in table 4.3.

Between perceived difficulties 3.5 and 5 there is a decline followed by a sharp incline, as seen in figure 4.8. Because of this more levels will be produced around these fitness values for the next test, to see if this problem persists.

![Figure 4.8: Fitness of level vs. average perceived difficulty of level with new variable and adjusted variables](image)

Table 4.3: Variables in fitness function

<table>
<thead>
<tr>
<th>Variable</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E(G_{iw}))</td>
<td>1.590</td>
</tr>
<tr>
<td>(G)</td>
<td>0.388</td>
</tr>
<tr>
<td>(H_g)</td>
<td>1.310</td>
</tr>
<tr>
<td>(S)</td>
<td>1.120</td>
</tr>
</tbody>
</table>

Difficulties were changed to be set individually for each chunk instead of the whole level, enabling a level to have varying difficulties between chunks. Difficulty thresholds were introduced in the algorithm, meaning that different features are introduced at different difficulties. This enables a sort of progression in the levels. The thresholds are shown in table 4.4. Aside from the spikes, the features do not impact the fitness of levels.
Table 4.4: Features and difficulty thresholds

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Trees in background</td>
</tr>
<tr>
<td>1</td>
<td>Spikes</td>
</tr>
<tr>
<td>3</td>
<td>Trees in foreground</td>
</tr>
<tr>
<td>3</td>
<td>Enemies (Chicks)</td>
</tr>
<tr>
<td>5</td>
<td>Spearpendulums</td>
</tr>
<tr>
<td>6</td>
<td>Moving platforms</td>
</tr>
</tbody>
</table>

**Evaluation**

Iteration 2 was evaluated with a user test, to ensure that the configuration of the fitness function reflected the difficulty of the created levels. For the test, six levels of varying difficulty were created. Levels were created with a focus around difficulties 3-5. The levels were created to have difficulties according to the “calculated difficulty” column in table 4.5 and presented in this order to the user.

Table 4.5: Fitness values and perceived difficulty in second user test

<table>
<thead>
<tr>
<th>Level</th>
<th>Fitness</th>
<th>Calculated Difficulty</th>
<th>Perceived difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.824</td>
<td>1</td>
<td>2.333</td>
</tr>
<tr>
<td>2</td>
<td>4.624</td>
<td>4</td>
<td>5.542</td>
</tr>
<tr>
<td>3</td>
<td>5.361</td>
<td>5</td>
<td>5.583</td>
</tr>
<tr>
<td>4</td>
<td>4.176</td>
<td>3</td>
<td>2.333</td>
</tr>
<tr>
<td>5</td>
<td>5.959</td>
<td>6</td>
<td>6.25</td>
</tr>
<tr>
<td>6</td>
<td>6.683</td>
<td>7</td>
<td>7.333</td>
</tr>
</tbody>
</table>

The test was conducted with twelve testers (eleven male and one female) from the campus of Linköping University. The tests were conducted with a mix between people who had tested the game earlier and people who had never seen the game before. The tests were conducted with six generated levels with fitness values ranging from 2,824 - 7,333, and rated in the same way as the first test.

Figures 4.9, 4.10 and 4.11 are examples of how easy, medium and hard chunks could look in this iteration. The results of the difficulty thresholds can be seen in the figures. Figure 4.9 is from the first level in the test, which had a difficulty rating of 1. A spike, a springboard and some trees in the background decoration layer can be seen.

Figure 4.9: Easy chunk (iteration 2), Fitness: 2.79504
4.3. Iteration 2

The chunk shown in figure 4.10 is part of the second level in the test, which had a difficulty rating of 4. In the chunk, it is possible to see more spikes, trees both in the foreground and background, and an enemy.

Lastly, figure 4.11 is a chunk in the final, and hardest, level. This level had a difficulty rating of 7. In the image a spear pendulum, more spike groups with less space between them, and trees in both the foreground and background can be seen. There are also enemies in the level, but not in this particular chunk.

Spear pendulums were sometimes placed directly under pits, and spikes could sometimes be placed very close pits (but not directly underneath). Both of these could result in unfair deaths.

It was not immediately apparent for players encountering pits for the first time that the goal was actually to fall down in them, not avoid them.

After the tests were conducted, the perceived difficulty of the levels was put in relation to the calculated fitness of the levels. The results are shown in figure 4.12 with a calculated difference of 0.1735.

After adjustment of the variables the difference dropped to 7.9223e-05. The result is shown in figure 4.13. The values of the adjusted variables are shown in table 4.6.

The results of the PENS-questionnaire are shown in table 4.7.
4.3. Iteration 2

Data collected during the tests lead to the following plots:

All of these plots use difference in perceived difficulty on the x-axis. This means the deviation of the reported difficulty from the difficulty used to create the level.

In figure 4.14 we see the deviation from the average number of deaths on each level against the difficulty.

The deviation from average time to complete a level is shown in figure 4.15.

Figure 4.16 shows the deviation from average number of coins picked up in a level against the perceived difficulty.

The levels perceived as being two steps easier had a coin pick-up rate of about 39% while the levels perceived as being two steps harder had a pick-up rate of about 59%.
Table 4.7: Results of PENS in iteration 2

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
<th>Cronbach’s Alpha</th>
<th>95% Confidence Interval (±)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intuitive Controls</td>
<td>5.28</td>
<td>0.72</td>
<td>0.51</td>
</tr>
<tr>
<td>Presence/Immersion</td>
<td>2.94</td>
<td>0.78</td>
<td>0.28</td>
</tr>
<tr>
<td>Autonomy</td>
<td>3.08</td>
<td>0.64</td>
<td>0.51</td>
</tr>
<tr>
<td>Competence</td>
<td>3.81</td>
<td>0.90</td>
<td>0.61</td>
</tr>
</tbody>
</table>

Figure 4.14: Deviation from average number of deaths vs. difference in perceived difficulty

Figure 4.15: Deviation from average time to complete level vs. difference in perceived difficulty

Tasks for the end phase

After inspecting the results and reviewing comments made by test subjects a set of tasks were decided that would be the main focus of the final phase of the thesis:

- Remove the possibility of spear pendulums being placed directly under pits.
- Make sure spikes aren’t placed directly next to pits.
- Insert coins in pits to make it a little more obvious that the character can jump down into them.

4.4 End phase

Because of events that were out of our hands, the evaluation in the end phase was only conducted by the developers.
4.4. End phase

Figure 4.16: Percentage of coins picked up in a level vs difference in perceived difficulty

When the configuration of variables found in iteration 2 were used to create levels, they were found to contain some strange elements. The configuration could, for example, put a large number of spikes in a level supposed to be created with a difficulty of 1. Because of this, the procedure to find a configuration was run again, this time with the information from both iteration 1 and 2. This resulted in a configuration of variables as shown in table 4.8 and the plot is shown in 4.17. The calculated difference for this configuration was 0.0819.

![Figure 4.17: Fitness of level vs average perceived difficulty of level (end phase)](image)

Table 4.8: Variables in fitness function, end phase

<table>
<thead>
<tr>
<th>Variable</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E[G_w]$</td>
<td>-0.8</td>
</tr>
<tr>
<td>$G$</td>
<td>0.9</td>
</tr>
<tr>
<td>$H_{gg}$</td>
<td>-0.18</td>
</tr>
<tr>
<td>$S$</td>
<td>0.94</td>
</tr>
</tbody>
</table>

Table 4.9: Results of PENS in both iterations

<table>
<thead>
<tr>
<th>Iteration</th>
<th>1</th>
<th>2</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intuitive Controls</td>
<td>5.665*</td>
<td>5.28</td>
<td>-0.385</td>
</tr>
<tr>
<td>Presence/Immersion</td>
<td>3.42</td>
<td>2.94</td>
<td>-0.48</td>
</tr>
<tr>
<td>Autonomy</td>
<td>4.08</td>
<td>3.08</td>
<td>-1.00</td>
</tr>
<tr>
<td>Competence</td>
<td>3.75</td>
<td>3.81</td>
<td>+0.06</td>
</tr>
</tbody>
</table>

32
Table 4.9 shows the results of the PENS questionnaires and a comparison between the values received in iteration 1 and 2.

\footnote{In iteration 1, one question from the Intuitive Controls category was missing due to a mistake.}
Here follow the discussion of the results gathered from the pre-study, each iteration and the end phase.

5.1 Pre-study

Initially, a GA was supposed to be found and modified to fit the needs of this thesis, however it was decided that it would be better to create an algorithm from scratch instead. This was since it was easy to find methods for making an algorithm, but hard to find one that fit this particular problem.

Problems

The goals of the pre-study were met with relative ease. However, at the start of the pre-study, spline interpolation was used to create the basic structure of each chunk. This means that a polynomial was found, points were extracted, and a curve was fitted to these points, which acted as the physical base of the chunk. These levels were found to be inappropriate for a platform game since they usually have more of a block structure. An example of this are the levels in Super Mario, which consist mainly of straight platforms that differ in height. Spline interpolation, on the other hand, created short platforms consisting of one or two tiles all differing by one tile in height. This lead to levels where the player constantly had to jump while none of the jumps were challenging in any way. This was believed to result in an interruption of the player flow, since the jumps didn’t really lead anywhere.

When the GA was first implemented other tiles than spikes and air were randomized, but it was found that this was undesirable. For example when randomizing ground tiles the algorithm took longer to finish and most results were esthetically unpleasing, with random small blocks of ground placed in midair. This was believed to limit the creation of variation through the use of the GA and another approach was chosen instead: To only randomize spikes and air, and insert tiles and objects that affect variation after the GA was done.
5.2. Iteration 1

Evaluation
The evaluation of the pre-study was, as written earlier, performed through a meeting with the stakeholder. The meeting and the discussion in it, produced a lot of information on possible tasks for the first iteration and was thus considered a success. As stated by Basili et al. an iterative process allows for timely and essential feedback [4, pp. 6-7]. Regular contact with the stakeholders, in the form of these meetings, allowed this kind of feedback and allowed them to affect the projects direction. Since the stakeholders are to receive the finished product it should meet their requirements.

Tasks for iteration 1
When gathering tasks for iteration 1, a task was found that was chosen to be omitted. The task and the reasons for omitting it is discussed below.

Adding the functionality to be able to steer the direction of the chunks generated. For example: Being able to force the generator into creating a level that only moves the player character in an eastern direction. This was omitted for two reasons. First, since this was viewed as reducing the randomness of the generator. The generator should be able to create acceptable levels with minimum user input, and this would add to the amount of user input required. Second, since the generation of levels will be done offline. Levels with an unwanted direction can easily be discarded and not used in the final game, which is backed by Togelius et al. [26]. Furthermore Whitehead et al. states that it is believed important to engage as much as the players body as possible [28], and allowing the generator to steer the direction of chunks in a single cardinal direction was believed to work against this theory since, for example, a level that only moves the player in a eastern direction would decrease the amount of body parts used. There is a point to this functionality but it was omitted because other tasks were found to be of considerably higher priority. Stakeholders also found that this functionality could be omitted, since one may actually want to create levels going in one direction, for pure level-design purposes. Code-wise it is possible to manually steer the creation of chunks, but there is no algorithmical logic preventing the random generation of chunks to end up in a single cardinal direction.

5.2 Iteration 1
Allowing the algorithm to completely randomize the directions of the chunks meant that north-going chunks could be introduced. Creation of these chunks was found to take a long time. The long creation time was due to the fact that the north-going chunks were not treated separately. This required the algorithm to randomize some sort of staircase or platform structure, which would help the player get to the top of the chunk. In the pre-study however, it was found that allowing the algorithm to randomly introduce ground tiles in the air gave strange results, and this possibility was removed. Along with this, the completely random creation of north-going chunks was made impossible, since platforms could no longer be randomly placed in the air. It’s important to note, however, that when the algorithm was allowed to randomize ground-tiles into the air, the random creation of north-going chunks was possible, but took a very long time. Because of these limitations a separate way to create these were introduced.

With the way the physics work in the game, the players character being three tiles high and jumping four tiles high (assuming there is no spring), the generator would be unable to place a platform directly above another while still allowing the player to reach both, since placing the second platform at a height that would allow the player to reach it from the first platform would at the same time make it impossible to be on the first platform. If we were to insist on solving the problem in this way, i.e. constructing platforms to allow the player to reach the
5.2. Iteration 1

top, either springs on the platforms to enable the player to reach the platforms above, or the creation of a sort of wide platform staircase would need to be utilized. This would work, but it would instead take up a lot of space in the already rather small chunks.

Evaluation

The user tests in iteration 1 were a success and gave a lot of data to work with. The testers played the easiest level first, to ease the subjects into the game, and the hardest level last, to avoid the player failing because of lack of experience.

The first piece of information that was extrapolated from the test was that the controls and precision of jumps was a big source of the games difficulty. Jumps were seen as harder if the landing space was small. A small amount of space between a set of spikes and the next set of spikes could have a big impact on the difficulty, since this would force the player to succeed with jumps that required a lot of precision, which is hard with the sensitive motion controls. Testers reported certain jumps that had small landing spaces as harder. Because of this, the amount of space between sets of spikes was decided to be included into the fitness function, and introduced into the fitness function at the end of the iteration. This improved the result of the fitness calculation, but by a smaller amount than expected. It could be that this has a lesser impact on difficulty than expected or perhaps the choice of using only the smallest landing space, instead of all available, could decrease the amount of information about a levels difficulty available from the variable.

In the test, once players realized that walking on a spike made the character invincible for a short period, some started using this to their advantage by skipping portions of the level by running while the invincibility was active. If the player died, they would be brought back to the nearest checkpoint with maximum life. If the player managed to get past the hard spot, there was a high possibility of a checkpoint being placed there. Even if the player would die afterwards, the goal of the game became to get the checkpoint in order to minimize the penalty of dying rather than avoiding to die, making the game less challenging. Therefore the space between checkpoints could be increased as the difficulty increases, to avoid players taking advantage of this invincibility to skip large portions of the level. However, this was considered a low priority task and in the end omitted because the problem did not persist. When the levels created better represented the desired difficulty chosen, players could get through most problem areas by skill and a correctly used progression where players weren’t exposed to difficulties above their skill level. Furthermore, levels are to be generated offline, allowing for changes in the levels after generation. Because of that, tasks that had small effects on the finished levels had a lower priority than tasks that affect larger portions of a level.

In figure 4.4 it is possible to see that the fitness matches the reported perceived difficulty for the most part, or at least in the sense that a higher reported fitness corresponds to a higher difficulty as perceived by the players. Between the fitness values of 1.880 and 2.367 the incline of the graph drastically decline. This may be because players see hard levels simply as hard. If a player has a hard time getting through a level with a fitness of 1.880, it stands to reason that the same player will have a hard time getting through a level with a fitness of 2.367. Both these levels may then be perceived as “hard”. Making it more difficult for the player to rate the two levels in relation to each other. Therefore it was decided that the test for iteration 2 should contain more than two levels in this fitness span, which hopefully would give more information on this phenomenon.

There seems to be a connection between both time spent in a level and the number of deaths to difficulty. If these might affect perceived difficulty in a way that might contort the results is uncertain. If the player spends a lot of time in a certain level, it cannot be said with
certainty that this is a direct cause of the levels difficulty. As explained by Togelius et al., this is an example of a gameplay related metric that could possibly be used to create a gameplay experience model [29]. The problem, however, is that assumptions about the data needs to be made. One such assumption could be that a longer time spent in a level would increase the difficulty perceived by a player. However, this may not be the case. For example: the level could be a long straight line or include a lot of exploration which would increase the amount of time spent in the level, but not necessarily increase the perceived difficulty. By the same reasoning it is not certain that a higher amount of deaths is because a level is of a higher difficulty. It could be that the level is introduced to the player too early and they lack the skills to complete it. This is an indication of the difficulty of the level in itself, but the goal of the thesis is to find ways to differentiate levels of difficulty from each other. As such, it cannot be used to decide if one level is harder than another. It could possibly be used to decide how difficult a specific level was to a certain player or player group, but that is seen as irrelevant in this thesis.

PENS

The questionnaire used ranked items based on a Likert-scale ranging between 1-7, where a higher number indicated a higher level of agreement with the item. This means that the average on any given item is 3.5. A higher number here could mean that the need for the specific PENS-variable is fulfilled.

To start, a question connected to the intuitive controls-variable was missing in this test. This was unfortunate, since evaluating the controls was a big part of these tests. Because of this, more weight was put on observations that were made during the test. The two remaining questions regarding this variable were included however. These were: "When i wanted to do something in the game, it was easy to remember the corresponding control" and "The game controls are intuitive". For these the participants answered an average of 5.91 and 5.42 respectively, which seems to indicate that the testers found the controls to be both easy to learn and easy to remember. However, through questioning we also found that they found the game to be hard to control and that high precision jumps were hard to achieve. This may indicate that, while the controls are easy to learn, they are in need of improvement since, while being easy to learn, they are still hard to master. It is important to note that the effect of the controls has been incorporated into the fitness function, since the perceived difficulties reported by the testers was reported when using the controls. This could mean that if changes are made to the controls, the levels created by the generator may start to differ from the difficulties that it is supposed to create. This assumes that the controls have an impact on the perceived difficulty by players, which seems likely considering the amount of comments received about the controls during the user tests.

Presence and immersion got an average score of 3.42 which, while being below average, is quite high considering the sparse game world and lack of elements in the tested levels, which consisted only of ground and spikes. The autonomy of the game was rated an average 4.08, which is above the average, meaning that the players felt that they could make some choices, but that the levels felt mostly linear. In future iterations things such as coins were added that might impact this in a positive way. Lastly the average score for competence was 3.75, which is slightly above the average. This seems to say that the level of competence felt by testers may stem from something else than the control schema, or at least not only from the control schema.

1A player group would in this case be a type of player having problems with specific mechanics, e.g. performing timed jumps.
Tasks for iteration 2

Iteration 2 included a task to ensure that spikes cannot be placed directly under pits. The reason for this is that when a player is killed by something they can’t possibly avoid, except by sheer luck or trial and error, it may be seen as unfair, something that could be a source of frustration instead of fun. This is in accordance with the learn-by-dying design, as described by Ernest Adams. If such unfair elements occur anyway, the player should be warned in some way [3, p. 417]. Since warnings in the form of graphics (e.g. arrows) was not a part of the game, the solution was to remove the spikes instead.

Start of a level should not look like end of the last level played. This is to avoid confusion, since the levels look very alike and it is not very apparent when a player transitions to a new level. Starting a new level where the player needs to change direction in the beginning may be confusing, to the point where the player doesn’t even realize that a new level has been loaded. Though this could be put as a task to be solved in iteration 2, it was not, for a few reasons. First, since the levels were supposed to be created offline it would allow the game creators to choose the levels that matched their expectations, and avoid putting levels with similar end/start-combinations to be played in progression. Second, all features hadn’t yet been implemented into the generator, e.g. decorations such as trees. It was speculated that trees could move focus from some of the similarities that may otherwise be present in a chunk, and therefore help the player realize that a new level had started. Lastly, there was no implementation to compare finished levels with each other, i.e. to read an actual level from a file and compare the individual chunks. This would be necessary if the levels were to be created on two separate occasions.

In the tests, dead ends were found to be bad. They give the impression that the world is bigger than it is, and that there are alternative routes to get to the destination. This was not the case, however, and once players realized this the world was perceived as more linear. Therefore, one task suggested for iteration 2 was to make sure that dead ends (not counting the trenches one can fall into, as these can be seen as traps) should always contain something, be it coins, tokens or power-ups. This task was not included in iteration 2 however, as other tasks were seen as having a higher priority. Another reason was that the game creators could open the level in the level editor Tiled and, if desired, put tokens or coins in said dead ends, with little trouble.

Lastly, there was a suggestion to include multiple paths in a level, giving the player a choice of which way to travel. While this was considered a good idea, it might add to the confusion of traveling through the level. This could be solved by including graphics to help the player see which ways are available, e.g. signs with arrows pointing the way. Inserting new graphics were not a part of this thesis project, however. It would also require a lot of work to accomplish this, and since the goal of the thesis is to be able to correctly represent different difficulties in created levels this task was considered to have low priority and ultimately not implemented at all.

5.3 Iteration 2

At this stage, the width of landing spaces were introduced into the fitness calculations. The solution was first thought to be to measure the average width of the landing spaces between the spike-groups in a chunk. Upon testing this variable, however, it was clear that there could be areas with few spikes, followed by areas with many spikes. This would drive up the average width of the landing spaces in the specific chunk, and make it hard to see the problem areas. Because of this, the variable was changed to instead find the smallest landing space in a chunk. This way, if there is a spike-group with a landing space of two (the width of
the character), it will drive up the difficulty significantly. The problem that could arise here was that if the spike group was placed close to a wall, e.g. in a dead end, it could still be counted towards the smallest landing space of that chunk.

The decrease in difficulty as the landing space increases is not linear. A small landing space, e.g. with a width of two, is seen as very hard, while a landing space of e.g. width five or six is seen as much easier. Because of this, a logarithmic scale was used in the tests. The logarithmic formula was inverted so that a larger landing space had a smaller impact on the fitness function. This seemed to result in good values, making sure that smaller landing spaces had a big impact, while larger smallest landing spaces had a much lower impact. One could also argue that an exponential scale could be used. This was tested, but didn’t work very well ($e^x$ becomes a very big number, very fast).

The new variable improved the result with a small amount. There was still a jump of fitness between the third and fourth levels, which may somewhat blur the lines between the difficulties of the levels in that area. Perhaps there should be more focused testing in this fitness area, to find more clear lines between the different difficulties. The only real difference between, e.g. difficulties 3 and 4, at least after the difficulty thresholds were introduced, were how the spikes were placed.

To increase the variation present in the generated levels decorations in form of trees was added to a background and a foreground decorative layer, which made the levels look much less barren. Adding trees could impact the difficulty since it is possible for elements such as spikes, springs and even the character to be obscured behind trees that are placed in the foreground layer. It could also end up cluttering the level with more things, making it hard for the player to keep track of important elements in the game. A way to mitigate the effect of this was to restrict the generator to not put trees in front of things that were small enough to be completely hidden behind them, mainly spikes and springs.

Another way that variation was introduced into the generator was that different objects available for use in the game world were restricted to only appear after certain difficulties. These objects and the difficulties at which they start to appear can be seen in table 4.4. While this impacts the variation and can be used to create a more meaningful progression between individual chunks, it could be seen as restricting the randomized generation. The problem with completely unrestricted generation of objects, however, is that they may be introduced too soon. If a level is created with the goal of being the first level to be played, there are most likely many elements that shouldn’t be present at all, which is accounted for with this solution.

The results from figures 4.14 and 4.15 points in the same direction as in iteration 1. Higher perceived difficulties generally correlates with a longer time spent in a level and more deaths. There isn’t much to say other than what was said in the discussion of iteration 1. It is important to note that all the levels in the tests, except for one in iteration 1, were of equal length. The increase in difficulty with a higher completion time could be because testers can get stuck on particular spots in the level. It could also be because harder levels take longer to get through. What wasn’t done in iteration 1 however was measuring the amount of coins picked up by the testers. This was done in iteration 2 and the result can be seen in figure 4.16. This wasn’t done in iteration 1 because the generator didn’t contain that many coins during that time.

By iteration 2, crates, enemies and spear pendulums were included, all of which provide coins available for pick up. Pedersen et al. makes the argument that more coins picked up in a level signifies lower difficulty, since the player does not have time to collect coins if they have a problem with a level [21]. In the game used in this thesis, however, there are no time-
5.3. Iteration 2

There is a timer, but it simply counts how long the player has spent in a level. In figure 4.16 we can see that, in general, picking up more coins means the level was perceived as harder. Considering that a time limit is not a factor in the game used in the thesis this doesn’t necessarily contradict the argument made by Pedersen et al.

Executing the matlab-program with more than three variables took a very long time, which is an effect of the curse of dimensionality [6]. In iteration 2, each variable had two-hundred different configurations, between -1.0 and 1.0, with steps of 0.01. Going through all configurations for x variables means going through 200^x configurations. Three variables would result in 200^3 = 8,000,000 configurations, and four variables would result in 1,600,000,000 configurations. Making the precision higher would further increase this number, and was not seen as an option. Because of this, there may be configurations that are better than the one found. There could also be specific toolboxes or programs made to solve this problem that could provide more exact answers, but these weren’t used. As mentioned earlier, however, the goal was not to find the best solution, but a good solution. Finding the best solution could, even with specialized tools, take a very long time. There is also no real telling if the “best solution” on paper would correspond to a fitness function that would help the algorithm to produce good levels.

PENS

The PENS questionnaire provided some interesting results, these are presented in table 4.7. The intuitive controls metric got an average score of 5.28, this time with all questions included, which is the highest value of all metrics by far. Once again this calls to question how the controls can be seen as intuitive while at the same time being, by the testers point of view, a big reason for the difficulty of the game. One could argue that the testers makes a distinction between preforming the actions available with the controls and their ability to master them. For example: Perhaps it is seen as easy to jump, generating movement in the middle of the screen, but hard to jump the way you want to and precisely control where you land and how far you jump. The score could be an indication that the controls are not necessarily bad, just imprecise, meaning that they need fine-tuning.

Presence/Immersion got an average score of 2.94. The low score is not surprising considering that the game is still in development at the time of writing and as such does not for example include any form of story or characters other than the players avatar.

Competence got a score of 3.81, which seem to say that the players do not find themselves to be particularly competent at the game. It should be noted that, much like in iteration 1, most players expressed a difficulty playing the game because of not being used to the controls. The majority of testers had never even seen the game before, which may also impact the competence score, i.e. the items are rated lower because they don’t feel like they’ve learned how the game works during the test-session.

Autonomy got a score of 3.08. The levels were made with one start and one goal in mind, and only one definite way to get there. Sometimes the player could have a choice between jumping or using a spring, or going over or under a platform. This doesn’t really provide different routes, however, and the lack of these different routes could explain the low score.

Tasks for the end phase of the project

In iteration 1, recall that spikes were considered unfair if placed underneath a pit, i.e. where the player would not see them before jumping down, in accordance with Ernest Adams game design principles [3]. This was also shown during testing, where they indeed were a source of frustration for players. This was fixed in iteration 2, however spikes could still be placed.
very close to pits, resulting in deaths or damage in much the same way as before, just not as frequently. As with spikes, randomly placed spear pendulums were found to be unfair if placed directly under a pit. These problems were fixed in a task for the end phase, i.e. ensure spear pendulums are not placed directly underneath a pit, and remove spikes that are adjacent to the "landing zone" beneath a pit.

There could be other solutions to the problem, rather than removing the items. The pits could be designed in such a way that the obstacles below could always be seen and therefore not be a surprise to the player. This would most likely impose more design restrictions on the chunks, since pits could no longer be deeper than the player could see, or require the game settings to be changed and for the camera to show more of the game, which may be an unwanted consequence.

Many testers were confused the first time they found a pit. They were unsure if they could/should jump down into them, or if doing so would kill the character, which could well be the case in other platform games. This affected the players flow in a negative way and therefore it was decided that the visual cues, indicating that the player could indeed fall down, should be more clear. A suggestion was to create a line of coins going into the pit that would signify a reward for falling down and, hopefully, remove any doubts of where the player should be heading. There could be more problems related to this however, since there were no further testing conducted after iteration 2.

5.4 End phase

The results of the final configuration is shown in figure 4.17. Even though the difference calculated was higher than the previous result, creating levels using this setup gave results that much better matched expectations. Therefore, this setup was used as the final configuration. It is interesting that even with a higher difference, the configuration still resulted in better generation of maps, when the theory was that a lower difference would result in better levels. The cause of this is believed to be the presence of outliers in the test, which then results in visible anomalies in the test data. It may also be that the connection between fitness and difficulty is non-linear, which is discussed later in section 5.5.

The configuration that resulted from the tests in the end of iteration 2 did not give the fitness function the ability to generate levels to the extent that was expected. Changes made to the configuration in previous phases of the project had generally made the algorithm perform better, but using the results from iteration 2 resulted in the creation of strange levels. Because of this, there was a change in the method to adjust the variables in the fitness function. Instead of using only the results from the previous iteration to create levels, adjustments were instead made using the results from both iteration 1 and 2. This problem may have arisen due to the small amount of testers, which will increase the impact that single testers have on the results, seeing as the changes to the configuration were made singularly using the results from the user tests. This means that if even a small amount of users thought that some levels were much harder, or much easier, than the rest, this may have a bigger impact on the result than it would have if there were more testers involved.

It should be noted that the iterative process made the task of changing the variables again easier, since this had been done many times before. The iterative process thus allowed for a change in the plan on short notice which is in accordance with the findings by Osorio et al. [20] p. 455] The risk of ending the project with a bad configuration was mitigated because of this, as well as the fact that, in the worst case, the configuration from the previous iteration could be used if the current one turned out to be worse.
Considering that inspection of the generated levels by the developers showed the configuration from iteration 2 to be flawed, perhaps the fitness function should have been evaluated by more means than the user tests. Perhaps, to increase the amount of sources of data, the focus of the stakeholder meetings and the developer inspection of produced levels should have been perceived difficulty.

**PENS**

The results of the PENS questionnaire comparison, seen in table 4.9 look bad. With some new elements added between the iterations, e.g. trees and enemies, it was believed that some metrics would be positively affected, e.g. increased immersion because of the presence of trees. This was since the version tested in iteration 1 was very barren, containing only spikes and ground. Therefore the results are surprising, unless the difficulty in itself affected the players perception of both Presence/Immersion, Autonomy and Competence. It’s important to note, however, that there will always be a natural variation in statistical surveys and that the differences between the tests in the iterations may be caused by this variation. In the results, this is shown as the confidence interval. The discussions below assume that the differences in the results are not caused by the natural variation, however this may still be the case.

The controls of the game did not change at all between the iterations, so the change of -0.385 in the intuitive controls section seems unreliable. This could be since, as mentioned before, a question in the variable was missing in the first iteration. This could in turn skew the results, making the gap between the first and second iteration seem bigger (or smaller) than it should be.

The players perception of Presence/Immersion could change due to the new elements added to the game between the iterations (trees, crates, enemies and coins). Instinctively new items should add to the immersion of a level. Therefore this result is hard to explain, since there was reported change of -0.48. However when asking a player who tested both versions if the added elements were welcome, the tester responded that the lack of elements in the first version had gone by mostly unnoticed. This could be since, in a test environment, the tester is mainly focused on performing well during game play, not paying attention to detail in the game world.

The autonomy of the game decreased by 1.0 in iteration 2. The autonomy section measures such things as if the game let the player do interesting things and if the player experience freedom in playing the game. While more items, such as crates and enemies, should add to what the player can do in the game. It could be that the specific levels used in the second test felt more linear to players. It was reported in the user test of iteration 1 that one player missed out on something they thought was a alternative path in a level. Something that was not found in iteration 2. It could be argued that things of this nature might be the reason for the autonomy score being lower after iteration 2.

The competence section went up by 0.06 points. However the increase is probably too small to draw any conclusions from. One could perhaps argue that nothing changed between the iterations that would change the competence measure felt by the players.

The two user tests were carried out with twelve participants in each iteration. This might have been too few to get accurate results in the PENS questionnaire. Some of the results in the PENS questionnaire seem unreliable, mainly the intuitive controls section, since this reported a change when no adjustments had been done to the controls. Therefore the responses to the questionnaires may be too small a sample to draw any conclusions with confidence.
By this logic the small sample size might have an effect on the fitness function as well, and further tests with a larger testing crowd should be conducted to verify the results above. This is unfortunately out of scope for this thesis.

5.5 Method

Improving upon the algorithm using an iterative method together with the EDPCG framework [29] defined by Yannakakis and Togelius worked well. For the thesis the goal was to answer the research question while at the same time going through a software development process that would result in a product ready for delivery. In order to give the customer a sufficiently good product the iterative process was a good choice. It allowed for regularly conducted user tests that helped acquire subjective and gameplay-related data. The subjective data was used to update the player experience model, while the gameplay-related data was used to research if data such as the time spent in a specific level would correspond to the perceived difficulty of that level. The results show that the gameplay-related data does not correspond well with the perceived difficulty, in most cases. As stated by Togelius and Yannakakis, using gameplay-related data could result in a faulty model, since it will be used as a basis for assumptions [29, p. 7]. This might be the reason that the gameplay-related data does not correspond well with the perceived difficulty. Objective data, which may be obtained by using e.g. sensors, were not considered as a way of measuring user experience in this thesis. The developers did not have the sensors required to consider this possibility.

Basili et al. states that the iterative development model should allow for invaluable user feedback during the course of a project, which was indeed the case in this thesis work [4]. Users provided data that enabled the GA to create levels with controllable difficulty. The user tests focused on measuring difficulty. If this is good or bad depends on from what point of view it is discussed. The goal of the thesis is to be able to control the difficulty of the levels created. And for this goal, the iterative process and the user tests was invaluable, seeing as the goal with the GA is to create levels that corresponds to users perceived difficulty. But at the same time the tests were designed to only measure difficulty, not the features in the game. This means that while the levels may have corresponding difficulty, it is not necessarily an enjoyable experience. Features to enhance the enjoyable experience could have been extracted from user tests, but because of the tests design, this was omitted. The measurement of enjoyment would most likely need to be a whole other type of test, specifically designed for this purpose. Such features were instead extrapolated from meetings with stakeholders, which could lead to a biased view of what features should be implemented into the final product.

The results of the PENS questionnaires were used mostly as a way to measure the quality of the product, which can be a good thing when designing or developing a game to see which parts of the current product may need to be changed in order for players to enjoy it [23]. However, it was not much help in evaluating the fitness function and perceived difficulty, which was the goal of the thesis and the main reason for carrying out the tests in the first place. The values received from the questionnaires may be used as a starting point when continuing the development of the game itself.

Considering that the goal was to create an algorithm that could correctly represent difficulty in the levels generated the user tests was invaluable. If the difficulty would have instead been measured by the writers of the thesis themselves and the customer, it is believed that the result would be much worse. Because of two reasons. First, the amount of data on perceived difficulty would come from only four people in total, which wouldn’t be enough, especially considering the second reason, all involved have earlier or during the work of the thesis spent a lot of time with the game. The reported difficulties would thus have been affected by a large amount of time playing the game. Simply put: what was considered an easy
level by customers and the writers of this thesis, was not considered easy by people who had never played the game before.

While the user tests was a good source of data, one thing would probably be changed if the work had been done again: The amount of testers should be increased. In this thesis twelve people were used per iteration. This was mainly an issue of time, but also because that generator did not need to reach perfection. A compromise between accuracy of the gathered data and the amount of functionality implemented in the finished product had to be made. The choice of preferring functionality is believed to be the correct one. Mainly because the generation of levels will be performed offline. Thus if a level generated is felt to not coincide with the desired difficulty, another level can simply be generated, or changes made to the generated level as per the principles of offline generation [26]. If there was more time available however, it would be recommended to perform more user tests in order to perfect the fitness function.

Since the users were asked to rate the difficulty of the level, they needed something to compare with. The rating was often made either in comparison to the first level played, i.e. if the first level was considered to be a 2, and another one was considered easier, it was rated as a 1. This is since the testers never played the game before, and had nothing else to compare it to. A suggestion was made by a tester to show some examples of levels and their difficulty before testing. The problem then was how to rank the difficulty of the levels shown, since this is what we were supposed to measure, and the tester would probably forget what the comparison-levels looked while in the middle of the test.

Overall the results from the user tests brought a lot to work with, and the difficulties seemed to correlate quite well with the defined difficulties of the levels. Since the testers weren’t explicitly told how the fitness-function worked, they rated difficulty by looking at more things than just the placement of spikes. One tester rated a level to be a 1 because it was perceived as short and another level a 9, because it was perceived as being long. In some levels, the ratings from some testers were higher because there were features present, like moving platforms. As mentioned above however, increasing the amount of testers would help eliminate these outliers and most likely give more accurate results.

A potential flaw in the method was that the difficulty of the levels were discretized into linear steps, i.e. the assumption that there would be a linear increase in difficulty, and that the steps between the difficulties would have the same distances. This may have affected the results negatively. In the tests it is possible to see that, while a harder (according to the GA) level has a higher perceived difficulty, it does not fully correspond to a linear increase in fitness. This may mean that the relationship between difficulty and fitness is not linear.

The evaluation of the iterations with stakeholders focused on developing the game rather than the GA and the fitness function. This might affect the output of the GA negatively, by dividing the attention of all involved to things not directly related to the goal of the thesis. Development of the game itself is important, but it was not the focus of the thesis. Perhaps this should have been a clearer delimitation at the start of the project.

5.6 Genetic algorithm

The choice to use a GA worked out relatively well. However, a GA true to its concept should probably allow for as much randomization as possible. Some restrictions to this was implemented in the GA, for example the mutation step only allowed for the mutation of some tiles into spikes and spikes into air. The ability to mutate a random position into ground was not allowed since this almost always created levels that were aesthetically unpleasing or impos-
sible to pass. By removing the ability to mutate positions into ground tiles, a lot of available solutions were removed from the solution space. To allow this ability, there would be a need to change how the game physics work, e.g. by making the character smaller or changing the jump height. One could also experiment with having different mutation rates for different tiles, i.e. that air tiles have a chance to turn into ground tiles, but the chances are smaller since this could lead to the level not being feasible. This would still not be fully true to the concept of a GA though. Allowing air tiles to be turned into ground tiles is desirable, since this allows the creation of more random platforms. However, there was not enough time to find a good solution in this project.

At the beginning of the project there was doubts that the feasibility algorithm would be fast enough to be used for each iteration of the GA, however this turned out not be a problem at all. Another step in the GA meant a longer execution time. But relative to the execution time of the rest of the algorithm and the increase in the quality of the output it provided, the added execution time was considered negligible.

The theory was to build a GA that based the fitness on the difficulty of a level, by using the difference of the actual difficulty and the desired difficulty, proposed by Hector Adrian and Ana Luisa [8]. This presented two problems. How to know the actual difficulty of a generated level, and how to describe this in the mathematical function used in the fitness function. The user tests were conducted in order to gather information on the difficulty perceived by players, and used as the actual difficulty. The work of adjusting the variables of the function was the attempt to describe the mathematical formula for the difficulty. When these problems were solved, using the difference between the actual difficulty and the desired difficulty turned out to work well.

The generated levels at the end of the project seem to correspond to desired difficulty, however some uncertainties remain. The correlation coefficients gathered by Pedersen et al. [21] differ vastly from the result of the final configuration. It may be that the effects on difficulty by different variables may vary between games, depending on what in the game affects difficulty. The assumption that spikes are comparable to pits in Super Mario may be false, and either the coefficients found by Pedersen et al. or the ones produced in this thesis may be incorrect, even if they produce good results.

Because of events that were out of the developers hands, the evaluation in the end was only done by the developers themselves. The stakeholders did not affect the final stage of the development process, meaning that it’s uncertain whether the final product creates levels that fully corresponds with the stakeholders expectations. While the algorithm makes levels with a controllable difficulty, and the results can be used to answer the research question, it is impossible to know how this situation may have affected the end results.

5.7 Research Question

This section presents the aspects of the thesis work that will try to answer the question introduced in the beginning of this thesis:

How can a genetic algorithm be implemented to create levels with controllable difficulty?

The use of a method similar to the EDPCG framework turned out to be a good way of developing the GA. The main idea is to use player experience and change the fitness function according to the player experience. This in tandem with the iterative development process allowed for the GA to be changed between iterations, using the player experience gained during the user tests.
The fitness function used in this thesis calculate the fitness of a level by using its difficulty, by comparing the desired difficulty to the actual difficulty as proposed by Hector Adrian and Ana Luisa [8]. It turns out to be a good way of creating levels with controllable difficulty, however with some limitations, such as the need to accurately describe the mathematical formula for difficulty and to accurately decide the actual difficulty of levels.

Since the correct measurement of difficulty is important, as much time as possible should be spent on extrapolating elements of difficulty from the game. This is because different games may include different elements of difficulty. The better the difficulty of levels is described, the better the produced levels will be in regards to the desired difficulty, since as many elements affecting difficulty as possible is included in the search for a level. It may be hard, however, to say exactly how many elements there are that affect the difficulty, and maybe even impossible to find all of them. For example Jordan Fisher, one of the developers of Cloudberry Kingdom [10] claims that their algorithm uses thousands of variables to control difficulty, while the algorithm in this thesis work uses only four variables.

One could also use other ways to create the levels, e.g. by defining chunks much in the same way as Tim Ziegenbein [30]. The level generator could be used to create huge pools of specific chunk types, since the main problem with the previous implementation was that players encountered the exact same chunks in a short amount of play time. The level generator could be used to generate the structure of the maps, but pre-defined chunks, also generated by the level generator, could be used instead. This would allow for a form of online generation in the game, as opposed to purely offline. Since the GA was created only with offline generation in mind however, this was not used.

With a lot of players testing the game, there could always be the odd chunk that was repeated. Even with a pool of e.g. a thousand generated chunks for a specific chunk type, users may still encounter the same chunk again. With the way random generation works, this could happen with the final implementation as well. However, checking the layout and look of the levels to be used when the game is finally released could diminish the risk of this happening, while checking up to several thousand chunks and looking for similarities would not be feasible. One can ask how good of a memory a player really has, and whether they would notice any repetition at all when the pool of available chunks is big.

To measure the GA’s ability to calculate the difficulty of created levels, user tests, such as the ones conducted in this thesis is recommended. Difficulty is to some extent subjective and in order to be able to create appropriately difficult levels for players, as much information on perceived difficulty of levels as possible is needed. In this thesis the tests were conducted with twelve people per test, but it is recommended to increase the amount of testers. This number should be increased since the number of elements in a level that affects difficulty can be a lot and testers perception of difficulty differs from person to person. Simply put, there is a lot of uncertainties when measuring difficulty and therefore more testers should lead to a result closer to the truth. The correct amount of testers is hard to say, and it cannot be stated with certainty what the right amount would be. There is however theory on software testing that would be more suitable to explain the needed number of participants and the methods used in testing.

5.8 The work in a wider context

The product created in this thesis is to be used in a game promoting movement and exercise. However, one could argue that the creation of this content would cause players to remain indoors looking at a computer monitor, instead of being outside, getting exercise. It is important to note that the game is developed with the purpose of helping people with possible
physical disabilities, or people who cannot get exercise to the same extent as people without these disabilities, i.e. people who may not have a lot of choice in ways of getting exercise.

If PCG were to be developed to such an extent that all content in games could be automatically generated, this might have a negative effect on the amount of available jobs in the game industry. While the PCG used in this thesis is far from being at that level, this thesis still adds to the research available in the field and thus furthers the development, making it a part in this process. However, as mentioned earlier, PCG would allow smaller game developers to have a stronger position on the gaming market, since developers can focus more on developing features of the game.
Conclusions

In this thesis, a genetic algorithm was implemented with the goal of procedurally generating levels for a 2D platform game, and to be able to control the difficulty of the generated levels. The purpose of this was to investigate how a genetic algorithm can be used to generate this type of content. The game was a motion-based game, which meant that it was controlled using the body.

It has been shown that the use of a genetic algorithm for this purpose is feasible. However there are some things that need to be considered when choosing this option.

Using a genetic algorithm is generally a slow approach to generating levels, simply because of the (generally) large search space. Because of this, the results of this thesis may not be applicable when considering using a GA with the purpose of online generation of content.

It is important to be sure that a genetic algorithm fits the purpose of the content to be generated. In the game worked upon in the thesis, there were design constraints, e.g. that certain game elements should not be generated in mid-air. This resulted in the genetic algorithm not being able to fully mutate every part of the level, if it wanted to. The end result is a genetic algorithm that, while levels can be generated well, may not be used to it’s full potential. However, in a game where the mutation of every position in a level is seen as correct, a genetic algorithm will be able to fully use the search space, which may give results that are feasible but that a human designer may not think of.

The fitness function needs to be well developed and as correctly as possible correspond to the mathematical formula of difficulty, to ensure that the created levels match the desired difficulty. This is done by investigating the game to find as many elements as possible that affect its difficulty. To ascertain that the elements found do in fact affect difficulty, user tests is a good way of comparing the algorithms output to perceived difficulty by users. In this thesis, the user tests were conducted with twelve people per test. This is possibly sensitive to outliers, particularly since perceived difficulty is subjective data. As such, the amount of testers used in future work is recommended to be higher.
An iterative development methodology is well suited as it allows for regular user tests being carried out and the output of these to affect the development of the fitness function. It works especially well if the mathematical formula for difficulty is underdeveloped, as it also allows for the inclusion of newly found variables in each iteration. It is recommended to use an established framework such as the EDPCG, since it gives good insight as to how player feedback may be used to develop the GA, and how the process should be conducted.

6.1 Future work

If online generation is desirable, this work could be combined with Tim Ziegenbeins design idea, as mentioned in the discussion. This would mean the creation of a pool of chunks for every type of chunk, i.e. with all the different combinations of exit/entry points.

A problem could arise when trying to match completely randomly generated chunks, as e.g. heights between the chunks could differ. This could be solved by putting restraints on the generation process e.g. defining that the entries and exits of chunks that starts in the west and exits to the east always should be of the same height. Chunks requiring pits would need to fit with the connected chunk as well. An example is that a pit in a south-going chunk would need to be connected, in some way, with an entry in the ceiling of the next chunk. These need to match, even with the random generation, and this could be added as a design restriction.

Since there was no time to test this idea in the thesis, it is not known whether this is a good idea or not, seeing as multiple design restrictions in this way may make the game seem less interesting and decrease the variation present.
Bibliography


Appendix - Matlab code
Example Matlab-code for variable adjustment step:

```matlab
function diff = calculate_integral()

matrix = [
    2 0.246047 0.721348;
    2 0.000000 0.197052;
    2 0.333333 0.558111;
    3 0.57835016 0.72134752;
    2 0.000000 0.72134752;
    2 0.666667 0.72134752;
    3 1.0 0.62131097;
    1 0.999999 1.442498;
    3 0.999999 1.442498;
    2 0.626667 0.91012363;
    4 1.24 0.012163 0.72134752;
    4 1.24 0.333333 0.14424989;
    5 1.24 0.000000 0.442498;
    2 1.24 0.666667 1.442498;
    2 0.000000 0.1442498;
    7 0.23023451 0.666667;
    7 0.23023451 1.442498;
    7 0.000000 0.14424988;];

best_corr = 0;  % Initial best correlation
best_number_of_gaps_corr = 500;  % Initial best number of gaps
best_avg_width_of_gaps_corr = 400;  % Initial best average width of gaps
best_bg_corr = 400;  % Initial best background correlation

x = [2.752 3.500 4.555 5.625 5.675 6.333];
normalized = (x - min(x)) / (max(x) - min(x));

for number_of_gaps_corr = 0:10:300
    for bg_corr = 0.01:0.01:1.0
        tmp = [number_of_gaps_corr avg_width_of_gaps_corr bg_corr smallest_landing_54
Appendix - PENS Questionnaire
Reflect on your play experiences and rate your agreement with the following statements:

*Obligatorisk

1. Learning the game controls was easy. *
   *Markera endast en oval.*
   
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strongly Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. I feel competent at the game. *
   *Markera endast en oval.*
   
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strongly Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. When moving through the game world I feel as if I am actually there. *
   *Markera endast en oval.*
   
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strongly Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. When I accomplished something in the game I experienced genuine pride. *
   *Markera endast en oval.*
   
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strongly Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. I am not impacted emotionally by events in the game. *
   *Markera endast en oval.*
   
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. I had reactions to events and characters in the game as if they were real. *
   *Markera endast en oval.*
   
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strongly Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

https://docs.google.com/a/student.liu.se/forms/d/1Q2RZDIs302UqeY2qGzay6g7dy6fmbAgSf5vwsC0rHS7aoQ/edit 1/3
Reflect on your play experiences and rate your agreement with the following statements:

7. When playing the game, I feel transported to another time and place. *
   *Markera endast en oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8. My ability to play the game is well matched with the game's challenges. *
   *Markera endast en oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9. I experience feelings as deeply in the game as I have in real life. *
   *Markera endast en oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10. When I wanted to do something in the game, it was easy to remember the corresponding control. *
    *Markera endast en oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11. Exploring the game world feels like taking an actual trip to a new place. *
    *Markera endast en oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12. The game controls are intuitive. *
    *Markera endast en oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

13. The game provides me with interesting options and choices. *
    *Markera endast en oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
14. **The game lets you do interesting things.**  
   *Markera endast en oval.*

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
   1 | 2 | 3 | 4 | 5 | 6 | 7

   Do Not Agree  | Strongly Agree

15. **I feel very capable and effective when playing.**  
   *Markera endast en oval.*

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
   1 | 2 | 3 | 4 | 5 | 6 | 7

   Do Not Agree  | Strongly Agree

16. **When playing the game I feel as if I was part of the story.**  
   *Markera endast en oval.*

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
   1 | 2 | 3 | 4 | 5 | 6 | 7

   Do Not Agree  | Strongly Agree

17. **The game was emotionally engaging.**  
   *Markera endast en oval.*

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
   1 | 2 | 3 | 4 | 5 | 6 | 7

   Do Not Agree  | Strongly Agree

18. **I experienced a lot of freedom in the game.**  
   *Markera endast en oval.*

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
   1 | 2 | 3 | 4 | 5 | 6 | 7

   Do Not Agree  | Strongly Agree