



Juri Larch

# Exploring automated game testing

## **Bachelor's Thesis**

to achieve the university degree of

Bachelor of Science

Bachelor's degree programme: Computer Science

submitted to

**Graz University of Technology**

Supervisor

Dipl.-Ing. Dr.techn. BSc Johanna Pirker

Institute of Interactive Systems and Data Science

Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, September 2018

## **Affidavit**

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present bachelor's thesis.

---

Date

---

Signature

# Abstract

Game designers and developers have to rely on expensive user testing to achieve a desired game experience. This thesis explores automated game testing of a parameterized version of the game Line Runner. It takes an existing approach in automatic game testing and optimizes the level generation process to output a larger amount of playable games. A difficulty model is developed in order to estimate the difficulty of the levels obtained by the random generation. The efficiency of the difficulty estimation model is verified by conducting a user test. Additionally, an in-depth analysis of the parameter's influence on the difficulty allows the prediction of parameters to obtain a certain game difficulty. Dynamic difficulty adjustment denotes the concept of adjusting a game's difficulty according to the player's skill. Two approaches to dynamic difficulty adjustment for Line Runner are outlined in this thesis. Both use different approaches in their design and can be realized in endless runners. The thesis discusses the creation of a perfect Line Runner by establishing certain recommendations in the parameter assignment that offer a higher playability of the game.

# Contents

<b>Abstract</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>3</b>
2.1 Flow . . . . .	3
2.2 Dynamic Difficulty Adjustment in Games . . . . .	3
2.3 Physics . . . . .	4
2.4 Regression . . . . .	5
2.4.1 Polynomial Regression . . . . .	5
<b>3 Related Work</b>	<b>7</b>
3.1 Automatic Gametesting . . . . .	7
3.2 Automated Playtesting . . . . .	8
3.3 Dynamic Difficulty Adjustment . . . . .	9
<b>4 Design</b>	<b>10</b>
4.1 Game Description . . . . .	10
4.2 Difficulty . . . . .	12
4.3 Level Generation . . . . .	12
<b>5 Implementation</b>	<b>13</b>
5.1 Improving The Level Generation . . . . .	13
5.1.1 Every Second Flat . . . . .	15
5.1.2 Smart Level Generation . . . . .	15
5.2 WebGL Versions . . . . .	16
5.2.1 WebGL Level Generation And Testing Site . . . . .	16
5.2.2 User Testing Site . . . . .	19
5.2.3 Data Collection And Visualization . . . . .	20

## Contents

<b>6</b>	<b>Analysis</b>	<b>22</b>
6.1	Influence Of Level Generation Enhancements . . . . .	22
6.2	Influence Of Parameters . . . . .	28
6.3	Game Difficulty Variation . . . . .	31
6.4	One Dimensional Sampling . . . . .	32
6.4.1	Speed Variation . . . . .	32
6.4.2	Block Length Variation . . . . .	33
6.4.3	Seed Variation . . . . .	36
6.5	Parameter Prediction Using Regression . . . . .	38
6.6	User Testing . . . . .	39
6.6.1	Level Selection . . . . .	40
6.6.2	Results Of User Testing . . . . .	40
<b>7</b>	<b>Discussion</b>	<b>44</b>
7.1	Ideal Jump Distance . . . . .	44
7.2	Dynamic Difficulty Adjustment . . . . .	44
7.2.1	Single Parameter Adjustment . . . . .	45
7.2.2	Sequence Of Different Level Types . . . . .	45
7.3	The Perfect Game . . . . .	45
<b>8</b>	<b>Future Work</b>	<b>47</b>
8.1	AI Improvements . . . . .	47
8.2	Game Experience . . . . .	47
8.3	Difficulty Calculation . . . . .	48
8.4	User Skills Based Dynamic Difficulty Adjustment . . . . .	48
<b>9</b>	<b>Conclusion</b>	<b>49</b>
	<b>Bibliography</b>	<b>52</b>

# List of Figures

2.1	Flow channel . . . . .	3
2.2	Jumping distances of a player . . . . .	4
3.1	Perfect jump position . . . . .	8
4.1	Screenshot of the game Line Runner . . . . .	11
5.1	$\sigma_d$ probability distribution . . . . .	14
5.2	Game difficulty probability distribution . . . . .	14
5.3	Every second flat . . . . .	15
5.4	Smart level generation . . . . .	16
5.5	Level generation and testing site . . . . .	18
6.1	Influence of level generation enhancements on $d$ probability .	23
6.2	Influence of level generation enhancements on $\sigma_d$ probability	25
6.3	$l_{block}$ dependencies because of smart level generation . . . . .	26
6.4	$h_{obstacle}$ dependencies because of smart level generation . . . .	27
6.5	Influence of $l_{block}$ on $d$ . . . . .	28
6.6	Correlation between $l_{block}$ and $S$ . . . . .	29
6.7	Influence of $f_{jump}$ on $d$ . . . . .	29
6.8	Influence of $f_{jump}$ and $f_g$ on $\sigma_d$ . . . . .	30
6.9	Correlation of $d$ and $S$ . . . . .	31
6.10	Difficulty variability . . . . .	32
6.11	$d$ when sampling $v$ . . . . .	33
6.12	$\sigma_d$ when sampling $v$ . . . . .	34
6.13	$d$ and $\sigma_d$ when sampling $l_{block}$ . . . . .	35
6.14	$d$ when sampling $l_{block}$ . . . . .	35
6.15	The deviation of $d$ and $\sigma_d$ when altering the seed . . . . .	37
6.16	Polynomial regression with a 6 degree polynomial . . . . .	39
6.17	Estimated difficulty to rated comparison . . . . .	41
6.18	Comparison of user score to $d_{metric}$ . . . . .	41

## List of Figures

6.19	Comparison of fun rating, difficulty rating and score . . . . .	42
6.20	Comparison of difficulty rating and $P(\textit{obstacle})$ . . . . .	42
6.21	User testing table . . . . .	43

# 1 Introduction

The task of tuning a game experience in such a way that players feel challenged enough is a tedious, yet crucial element of game design. Game developers aren't suited for this task, because they know every possibility to beat the game and are therefore experts in playing it. Often the task of fine-tuning the game experience encompasses high costs and a lot of hassle by undertaking a lot of user tests. This process slows down the development process drastically. Therefore, this thesis expands the method of automatic user testing, that can eventually ease the arduous process of user testing, by demonstrating how to analyse the data of automated playtesting and use it to build a game.

The work in (Steurer, 2018) is used as a base implementation for this thesis. It uses Monte Carlo simulation and player score analysis to estimate the difficulty of a level and does an extensive analysis of the parameter space of the game Line Runner.

Isaksen, Gopstein, and Nealen pursued another approach in analysing the game Flappy Bird with a similar method. They verified their model's correctness by conducting a user study, showing the capability of automated play testing (Isaksen, Gopstein, & Nealen, 2015).

In order to verify the difficulties calculated using this work's methods a user study is conducted. In contrast to the user study of (Isaksen, Gopstein, & Nealen, 2015), which only compared pairs of games with one altered parameter, this work's user study attempts to rate and compare games at an absolute scale.

The Level Generator of (Steurer, 2018) generates about 78.8% levels that aren't playable at all. This thesis optimizes the random level generation process and leads to a spike of playable levels that are generated.

In addition to the parameter analysis of (Steurer, 2018), this thesis performs a more sophisticated analysis and shows the influence of parameters in detail using one dimensional sampling. Moreover, it explores parameter prediction using regression and validates its results. On top of the parameter



## 1 Introduction

prediction, this thesis demonstrates two approaches for realizing dynamic difficulty adjustment (DDA) in endless runner games. Both approaches are different from the procedures used in (Hunicke, 2005), (Hunicke & Chapman, 2004) and (Tan, Tan, & Tay, 2011). The approached solution for DDA in this thesis is quite simple, but requires an exact parameter analysis of the game and its parameters.

A WebGL version of the game Liner Runner is developed that allows the game to function cross-platform. A web browser with the ability to render WebGL is the only requirement to run the game.

This thesis is organized into the following chapters. Chapter 2 discusses the theoretical and physical background. Chapter 3 describes the foundation of automated playtesting and dynamic difficulty adjustment from other papers. Chapter 4 talks about the underlying design. Chapter 5 explains the implementation of the Level Generator improvements and the process of building a WebGL version of the game. Chapter 6 analyses the individual parameters of the game and the result of the user testing. Chapter 7 discusses two approaches to dynamic difficulty adjustment. Chapter 8 provides a prospect for future work. Chapter 9 contains the conclusion and discusses the advantages of automatic play testing for game designers.

## 2 Background

### 2.1 Flow

The flow model by M. Csikszentmihalyi illustrates how to achieve the perfect game experience. The goal is to keep the player inside the flow channel. Figure 2.1 shows the flow channel as the perfect relation between player skill and game difficulty. As soon as the player leaves the flow channel, the game becomes either too easy or too hard for the user to play (Hunicke & Chapman, 2004). This will result in frustration or boredom and an overall bad game experience for the player. The papers (M. Csikszentmihalyi & Csikszentmihalyi, 2000) and (M. Csikszentmihalyi, 2009) contain a more detailed description of the flow.

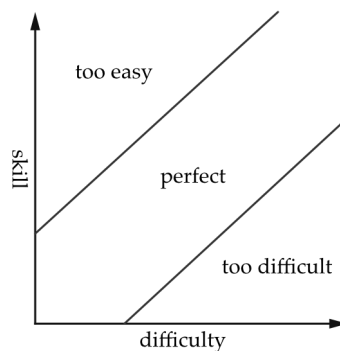


Figure 2.1: This graph shows how the relation between skill and difficulty span the flow channel to achieve the best game experience.

### 2.2 Dynamic Difficulty Adjustment in Games

Dynamic difficulty adjustment (DDA) allows to scale the difficulty of a game according to the player's ability. This enables game developers to

## 2 Background

design a game that is able to satisfy the variety of players that will interact with the game (Tan et al., 2011). DDA has some drawbacks as it takes away control from the designer and puts it in the hands of code. Two approaches to DDA are: (1) annotate game tasks with information about difficulty prior to evaluation and (2) use a combination of data mining and off-line analysis (Hunicke & Chapman, 2004). An approach to DDA proposed in the paper (Hunicke & Chapman, 2004) is to dynamically evaluate the difficulty of obstacles based on user performance.

### 2.3 Physics

This thesis uses two different formulas to calculate the motion of a projectile. Figure 2.2 shows how Equation 2.1 and Equation 2.2 relate to the trajectory of a projectile.

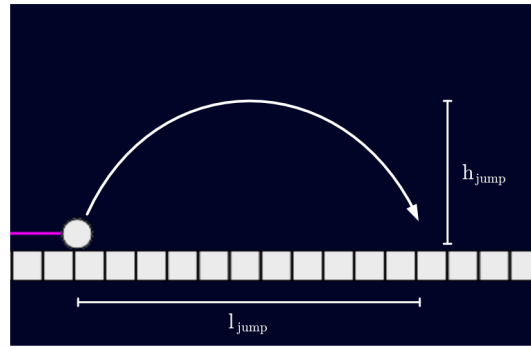


Figure 2.2: Jumping distances of a player

Formula 2.1 calculates the maximum distance traveled by the projectile in horizontal direction.  $v$  denotes the initial velocity the projectile travels in direction of the angle  $\theta$ .  $g$  is the gravitational force towards the ground.

$$l_{jump} = \frac{v^2 \sin 2\theta}{g} \quad (2.1)$$

## 2 Background

To calculate the maximum height traveled by the projectile Formula 2.2 is used. The variable  $v_y$  conforms to the initial vertical velocity and  $g$  is the gravitational force.

$$h_{jump} = \frac{v_y^2}{2g} \quad (2.2)$$

## 2.4 Regression

Regression analysis tries to model the relationship between two sets of variables  $X$  and  $Y$  by estimating an interpolation function.

There exist different models for regression, such as linear regression and polynomial regression. There are two methods to solve this problem: (1) gradient descent which uses an iterative algorithm and (2) the analytical solution which produces a direct solution. The analytical solution requires a design matrix in order to calculate the optimal parameters.

### 2.4.1 Polynomial Regression

To calculate the analytical solution for polynomial regression, the design matrix  $\Phi$  is constructed as defined in Formula 2.3, where  $x_k$  are the x-axis values of the given data points from the training set. The vector  $y$  (Formula 2.4) contains all the y-axis values of the same data points.

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ 1 & x_3 & x_3^2 & \dots & x_3^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{bmatrix} \quad (2.3)$$

## 2 Background

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad (2.4)$$

In order to calculate the optimal parameters  $\theta^*$  Formula 2.5 is used. It calculates the Moore–Penrose inverse of  $\Phi$  and multiplies the result with  $y$ . Using the vector  $\theta^*$  the interpolation function shown in Formula 2.6 can be constructed. This function must then be validated against the validation set and can then be used to predict new data points.

$$\theta^* = (\Phi^T \Phi)^{-1} \Phi^T y \quad (2.5)$$

$$f(x) = \theta_0 + \theta_1 * x + \theta_2 * x^2 + \dots + \theta_n * x^n \quad (2.6)$$

## 3 Related Work

### 3.1 Automatic Gametesting

“Automatic Gametesting” by Steurer discusses an approach to automated game testing using an implementation of the game Line Runner. The author explores how the game can be parameterized and tested automatically using a Monte Carlo simulation, which also lays the foundation for this thesis. Table 3.1 shows the parameters the author was able to extract from the game. Using these parameters he randomly generated different levels and analysed them (Steurer, 2018).

Parameter	Symbols	Description
P(flat)	$p_{flat}$	Probability of <code>BlockType.Flat</code> blocks
P(hole)	$p_{hole}$	Probability of <code>BlockType.Hole</code> blocks
P(obstacle)	$p_{obstacle}$	Probability of <code>BlockType.Obstacle</code> blocks
speed	$v$	Horizontal speed of the player
force	$f_{jump}$	Vertical force applied to the players when jumping
gravity	$f_g$	Gravity constant of the simulation (in g)
obstacle height	$h_{obst}$	Height of the obstacle blocks in the game
block length	$l_{block}$	Length of the blocks of the game
seed	$seed$	Initialization vector for the random generator used in the game. Describes a unique sequence of blocks per seed.
max players	$n_{players}$	Amount of synthetic players used for testing.

Table 3.1: Parameters extracted from the game Liner Runner in (Steurer, 2018)

The difficulty of each level was calculated accordingly to Table 3.2. The tuple of  $(d, \sigma_d)$  is used to depict how challenging a level is, where  $d$  states the difficulty of the game resulting from the simulation and  $\sigma_d$  describes the distribution of the player scores.

### 3 Related Work

Name	Symbol	Description
difficulty	$d$	The calculated difficulty of the level.
difficulty_standard_deviation	$\sigma_d$	Standard deviation of the per player difficulties

Table 3.2: Difficulty units used by (Steurer, 2018)

Steurer generated 500 games with a random parameter assignment and simulated each one with 100 players operated by an AI. The AI searches the visible game for forthcoming obstacles and calculates the perfect jump position for it. As seen in Figure 3.1, the perfect jump position centers the players jump trajectory at the center of the obstacle block.

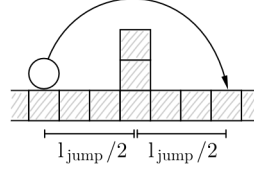


Figure 3.1: The perfect jump position for a player at an obstacle.

“Automatic Gametesting” then analyses the resulting data and highlights unique levels that were generated by the level generator (Steurer, 2018).

## 3.2 Automated Playtesting

In “Exploring Game Space Using Survival Analysis.” Isaksen, Gopstein, and Nealen present a method to predict the game difficulty of a parameterized version of the game ‘Flappy Bird’ using automated play testing (Isaksen, Gopstein, & Nealen, 2015). To calculate the difficulty survival analysis is used, a mathematical overview is given in the paper (Isaksen & Nealen, 2015). A method to efficiently search for games with a specific game difficulty is presented in (Isaksen, Gopstein, & Nealen, 2015).

Furthermore, in (Isaksen, Gopstein, & Nealen, 2015) a user study is done in order to compare the predicted difficulties with difficulty rankings gathered from human play testers. The testers were asked to play 7 pairs of games and rate each game’s difficulty on a 7-point scale compared to the

### 3 Related Work

other game's difficulty in the pair. To create the pairs of games, 6 unique variants were created with difficulties between  $d = 0$  and  $d = 0.5$ . In order to limit the amount of variables, the only parameters changed were gravity  $g$  and jump velocity  $j$ . The test was completed by 20 users and resulted in agreeing with the predictions 77.9% of the time when asked and 79.3% when comparing mean score (Isaksen, Gopstein, & Nealen, 2015), showing that the predictions are effective.

In (Isaksen, Gopstein, Togelius, & Nealen, 2018, 2015; Isaksen, 2017; Isaksen, Gopstein, & Nealen, 2015) the creation of more interesting levels is analysed and a more in-depth view on the subject of computational creativity is given. Computational creativity is the process of finding playable game variants that differ as much as possible from existing versions. These are generated by allowing the system to vary every parameter and return the games that are playable by humans. These variants are helpful for designers to explore game space, find inspiration and new ideas (Isaksen et al., 2015).

The paper (Holmgard, Green, Liapis, & Togelius, 2018) takes a similar approach by using artificially intelligent personas, controlled using Monte Carlo tree search, to construct synthetic playtesters.

### 3.3 Dynamic Difficulty Adjustment

The paper (Hunicke, 2005) explores the design requirements for a dynamic difficulty adjustment system and shows by using a user study that DDA algorithms can improve performance, while retaining the player's sense of accomplishment. In (Hunicke & Chapman, 2004) a probabilistic technique is proposed that analyses the user performance during game play to evaluate the difficulty of given obstacles. Using ideas from reinforcement learning and evolutionary computation, (Tan et al., 2011) presents two adaptive algorithms that improve player satisfaction by scaling the difficulty of the game AI. Both algorithms were able to match the different opponents in terms of mean score and winning percentage (Tan et al., 2011).



## 4 Design

The design of this project builds on the previous work of (Steurer, 2018). It takes the previously developed approaches and optimizes the level generation to produce a higher amount of playable levels. The creation of WebGL versions allows for a better data analysis.

### 4.1 Game Description

The implementation of Line Runner (Figure 4.1) used in this project is the same one that was developed in (Steurer, 2018). The following paragraph is directly cited from (Steurer, 2018):

The game developed and analysed in this work is a simple 2D platformer with constant side-scrolling that is called Line Runner. The challenge of the game is to jump over obstacles and holes and get as far as possible without dying.

## 4 Design

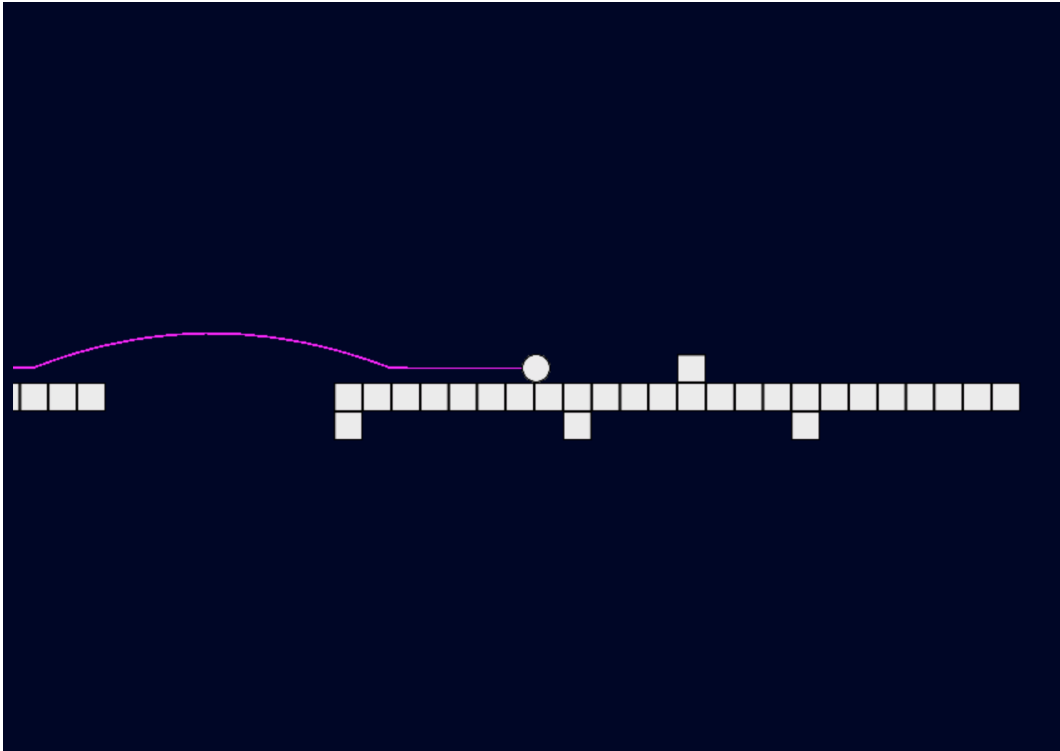


Figure 4.1: Screenshot of the game Line Runner

## 4.2 Difficulty

The difficulty calculation done in “Automatic Gametesting” allows to classify the levels by the difficulty  $d$ , but it doesn’t take into account the influence of  $\sigma_d$  (Steurer, 2018). The variance  $\sigma_d$  tells how distributed all players died, hence if  $\sigma_d = 0$ , all players must have died at the same position. Meaning that the level contains a position from which it is impossible to continue. Therefore, the variance also has an influence on the game difficulty. For this purpose and the practicality of having a single value for a game’s difficulty this thesis introduces the variable  $d_{metric}$ . It is calculated as shown in Formula 4.1, which consists of  $d$  and  $\sigma_d$ .

$$d_{metric} = d * (1 - \sigma_d) \quad (4.1)$$

## 4.3 Level Generation

The approach used in “Automatic Gametesting” uses complete randomization of the levels parameters to generate game configurations. This approach is far from optimal and results in 78.8% of levels having  $\sigma_d = 0$  and a difficulty of  $d > 0$ , which means that no player made it to the end (Steurer, 2018).

This thesis improves the level generation by adding restrictions to the Level Generator, resulting in the generation of a lot more playable levels. By identifying potential unplayable levels beforehand, the Level Generator can prevent them in the first place.

## 5 Implementation

The Line Runner implementation used in this project is built on top of (Steurer, 2018). This chapter describes the enhancements made to the level generation and all other additions made to the project. The game was ported to WebGL for easier level testing and data visualization. The following section of this thesis will go more into detail of each enhancement made.

### 5.1 Improving The Level Generation

The level generation of the game developed in (Steurer, 2018) offers a lot of surface in which improvements can be applied in order to get better levels. Figure 5.1 points out that 89.3% of all generated games have a difficulty standard deviation  $\sigma_d = 0$ , which means that at a certain position all players die at once or make it to the end position. 10.5% of the games have a difficulty  $d = 0$ , as seen in Figure 5.2. Therefore, 78.5% of the generated games are not playable at all, as all players die at once. The following two optimizations will increase the output of playable levels from the level generator.

## 5 Implementation

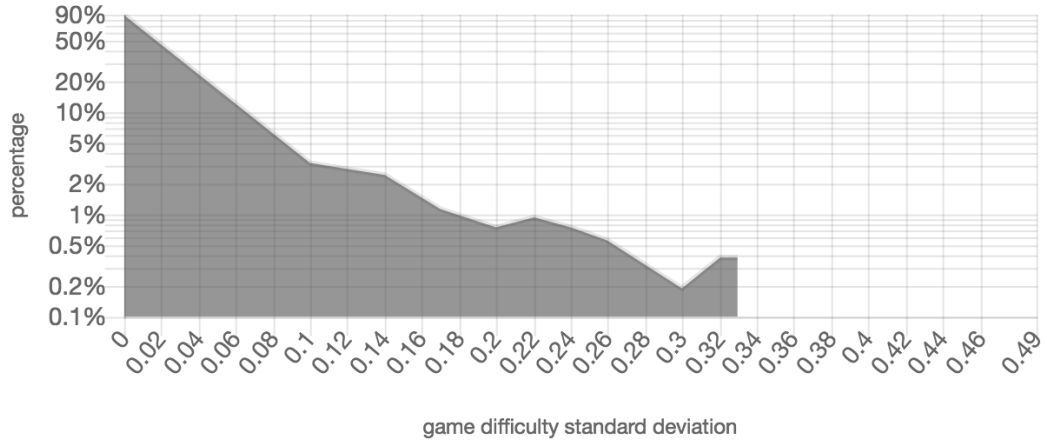


Figure 5.1: The distribution of  $\sigma_d$  generated by the level generator of (Steurer, 2018). 89.3% of all generated games have a difficulty standard deviation  $\sigma_d = 0$ .

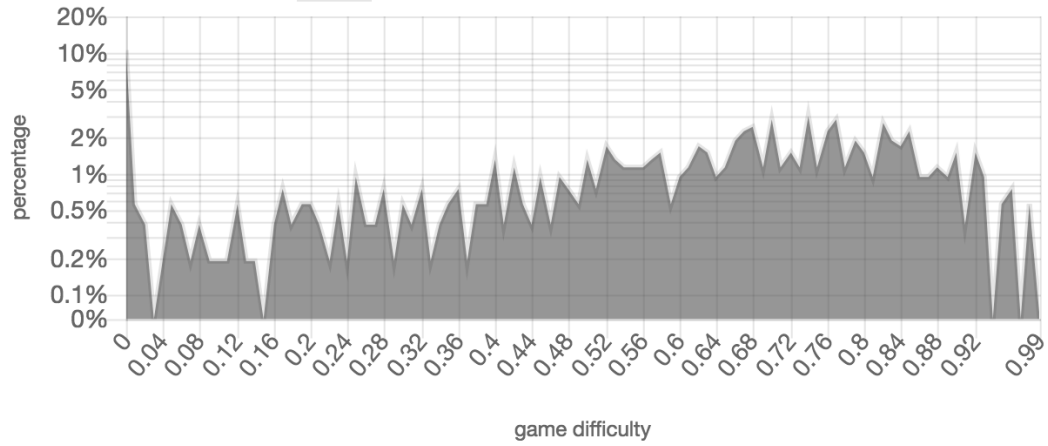


Figure 5.2: The distribution of game difficulties generated by the level generator of (Steurer, 2018). 10.5% of the games have a difficulty  $d = 0$ .

## 5 Implementation

### 5.1.1 Every Second Flat

The first improvement covered this section will lower the amount of players that die because of unplayable block sequences. One such block sequence, where a hole is followed by an obstacle, is shown in Figure 5.3. In case of an ideal player that perfectly jumps over the hole, he only has a distance of less than  $l_{block}/2$  to react to the obstacle in front of him. Due to game parameters this problem occurs in approximately 18% of all generated levels. This is due to the jumping range being too long for the player to react.

Another example would be the reverse case of an obstacle block followed by a hole block. If the player jumps further than  $l_{block}$ , he makes it past the obstacle and falls directly into the hole. If he can't jump further than twice  $l_{block}$ , he won't be able to make it past the hole.

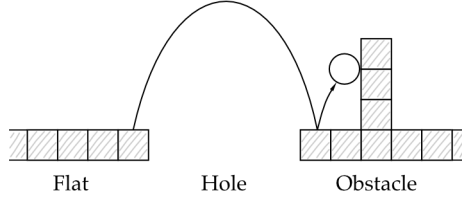


Figure 5.3: Issue that is solved by Every Second Flat

This issue can be fixed by forcing the level generator to keep every even numbered block of type `Blocktype.Flat`.

This improvement, called "Every Second Flat", reduces the amount of unplayable levels from 78.8% to 64.8%.

### 5.1.2 Smart Level Generation

The improvement "Smart Level Generation" passes the parameters regarding the players into the Level Generator. These parameters are used to calculate the maximum jump height  $h_{jump}$  and the jump distance  $l_{jump}$  in order to disregard levels where the players aren't able to jump over holes or obstacles in the first place. The two cases where all players die instantly because of falling in a hole or crashing into an obstacle are illustrated in Figure 5.4.

The improvements to the level generation show that the amount of unplayable levels is reduced from 78.8% to 34.8%.

## 5 Implementation

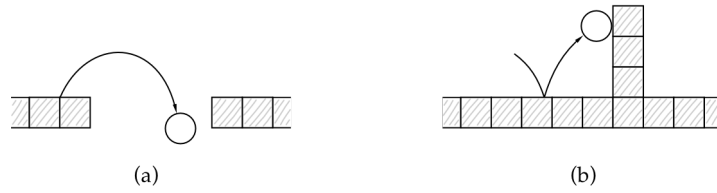


Figure 5.4: Problems that are solved by Smart Level Generation

### 5.2 WebGL Versions

The implementation of Line Runner used in this work is enhanced by a WebGL component, which provides cross-platform compatibility of the game. This aspect is very important for the user testing that has been conducted for this thesis, because it allows to broaden the target group. The game itself is developed in Unity using C# as programming language and is exported using a custom-built WebGL template that allows to add various HTML components to the site. The simplicity of HTML enables the website to be populated by sliders that allow each parameter to be tweaked directly and the survival graph that shows the current amount of survivors at any point in the simulation. While the Line Runner itself is written using C# code, the web component uses JavaScript. The communication between the two components is done via an interface provided by Unity that allows exchanging information via function calls. This works in both directions, meaning that you can call JavaScript code from C# and vice versa.

#### 5.2.1 WebGL Level Generation And Testing Site

The level generation and testing site is used to generate and discover different levels and testing them in a simulation. This page is used for any level generation or discovery throughout this thesis.

This site's performance is dependent on the machine that is used, a more powerful machine will perform more accurate and is able to handle simulations at a higher simulation speed. During the development of this thesis a 2014 MacBook Pro was used as main test and development machine. The performance of the simulation on this machine also varied from browser to browser. While Safari wasn't able to run the simulation smoothly, Google

## 5 Implementation

Chrome had no performance issues at all.

The layout of the page, as shown in Figure 5.5, consists of the game view, a sidebar and the history. The game view on the top left contains the simulation of the level, this part corresponds to the version built using Unity in (Steurer, 2018). The sidebar holds buttons to change the simulation mode, a slider for each game parameter that can be altered, a slider for handling the simulation speed and finally the survival chart.

The simulation speed slider allows to tweak the `Time.timeScale` parameter of the Unity scene. If the `Time.timeScale` gets raised too high, the simulation will become inaccurate and the results are useless for the later analysis. The survival chart displays how many players were alive at each point in time during the simulation. It gives an overview on how difficulty a level is and whether the level contains any difficult positions where multiple players die at once.

Below the game view, the game history displays all previously simulated games. These simulated games have been submitted to the collection server and inserted into the server-side MySQL database. Furthermore, each level stored in the history can be resimulated or exported as a JSON file that can later be reimported. The JSON in Listing 5.1 poses as an example on how such a JSON is structured. The JSON contains all parameters used to describe the game and also contains the results computed using the game simulation.

Listing 5.1: A exemplary result of a simulation in JSON format.

```
{
  "p_flat": 0.6,
  "p_hole": 0.4,
  "p_obstacle": 0,
  "speed": 11.26,
  "force": 6.63,
  "gravity": 1.04,
  "obst_height": 1,
  "block_length": 13,
  "seed": 120124950,
  "score": 30,
```



## 5 Implementation

```

"difficulty": 0.71,
"difficulty_standard_deviation": 0.14,
"survivors": 1,
"max_players": 100
}

```

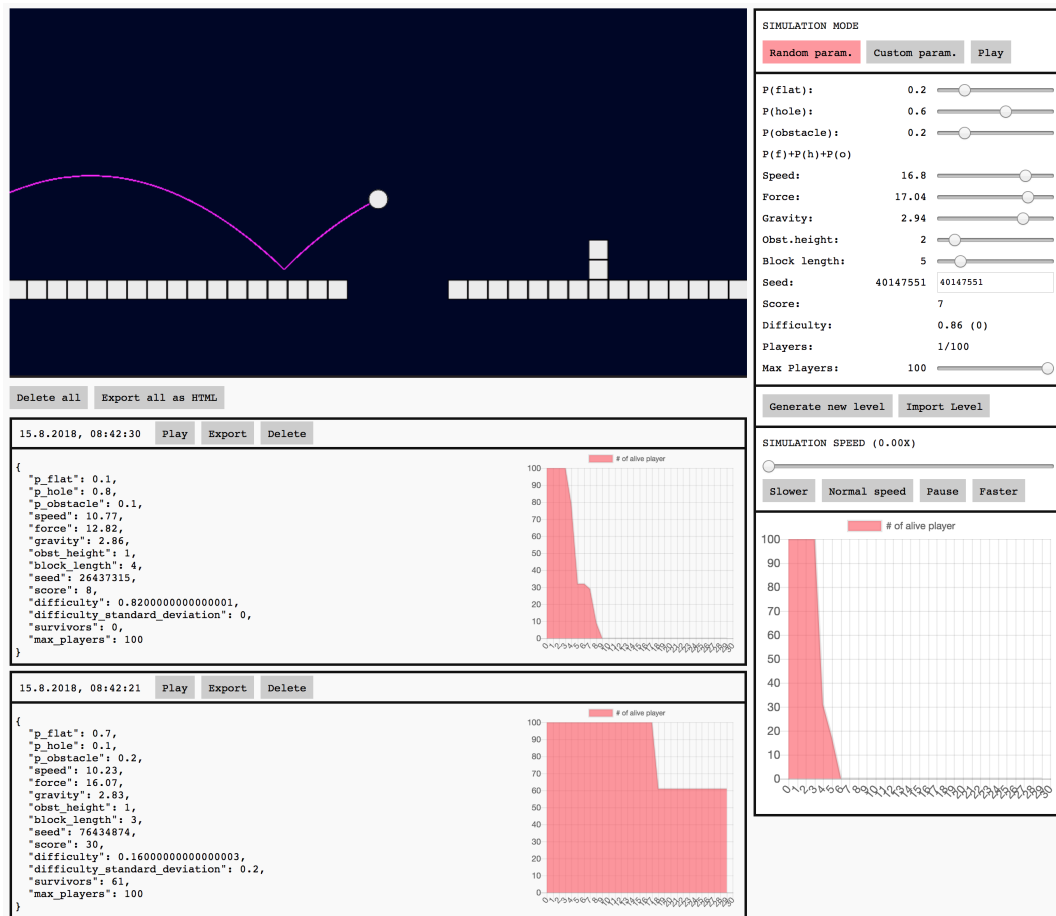


Figure 5.5: Overview of the WebGL level generation and testing site

## 5 Implementation

The sidebar allows to change the simulation mode of the site between random, custom and play mode. They are needed for level discovery and testing of the levels.

### **Random Mode**

The random mode generates random levels and simulates them. This simulation mode is a WebGL wrapper of the simulation built in (Steurer, 2018) with functional additions. At the end of each simulation the parameters of the game and their estimated difficulty values are saved to the history and sent to a server for further evaluation.

### **Custom Mode**

During custom mode, the user can modify the parameter's values and thus simulate a customized level. These levels aren't randomly generated and therefore aren't object to the server-side analysis. The custom mode passes the parameters chosen in the sidebar into the WebGL simulation using the Unity interface and replaces the previously set parameters with the new ones. Afterwards the simulation is started with these parameters.

### **Play Mode**

Play mode allows to test the current parameter configuration firsthand. Using this mode the level can be played directly in the browser to get a first impression how the level plays and feels. Any previously set configuration by either the random mode or the custom mode can be tested this way, since the parameters will be left unchanged. When switching to play mode, the WebGL view will start catching any user based keyboard input and therefore any text input field on the page will cease to work.

### **5.2.2 User Testing Site**

Another addition poses the User Testing Site, which is used for testing a selected amount randomly generated levels. This site is built for the sole purpose of user testing.

First the page fetches a randomly selected level configuration from the level

## 5 Implementation

collection server. Afterwards the game view switches into game mode and the user can start playing the loaded level. At the end of each level he is then confronted with three different options to continue. He can either rate the difficulty and the fun factors of the game, play the same level again or try another level configuration. The ratings submitted by this page are saved on the collection server and can further be analysed on the Data Collection and Visualization site. If the user decides to try a level again, the game will be reset and started from the beginning. On the other hand if he decides to try a new level, a new configuration is fetched from the collection server and afterwards the game will be restarted.

### 5.2.3 Data Collection And Visualization

In order to gain a better understanding of various factors of the game simulation and get a better understanding of the influence of game parameters, 2749 random games were simulated and the results were stored in a mySQL database. This data is used to create graphs and tables which are used for the analysis.

#### Game Data

The Game Data site is used to display and analyse the data from all randomly generated levels. It contains graphs that allow to compare the parameters and results of the different games created. Furthermore, it automatically generates graphs that compare the distribution of difficulty and its standard deviation among the games. All these graphs compare the impact of the level generation improvements described in section 5.1 and is subject to the discussion in chapter 6. Underneath these graphs a table that contains all generated levels from the database is shown. These levels can be sorted by their parameters in either ascending or descending order.

#### User Testing Data

In order to visualize and analyse the data collected during user testing, the ratings visualization on the User Testing Data site is used. This site is supposed to list all levels, their user ratings and illustrate this data within multiple graphs. The information illustrated on this site will be discussed

## 5 Implementation

later in the analysis in chapter 6. At the top of this site a table with all levels is shown, which allows to quickly view and compare key properties of individual levels. The table can be sorted by any property to get a better understanding of the data. Underneath the table multiple graphs visualize a comparison of various properties like game difficulty and results from the user testing. On the bottom of the page a second table shows all users that participated in the user tests together with statistics like their average ratings.

For the purpose of getting a more detailed insight into the data of a specific game, a game page is shown when clicking on a game in the table. This page not only contains all game parameters and average ratings, but also a list of all ratings of the game. Underneath this table, graphs visualize relations of the ratings.

A similar page is displayed when clicking on a user. This page contains a list of all games rated by the user and charts to show relations of his ratings. In chapter 6 the data visualized on this site is further analysed.

## 6 Analysis

This chapter uses the results of 2749 simulated games to analyse the influence of the level generation enhancements and the influence of parameters. Furthermore, this section includes one dimensional sampling of parameters, parameter prediction and the results of the user test.

### 6.1 Influence Of Level Generation Enhancements

The improvements to the level generation in this thesis have done a pretty notable change to the difficulty distribution across all levels. The Level Generator is able to produce a lot more playable games when used in random mode. The following graphs are generated from data collected in the database and are used to highlight the impact of the improvements.

## 6 Analysis

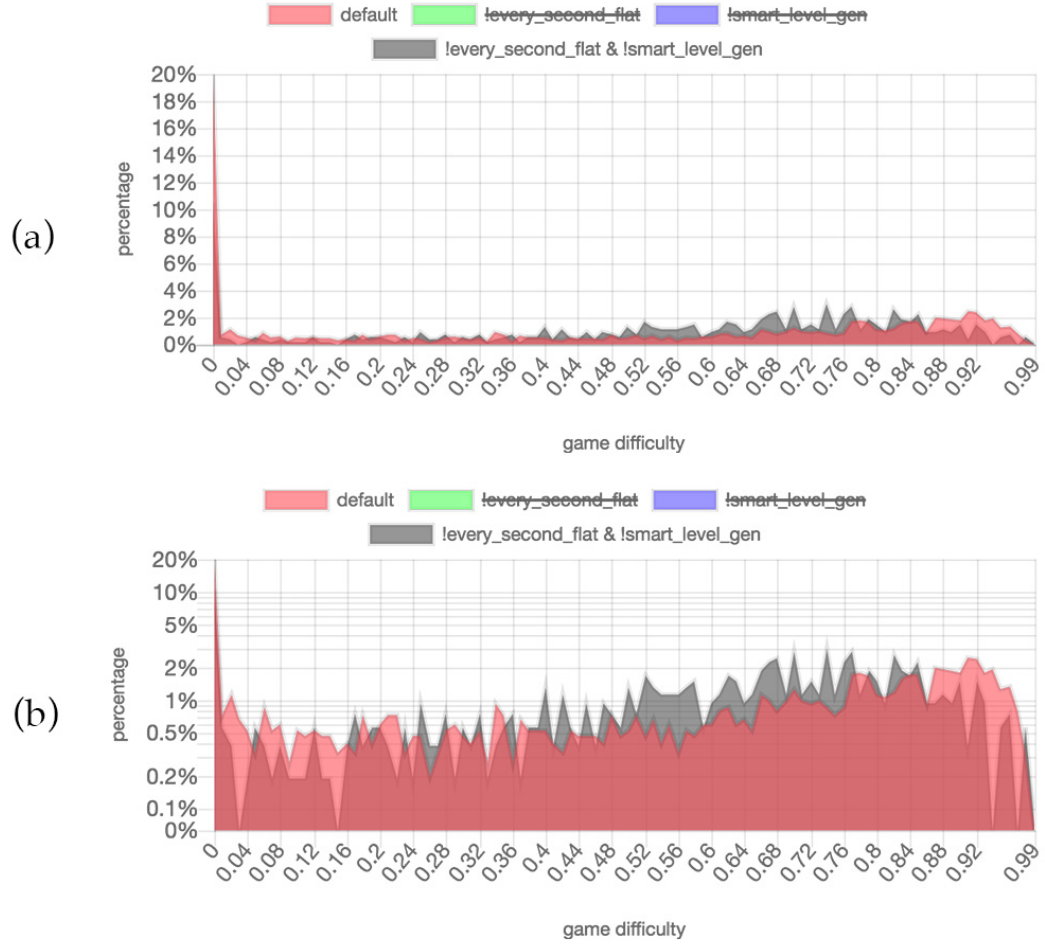


Figure 6.1: These graphs show the influence of the level generation enhancements by comparing the game difficulty  $d$  probability. Graph (a) uses a linear scale and graph (b) visualizes the identical data with a logarithmic scale.

## 6 Analysis

The graphs in Figure 6.1 show that by using the enhancements easier games are generated. The amount of games with  $d = 0$  has risen about 9 percentage points from 10.5% to 19.89%.

Without these improvements, the amount of randomly generated games with a difficulty standard deviation  $\sigma_d = 0$  was 89.3% as seen in Figure 6.2 (a). Figure 6.1 (a) shows that only 10.5% of the generated games without the improvements have a difficulty  $d = 0$ , meaning in 78.8% of generated levels all players die at one point, making them impossible to play. The level generation improvements reduced this amount greatly with only 54.67% of randomly generated games having a difficulty standard deviation  $\sigma_d = 0$ , as seen in Figure 6.2 (a). The other 45.33% of the games are the most interesting for this thesis, since they are playable and yet not too easy. Despite the fact that the amount of levels with  $d = 0$  has increased to 19.89%, the amount of impossible games has dropped from 78.8% to only 34.78% because of the level generation improvements. Figure 6.2 (a) shows the  $\sigma_d$  probability distribution with a linear scale, Figure 6.2 (b) visualizes the same distribution with a logarithmic scale and Figure 6.2 (c) shows the distribution for each individual level generation improvement. The improved version scored the best results in generating the highest amount of levels with higher difficulty standard deviation  $\sigma_d$ , as can be seen in Figure 6.2 (c).

The smart level generation is one of the main reasons for the decreased amount of impossible games generated. Its basic functionality is to calculate the jump distance  $l_{jump}$  and jump height  $h_{jump}$  and discard games where player can not jump over obstacles or holes. The calculation of the jump distance  $l_{jump}$  traveled by a player can be calculated using Formula 6.1.

$$l_{jump} = \frac{v_0^2 \sin 2\theta}{g} \quad (6.1)$$

Under closer consideration, Formula 6.1 shows that the block length is dependent on the values of gravity  $f_g$ , jump force (influencing the angle  $\theta$ ) and the speed  $v_0$  of the player. Figure 6.3 (a) shows that the bigger  $l_{block}$ , the lower  $f_g$  has to be in order for the player to jump over holes. A similar property can be observed in Figure 6.3 (b), the greater  $l_{block}$ , the bigger  $f_{jump}$  has to be to jump over holes. For the speed parameter  $v$  this also applies, because a higher value for  $l_{block}$  means that the player has to be faster to be able to jump over holes (Figure 6.3 (c)).

## 6 Analysis

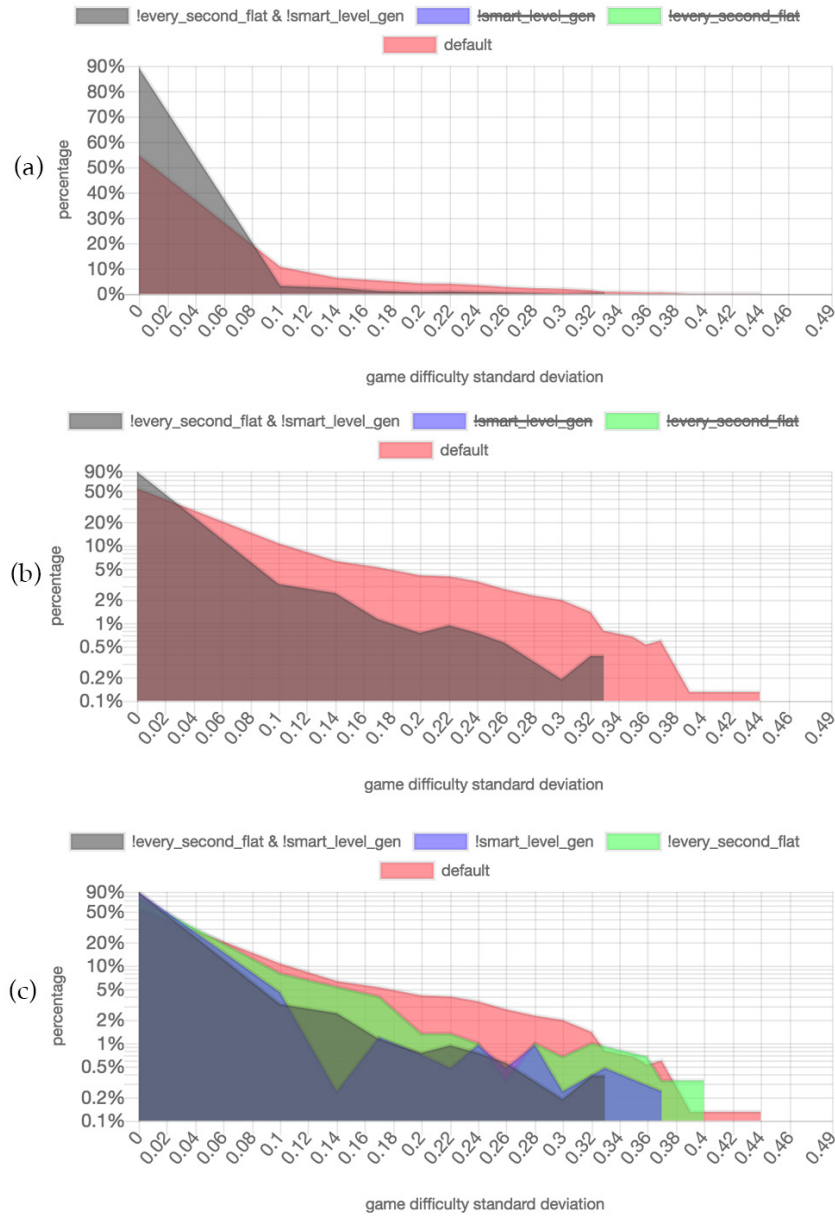


Figure 6.2: These graphs show the influence of the level generation enhancements by comparing the  $\sigma_d$  probability. Graph (a) uses a linear scale and graph (b) visualizes the identical data with a logarithmic scale. Graph (c) gives a more detailed insight to the influence of the individual improvements.



## 6 Analysis

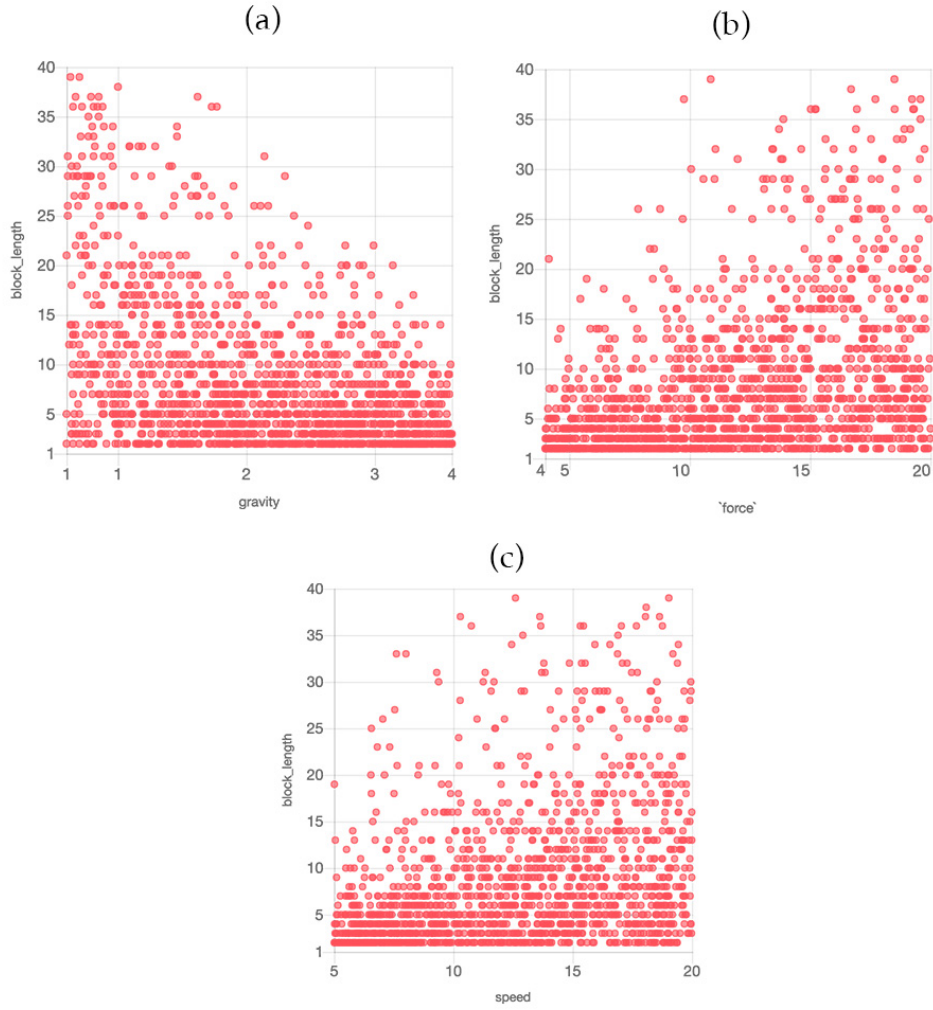


Figure 6.3: Due to the smart level generation, the block length  $l_{block}$  depends on the parameters  $v$ ,  $f_{jump}$  and  $f_g$ .

## 6 Analysis

Equation 6.2 is used to calculate the players jump height. It shows that the jump height and therefore the obstacle height is dependent on the values of the player's jump force  $f_{jump}$  and the gravity  $f_g$  of the simulation. Figure 6.4 (a) shows that the higher  $h_{obstacle}$ , the bigger  $f_{jump}$  has to be in order for the player to jump over obstacles. For  $f_g$  similar properties are observable, the higher  $h_{obstacle}$ , the smaller  $f_g$  has to be, as can be seen in Figure 6.4 (b).

$$h_{jump} = \frac{v_y^2}{2g} \quad (6.2)$$

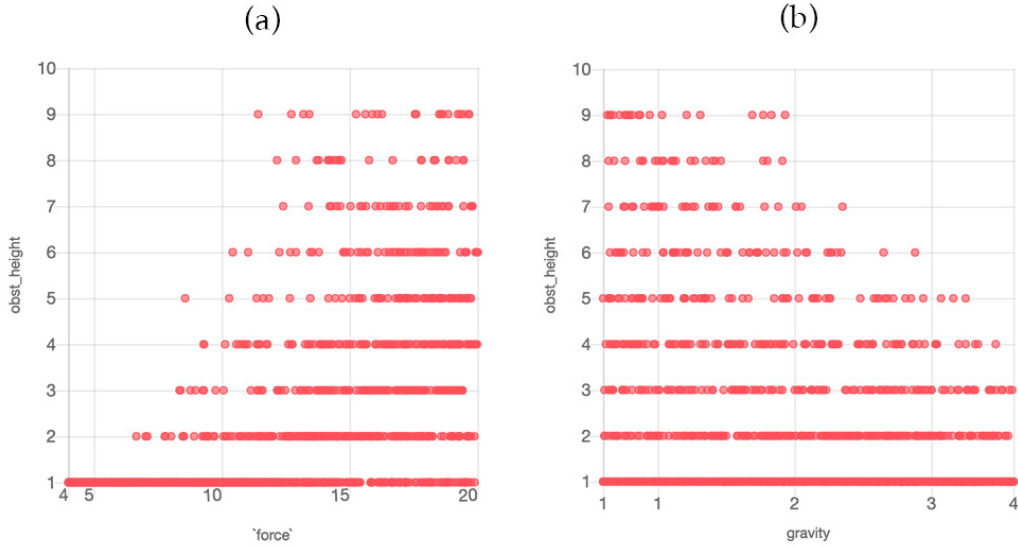


Figure 6.4: Due to the smart level generation, the obstacle height  $h_{obstacle}$  depends on the parameters  $f_{jump}$  and  $f_g$ .

## 6.2 Influence Of Parameters

This section evaluates the influence of different game parameters to the game difficulty. The following graphs are created from the games simulated with random parameter selection, which means that the results seem more variable than those obtained by one-dimensional-sampling. Nevertheless, the influence of the parameters is clearly visible.

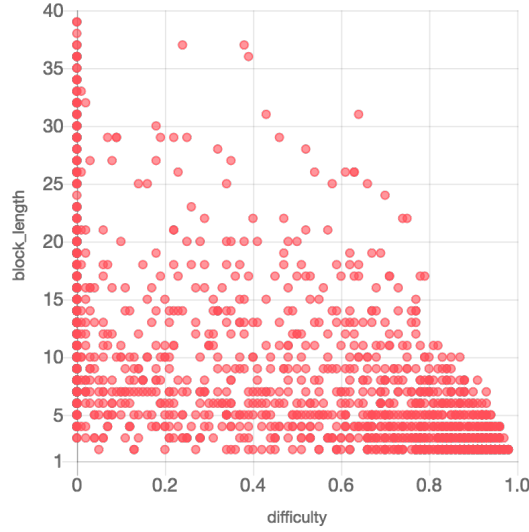


Figure 6.5: The influence of the parameter  $l_{block}$  on  $d$ . Games with a small  $l_{block}$  tend to have a high  $d$ .

Games that were generated with a smaller  $l_{block}$  value often also have a higher difficulty. The opposite case, games with a bigger  $l_{block}$ , tend to have a small difficulty, as Figure 6.5 illustrates. When increasing  $l_{block}$  the data points in the graph become more and more scattered to the left side of the graph. This happens due to the fact that when  $l_{block}$  is increased, the time players need to reach the first obstacle also rises proportional. The first possible point in the game where players can die takes more and more time to be reached which results in a decrease of the difficulty  $d$  for the level.

For games with a small  $l_{block}$  the best score  $S$  tends to rise when  $l_{block}$  is raised. On games with a higher  $l_{block}$  this trend continues to show, but more and more games begin to reach the maximum score  $S_{max}$ , as shown in Figure 6.6.

## 6 Analysis

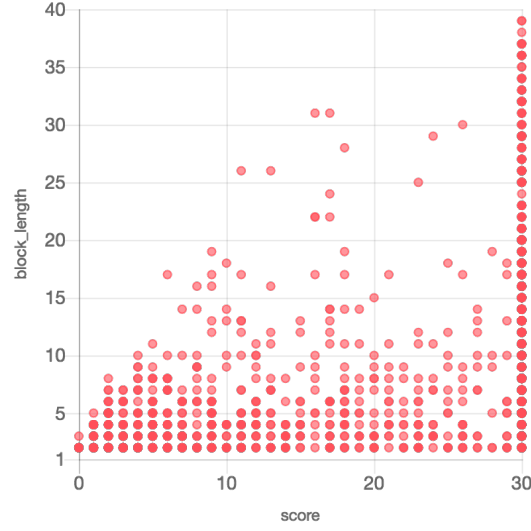


Figure 6.6: Correlation between  $l_{block}$  and  $S$ . The higher  $l_{block}$ , the higher the amount of games with  $S = 30$ .

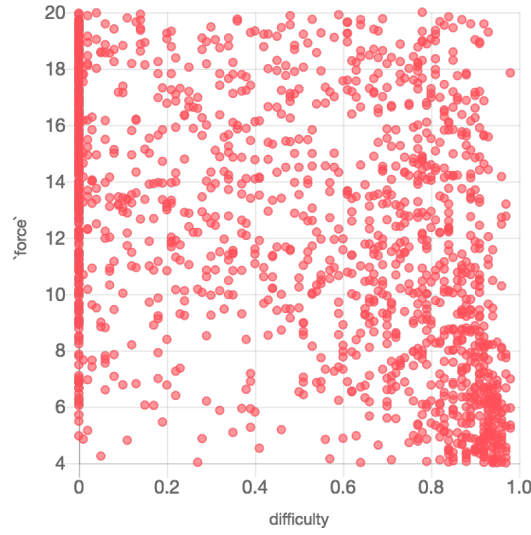


Figure 6.7: The influence of  $f_{jump}$  on  $d$ . The lower  $f_{jump}$ , the more games with high  $d$ .

## 6 Analysis

The jump force's influence on the difficulty  $d$  can be seen in Figure 6.7. Games that have a small  $f_{jump}$  value are mostly very difficult. When  $f_{jump}$  is raised in direction of the maximum, the distribution of the difficulties on the graph becomes more and more uniform. This means in order to generate games with a higher playability  $f_{jump}$  must also be greater.

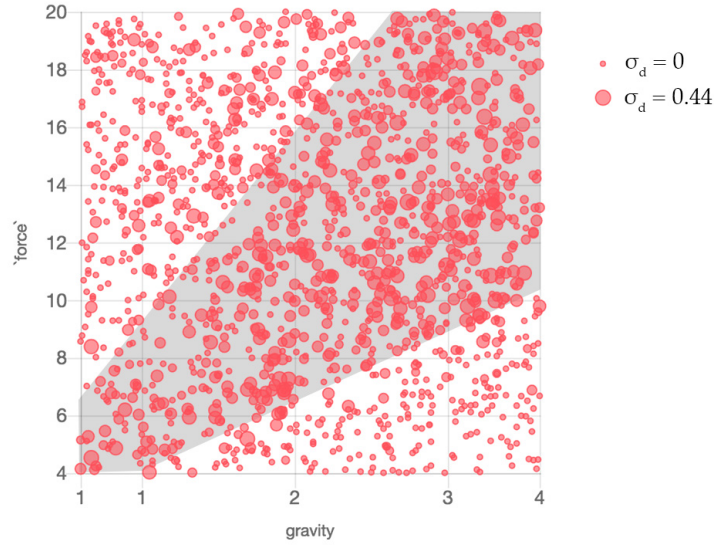


Figure 6.8: This graph shows the impact of the parameters  $f_{jump}$  and  $f_g$  on  $\sigma_d$ , which is visualized using the radius of the points. The gray area illustrates the optimal combination of  $f_{jump}$  and  $f_g$ .

When comparing  $f_{jump}$  to  $f_g$  with  $\sigma_d$  used as point radius a clear channel of games with a high  $\sigma_d$ , reaching from the bottom left to the top right, reveals the best combinations between  $f_g$  and  $f_{jump}$  (Figure 6.8). This 3D parameter analysis allows to find games with a high probability to achieve a flow state.

Another interesting graph is created when comparing  $d$  and  $S$  as can be seen in Figure 6.9. It shows the logical relation of these two parameters, namely the lower the difficulty  $d$ , the higher the score of the best player  $S$ .

## 6 Analysis

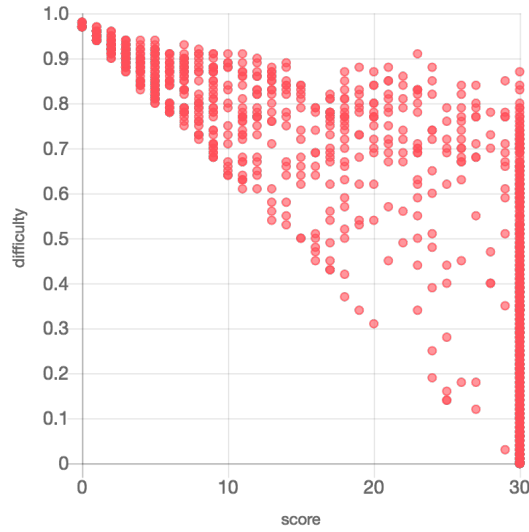


Figure 6.9: The correlation of  $d$  and  $S$ . The lower  $d$ , the higher  $S$ .

### 6.3 Game Difficulty Variation

In order to explore the variability of the game difficulty 20 different games with  $\sigma_d > 0$  were examined. Each one of the 20 levels was simulated 15 times with the exact same parameters. As Figure 6.10 shows, the variability of the difficulty resulted in a standard deviation of 0.022 for  $d$  and 0.024 for  $\sigma_d$ . This means, that the variability is rather small and can therefore be ignored in this work. This happens because the levels are tested with 100 players simultaneously, which suppresses the variability a considerable amount.

## 6 Analysis

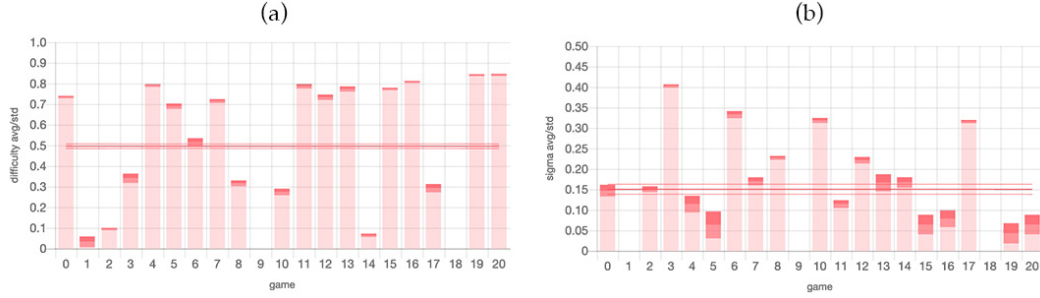


Figure 6.10: The variation of (a)  $d$  and (b)  $\sigma_d$  for 20 games. The bars illustrate the average value and its standard deviation for each game. The Lines show the average value and standard deviation of all games.

### 6.4 One Dimensional Sampling

In one dimensional sampling multiple simulations are run where all parameters are kept fixed except one. This parameter is sampled at a fixed interval in a defined range which allows to do an analysis on how the parameter impacts a game's difficulty.

#### 6.4.1 Speed Variation

It can be observed that the speed has a high impact on a game's difficulty. Figure 6.11 shows four different levels that were analysed by altering the speed parameter from minimum to the maximum value. While one might tend to think that by raising the speed parameter the game becomes more challenging, Figure 6.11 shows that this isn't actually the case.

##### Difficulty

For each game can be observed that there exists a value for the speed parameter at which the game becomes the easiest. Level 3 and level 4 in Figure 6.11 are the easiest at  $v = 12.0$ . If the speed parameter then gets further increased or decreased, the game becomes more difficult. Level 1 has its lowest difficulty value 0 at  $v = 18.0$  and when  $v$  is decreased it climbs to a difficulty of 0.77. Since  $d$  isn't only depending on the parameter  $v$ , there is no way to know where the minimum in the game speed parameter function

## 6 Analysis

will be and how the difficulty changes.

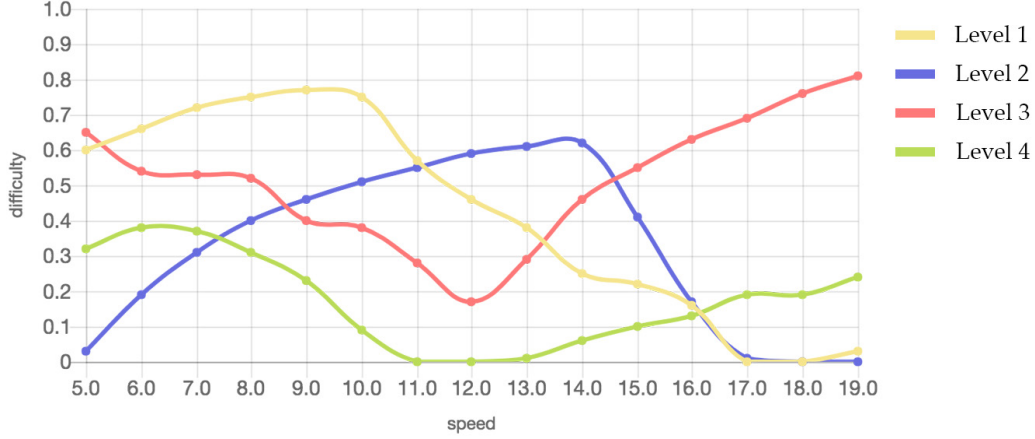


Figure 6.11: This graph shows how the difficulty  $d$  behaves if the speed  $v$  is varied for four exemplary games.

### Difficulty Standard Deviation

The difficulty standard deviation  $\sigma_d$  shows a similar behavior like the difficulty. Each level shown in Figure 6.12 has its own minimum, which has the same  $v$  value as with  $d$ , where the game is the easiest. For example Level 1 starts with a value of  $\sigma_d = 0$  where the difficulty shown in Figure 6.11 is greater than 0. This means that at some point all the players die at once. When the speed continues to grow,  $\sigma_d$  also grows to 0.41. The difficulty on the other hand begins to fall, meaning the level gets easier, but the players are still dying relatively distributed. As the speed continues to grow  $d$  falls to 0 and so does the standard deviation, resulting in the easiest variant of the level.

### 6.4.2 Block Length Variation

$l_{block}$  has a similar effect on a level's difficulty like  $v$ . As shown in Figure 6.13 (a), the difficulty function drops to a local minimum before rising again.



## 6 Analysis

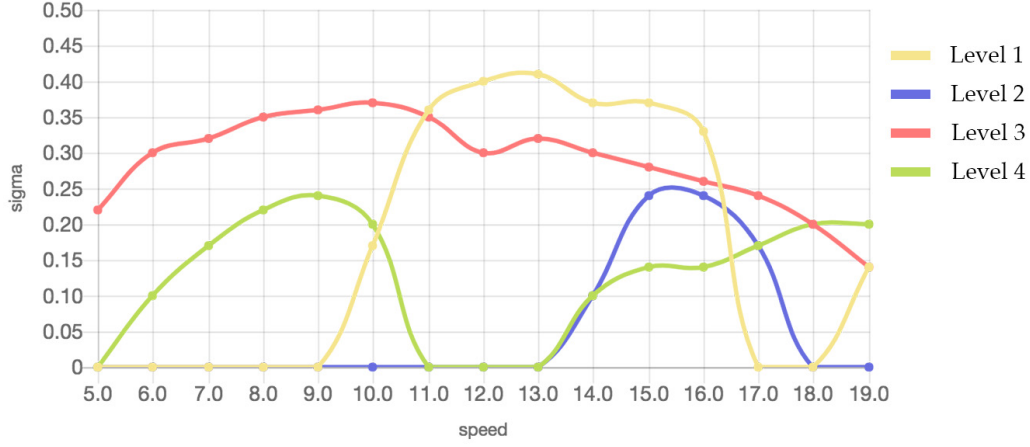


Figure 6.12: This graph shows how the difficulty standard deviation  $\sigma_d$  behaves if the speed  $v$  is varied for four exemplary games.

The position of this minimum correlates to the value of  $l_{block}$  where the level is the easiest to play.

After the minimum, the function rises again to a local maximum and then starts to slowly fall again. As seen in Figure 6.13 (b), after this point the  $\sigma$  value becomes zero indicating that a  $l_{block}$  has been reached where no more players manage to achieve the maximum score  $S_{max}$  and all players die at one point. The difficulty function slowly declines towards 0, which can be explained by illustrating the following example:

A level contains a block index  $i_{death}$  where all players die at once. If  $l_{block}$  is increased, the distance to the block index  $i_{death}$  is also increased directly proportional to the block length as illustrated in Formula 6.3. Meaning that the death position also increases and therefore the difficulty  $d$  decreases.

$$P_{x_{death}} = i_{death} * l_{block} \quad (6.3)$$

Figure 6.14 shows the average difficulty variation initiated by  $l_{block}$  among 20 randomly picked levels with  $\sigma_d > 0$ . It shows, that the difficulty slowly declines when increasing  $l_{block}$  and that it contains a local minimum where

## 6 Analysis

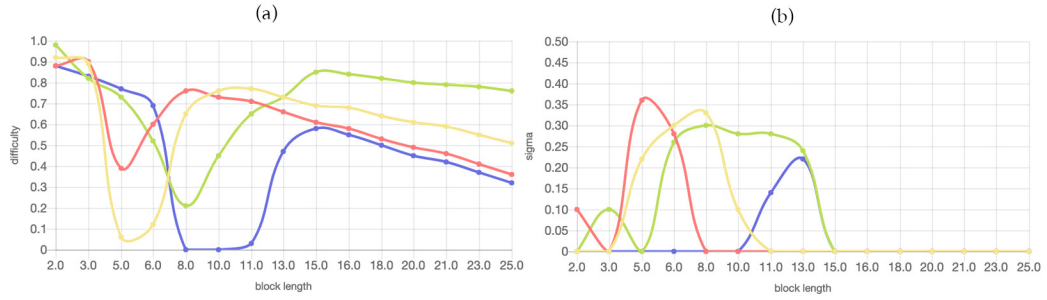


Figure 6.13: Graph (a) shows how  $d$  and graph (b) how  $\sigma_d$  behave if  $l_{block}$  is varied for four exemplary games.

the games are the easiest to play. Furthermore, the graph shows that there exists a value of  $l_{block}$  for every level where  $\sigma_d$  stays 0 when  $l_{block}$  is raised. This indicates that holes have become too long for players to jump over, leading to a collective death of all players.

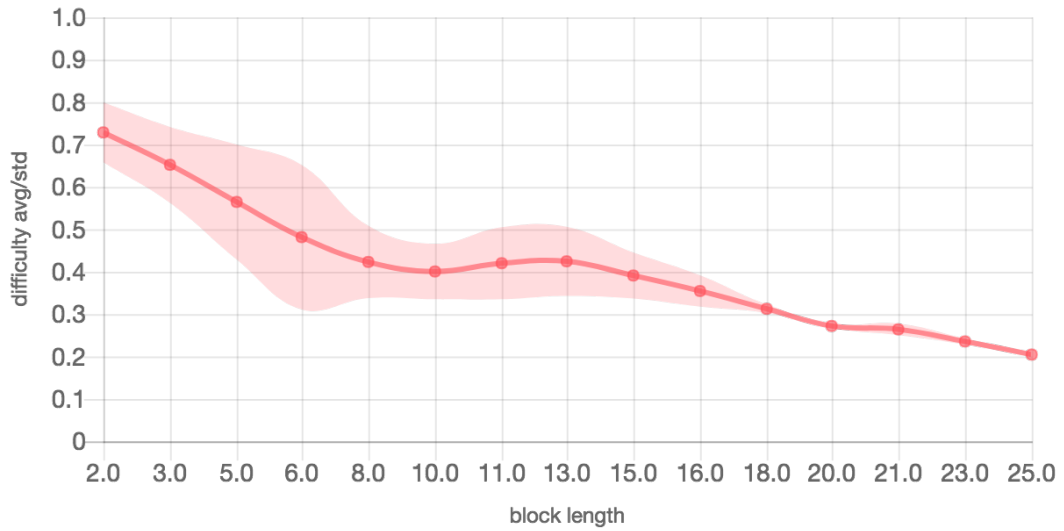


Figure 6.14: The line in this graph illustrates the average  $d$  when sampling  $l_{block}$ . The area around the line represents the average  $\sigma_d$ .

### 6.4.3 Seed Variation

By only changing the seed parameter further effects on the difficulty of the game are analysed. The same games as in section 6.3 were used for this test. As seen in Figure 6.15, the resulting game difficulty  $d$  has an average standard deviation of 0.09 (Figure 6.15 (a)) and  $\sigma_d$  has an average standard deviation of 0.06 (Figure 6.15 (b)). When compared to the results of the game difficulty variation in section 6.3, a higher impact on the variation of a game's difficulty can be seen. An explanation of this fact can be made by illustrating the role of the seed in the level with an example.

Assuming a level has a sub-block-sequence with a high difficulty. When the level is generated with a different seed, the sequence of the blocks is randomized differently. This results that the sub-block-sequence, can be generated earlier or later in the game. The difficulty will adapt to this change and rise or fall likewise, leading to a higher variation.

## 6 Analysis

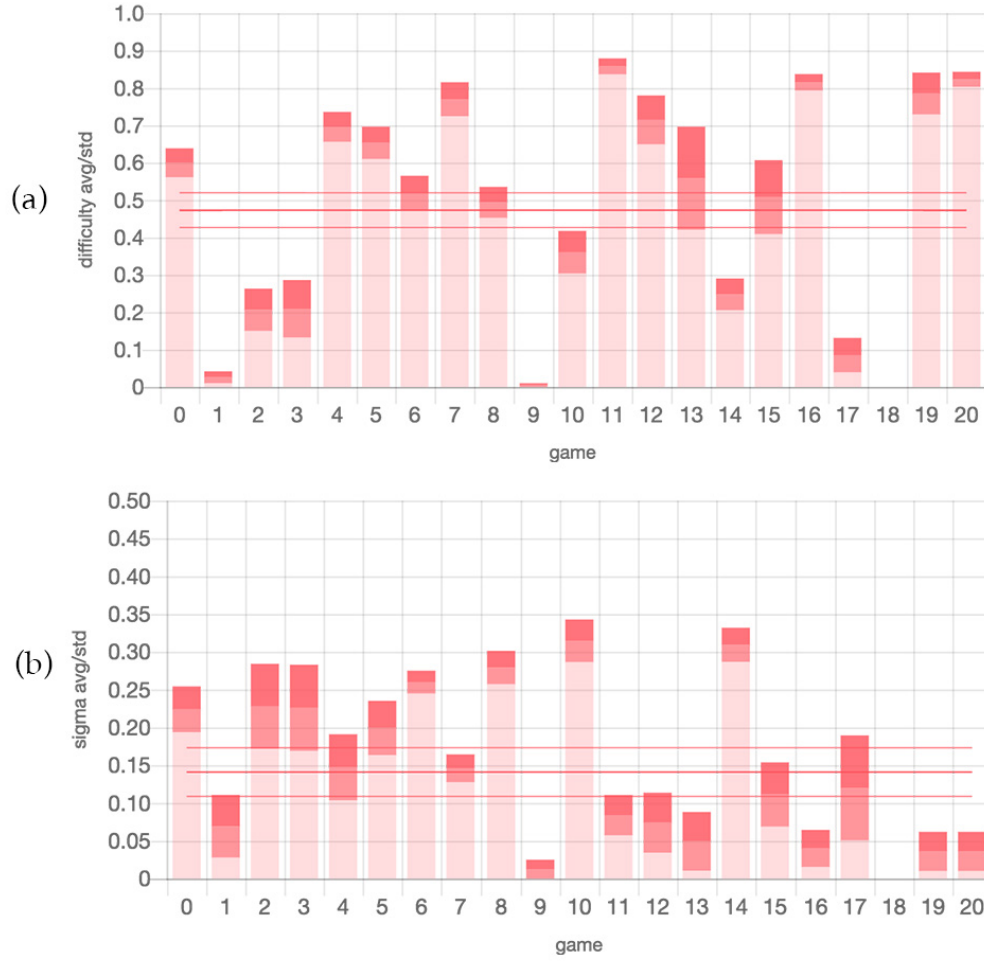


Figure 6.15: The deviation of  $d$  (a) and  $\sigma_d$  (b) when altering the seed for 20 different levels. The bars show the average value and standard deviation for each game. The horizontal lines show the average value and standard deviation of all games.

## 6.5 Parameter Prediction Using Regression

Polynomial regression is used in this work to interpolate the data in the 2D space of speed and difficulty in order to predict an assignment of the parameters to reach a certain difficulty.

For testing purpose all parameters of a single level are kept fixed except for the speed parameter. The level will be simulated with 15 different speed values in the range between 5 and 20 in order to approximate the real difficulty value. The resulting data set is split into a training set of 8 data points and a validation set of 7 data points. In order to generate an optimal interpolation function, polynomials of a different degree are trained using the training set and afterwards evaluated via the validation set. The polynomial with the lowest validation error will be further used for the speed parameter prediction. The best results were reached with a polynomial of degree 6.

Equation 6.4 describes the interpolation function for an exemplary game and a comparison of this function to the sampled values is shown in Figure 6.16. To calculate the value of  $v$  that produces a difficulty of  $d = 0.5$  for this game, the variable  $y$  in Equation 6.4 needs to be substituted by 0.5 and the equation solved to  $x$ .

This results in 7.75282 and 14.6821 as possible values for the speed parameter. Lastly this result needs to be verified by simulating both values about 10 times each, which results in the values shown in Table 6.1. The results show that the predicted values for  $v$  produce a game with almost the exact wanted difficulty.

$$\begin{aligned}
 y = & 17.7171189 \\
 & -10.7466443 * x \\
 & +2.69914919 * x^2 \\
 & -0.34586153 * x^3 \\
 & +0.02371903 * x^4 \\
 & -0.00082672 * x^5 \\
 & +0.00001150 * x^6
 \end{aligned} \tag{6.4}$$

## 6 Analysis

$v$	$\mu$	$\sigma$
7.75282	0.5078	0.03073
14.6821	0.527	0.02312

Table 6.1: Average results of the simulation using the values calculated via regression.

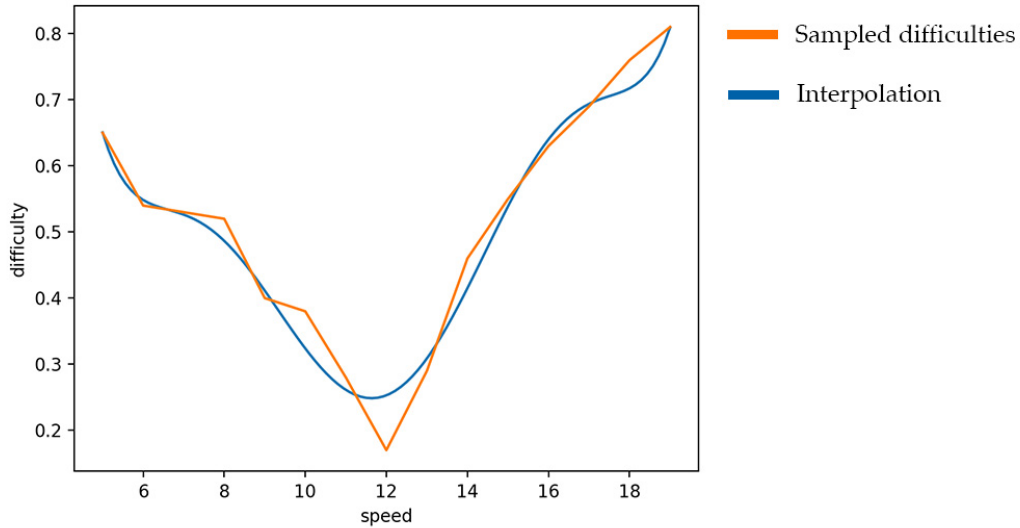


Figure 6.16: This graph compares the function generated with polynomial regression to the sampled difficulties.

### 6.6 User Testing

To verify the correctness of the predicted difficulties a user study using human play testers was done. The users were instructed to play different levels using the previously described testing webpage and rate the difficulty and fun level of each game on a scale from 1 to 5, where 1 is the lowest rating and 5 highest rating.

Although users perceive the difficulty of the levels differently and therefore also rate the difficulty of game differently, by using the mean value of the ratings these uncertainties can be minimized and the results give a good insight on how difficult a level was for human players to play.

Unlike the user study done in (Isaksen, Gopstein, & Nealen, 2015), which obtained the relative difficulty of levels by comparing a pair of two games,

## 6 Analysis

this user study rated the levels at an absolute scale. The participants were able to rate each game independently based on their current game experience.

Furthermore, the effects of the learning rate (Isaksen & Nealen, 2015) for this user testing were minimized by allowing the test users to replay a game until they felt confident to rate the difficulty of a game.

The study was completed by 11 participants with different game expertise, which gave a total of 146 ratings. All users were supervised during the study in order to prevent uncertainties.

### 6.6.1 Level Selection

20 different levels were selected for user testing. These levels were chosen by hand from the database of random generated levels with the intention to select different outliers. All selected levels are playable and span in a difficulty between easy to very hard.

### 6.6.2 Results Of User Testing

The results from user testing greatly show this work's effectiveness on predicting game difficulty. The graph in Figure 6.17 (a) shows how the calculated game difficulty  $d$  relates to the rated difficulty and confirms that the predictions were mostly correct. Figure 6.17 (b) shows the calculated difficulty  $d_{metric}$  and the user rated difficulty for each game. There are some outliers, for example one game seems to have a quite high estimated difficulty ( $d_{metric} = 0.76$ ), but received an average rating by the test users of only 2. This game has a high difficulty  $d_{metric}$  because after jumping over the first obstacle most players land in a hole. This is because the AI does not optimize the jump position if the player is able to jump over multiple obstacles. The human play testers on the other hand do this subconsciously.

The average score achieved by the users also relates as expected to the calculated game difficulty  $d_{metric}$ , the higher the difficulty, the lower the score, as seen in Figure 6.18. The user score substantiates the capability of this work's difficulty predictions, since the score is not influenced by the possible irregularities of the user ratings.

## 6 Analysis

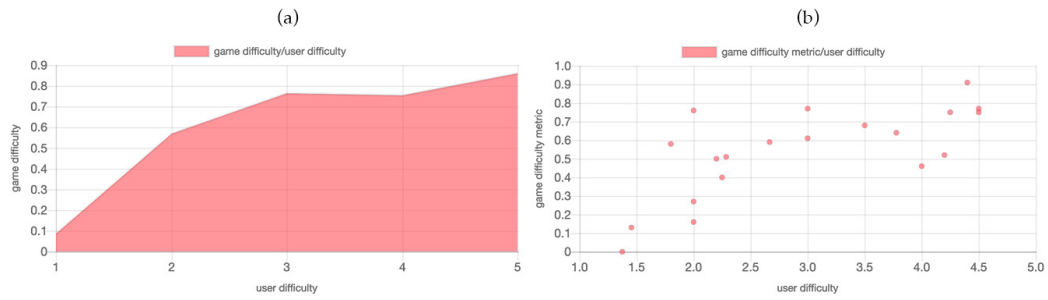


Figure 6.17: These graphs compare the estimated difficulty with the difficulty ratings of the test users. Graph (a) shows the average  $d$  per user rating. Graph (b) shows the average user rating for each game.

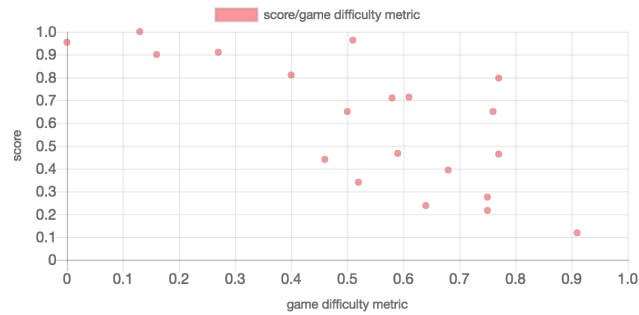


Figure 6.18: The relation between the score achieved by users and the difficulty  $d_{metric}$ .



## 6 Analysis

An interesting relation is shown in Figure 6.19, if a user rated a game as difficult, the fun was rated higher when the user achieved a higher score. This indicates the flow state, where the user perceives a game as challenging but not too hard to become frustrated. Figure 6.20 shows that games with a higher  $P(obstacle)$  were rated as more difficult than games with a lower  $P(obstacle)$ .

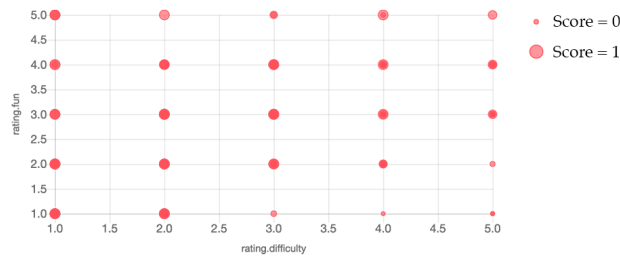


Figure 6.19: The relation between the user's fun rating, the user's difficulty rating and the achieved user score, illustrated as the point radius.

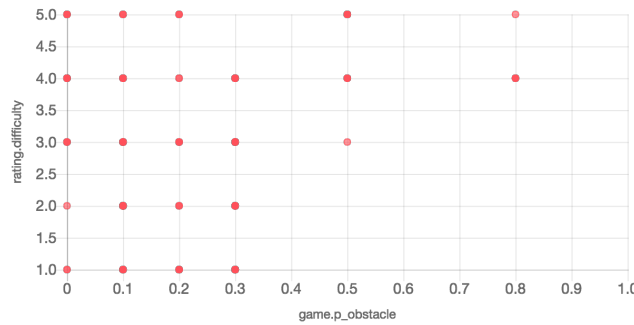


Figure 6.20: The relation between the user's difficulty rating and  $P(obstacle)$ .

Figure 6.21 shows a part of the table from the User Testing Data site. This table can be used to investigate the ratings and scores of the games.

## 6 Analysis






game	game difficulty	game sigma	#ratings	$\mu(\text{fun})$	$\sigma(\text{fun})$	$\mu(\text{difficulty})$	$\sigma(\text{difficulty})$	$\mu(\text{score})$	$\sigma(\text{score})$
2706	0.73	0.17	10	★★★★★ (4)	0.89	★★★★★ (3)	0.77	 (0.71)	0.32
1597	0.61	0.24	7	★★★★★ (3.57)	1.29	★★★★★ (4)	1.07	 (0.44)	0.27
1685	0.85	0.1	6	★★★★★ (3.5)	0.76	★★★★★ (4.5)	0.5	 (0.46)	0.22
2303	0.83	0.1	4	★★★★★ (3.25)	0.43	★★★★★ (4.25)	0.83	 (0.28)	0.03
2818	0.77	0.17	9	★★★★★ (3.11)	0.74	★★★★★ (3.78)	0.42	 (0.24)	0.02

Figure 6.21: A part of the table from the user testing data site, showing the 5 games with the highest fun rating.

## 7 Discussion

### 7.1 Ideal Jump Distance

An observation that emerged from the analysis chapter is the ideal jump distance. The images found in Figure 6.12 and Figure 6.13 (a) show that for both  $v$  and  $l_{block}$  there exists a possible value where a level is the easiest. A logical first assumption would be that all players must jump exactly  $l_{block}$  far. If a player would then attempt to jump over a block of type `BlockType.Hole`, he is only able to make it to the other side if he jumps perfectly at the edge of the previous block. All players jump in the range between  $l_{react}$  and 0 prior to the hole and in order for all of them jump over the hole they must travel at least  $l_{block} + l_{react}$  far. This results in the perfect jump distance described in Equation 7.1, where  $l_{react} = t_{react} * v$ .

$$l_{ideal\_jump} = l_{block} + t_{react} * v \quad (7.1)$$

### 7.2 Dynamic Difficulty Adjustment

Dynamic difficulty adjustment requires knowledge about the player's skill. Since the Line Runner implemented in this work does not allow to measure the player's skill while playing, the methods in this thesis target to increase the difficulty of the game over time, to become more challenging for the user. There are two different approaches covered in this thesis. The first one relies on changing only one parameter during a game session, while the second one relies on changing the whole parameter set. Both approaches enable the game to remove the necessity of having an end position  $S_{max}$ , because the difficulty of the game increases and will eventually lead to an impossible game.

### 7.2.1 Single Parameter Adjustment

Figure 6.11 shows that the difficulty rises and falls when the speed parameter is changed. This circumstance can be exploited to develop a Line Runner with a variable difficulty. This would be also possible by taking another parameter like e.g.  $l_{block}$ , but the sample range possible by using the speed parameter is much higher because it is a floating-point parameter.

In order to get a rising difficulty function that is used for this approach, a minimum speed  $v_{min}$  and a maximum speed  $v_{max}$  is chosen. For this process regression can be used to calculate the difficulty interpolation function, which is then searched for a local minimum and local maximum in the available parameter space for the speed. At this point should be noted that any other arbitrary value can be chosen for  $v_{min}$  and  $v_{max}$ , but using the minimum and the maximum yields in having a maximum flexibility in altering the difficulty. When  $v_{min}$  and  $v_{max}$  are chosen, the game is initialized with  $v = v_{min}$ . Over time the parameter  $v$  is altered towards  $v_{max}$  and the game becomes more difficult.

### 7.2.2 Sequence Of Different Level Types

The second method described in this section uses a sequence of levels with different parameters to achieve an increasing difficulty over time. The levels can be randomly chosen and must be sorted by difficulty, thus  $d_{n-1} < d_n$  must hold for every level in the sequence.

The disadvantage of this approach is, that every time the parameters change, the player needs to adapt to the new mechanics of the level. This adaptation poses a challenge even to experienced players of the game and therefore makes the game even more difficult.

## 7.3 The Perfect Game

Rather than formulating a formula for generating the perfect game, this thesis provides recommendations in order to find parameter configurations for the game Line Runner that promise the best experience. These recommendations are formulated by discussing observations obtained during the user tests of the game.

## 7 Discussion

The first statement received from a user was, that games with more frequent obstacle blocks are more challenging and also more fun. Another user reported that a higher speed yields in the same excitement, which is logical since a too small player speed parameter results in a stale game experience. These statements indicate the perfect parameter values in which users can reach the flow state. The same principle applies to all other parameters which have to be adjusted accordingly to the three-dimensional parameter analysis as described in section 6.2.

The difficulty of games with the highest fun rating during user testing ranged between 0.61 and 0.87, which means harder levels also make more fun.

DDA additionally allows to keep the player interested in the game by dynamically adjusting the game difficulty over time.

## 8 Future Work

### 8.1 AI Improvements

A further improvement to the AI is the support of different display sizes. For instance on a mobile device the player is able to see less upcoming blocks, which also means that they have less time to react to new appearing obstacles. This effect did not occur in this work, since the all games were tested with the same machine and the same display size.

The foresight of the simulated players can be improved by respecting human reaction time for new obstacles appearing on the screen. Thus, the players see new obstacles with a small delay of  $t_{react}$ . This has also been implemented by Isaksen, Gopstein, and Nealen in the paper “Exploring Game Space Using Survival Analysis.”

### 8.2 Game Experience

The game experience of the Line Runner is very rudimentary and is developed for the purpose of automatic testing. While this implementation serves its purpose in this thesis, there could be invested a lot more effort into the enjoyment factor of the game. In addition to the normal mechanics the game could be augmented by collectible items or opponents that hinder the player from winning the game. This would enhance the game experience and lead to an overall more interesting game concept.

A further improvement to optimize the game experience is Dynamic Difficulty Adjustment.

### **8.3 Difficulty Calculation**

The paper (Isaksen, Gopstein, & Nealen, 2015) takes a different approach to the difficulty estimation of a game. Implementing this approach in this work would allow for a direct comparison of both methods. This could be used to show differences of the estimated difficulties and compare the efficiency of both methods using the users ratings.

### **8.4 User Skills Based Dynamic Difficulty Adjustment**

The approach of dynamic difficulty adjustment in this thesis disregards the user skill. As the paper (Aponte, Levieux, & Natkin, 2011) depicts, the difficulty of a level should be based on a player's ability to solve the challenges contained in the level. Otherwise, the lack of positive experience will result in frustration of the user. For this purpose a method to measure the user's score while playing is needed. A simple attempt to achieve this for this Line Runner is to add items to the game the player has to collect. The player's skill can then be estimated by the amount of items he was able to collect.

## 9 Conclusion

This thesis presents various improvements to the automated difficulty estimation of a parameterized version of the Line Runner from (Steurer, 2018) and validates the results with a user test.

To increase the amount of playable games generated by the Level Generator, this work implements two improvements: Every Second Flat and Smart Level Generation. The analysis shows that these two improvements dropped the amount of impossible games generated with  $\sigma_d = 0$  and  $d > 0$  from 78.8% to only 34.78%, confirming their efficiency.

A WebGL version of the game, that allows to randomly generate games, modify parameters and play the games, was created. A second WebGL version was used to conduct a user study and gather difficulty ratings from human playtesters.

Graphs, created using data of random games, show the impact of parameters on the game's difficulty and allow to understand relations of parameters. Three dimensional analysis gives the opportunity to find games that are likely to achieve a flow state. This state is desired to be achieved in order to keep the game interesting for the users.

One dimensional sampling falsifies the assumption that increasing the player speed also increases a game's difficulty.

Using polynomial regression and the analysis from one dimensional sampling, parameter prediction allows to alter the value of a parameter in order to get a certain difficulty and tune a previously impossible game to a playable one.

Using the data from the analysis and polynomial regression, dynamic difficulty adjustment can be implemented. Two different approaches to DDA are discussed in this work: single parameter adjustment and sequence of levels.

A user test is done in order to verify the predicted difficulties. The test users were asked to play previously selected games and rate the difficulty and fun on a scale from 1 to 5. The results of the user test show the efficiency



## 9 Conclusion

of this work's difficulty prediction. The user ratings are in accordance with the previously estimated game difficulties.

Using the information obtained during the user tests and the data analysis this thesis recommends keeping the parameters  $P(\textit{obstacle})$  and  $v$  relatively high and use DDA to keep the game interesting.

Future work includes further improvements to the AI, the Line Runner game and the game difficulty calculation.

The results of this work demonstrate the advantages of automatic playtesting, allowing game designers to achieve the desired game feel and difficulty without extensive user testing. It also provides an easy way to model dynamic difficulty adjustment for endless runner games.

# Appendix

# Bibliography

- Aponte, M.-V., Levieux, G., & Natkin, S. (2011). Measuring the level of difficulty in single player video games. *Entertainment Computing*, 2(4), 205–213.
- Csikszentmihalyi, M. (2009). *Flow*. Harper Collins.
- Csikszentmihalyi, M. & Csikszentmihalyi, I. S. (2000). *Optimal experience*. Cambridge University Press.
- Holmgard, C., Green, M. C., Liapis, A., & Togelius, J. (2018). Automated playtesting with procedural personas with evolved heuristics. *IEEE Transactions on Games*.
- Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 acm sigchi international conference on advances in computer entertainment technology* (pp. 429–433). ACM.
- Hunicke, R. & Chapman, V. (2004). Ai for dynamic difficulty adjustment in games.
- Isaksen, A. (2017). *Score distribution analysis, artificial intelligence, and player modeling for quantitative game design* (Doctoral dissertation, Polytechnic Institute of New York University).
- Isaksen, A., Gopstein, D., Togelius, J., & Nealen, A. (2015). Discovering unique game variants. In *Computational creativity and games workshop at the 2015 international conference on computational creativity*.
- Isaksen, A., Gopstein, D., Togelius, J., & Nealen, A. (2018). Exploring game space of minimal action games via parameter tuning and survival analysis. *IEEE Transactions on Games*, 10(2).
- Isaksen, A., Gopstein, D., & Nealen, A. (2015). Exploring game space using survival analysis. In *Fdg*.
- Isaksen, A. & Nealen, A. (2015). Comparing player skill, game variants, and learning rates using survival analysis. In *Eleventh artificial intelligence and interactive digital entertainment conference*.
- Steurer, M. (2018). *Automatic gametesting* (Bachelor's Thesis, Technical University of Graz).

## Bibliography

- Tan, C. H., Tan, K. C., & Tay, A. (2011). Dynamic game difficulty scaling using adaptive behavior-based ai. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(4), 289–301.