Master Thesis

AI Game Design Generation and Evaluation for 3D Platformer Games

Coen Hacking

Master Thesis DKE-20-14

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Data Science for Decision Making
at the Department of Data Science and Knowledge Engineering
of the Maastricht University

Thesis Committee:

Dr. C. Browne
Dr. M. Stephenson
Prof. dr. M. Winands

Maastricht University
Faculty of Science and Engineering
Department of Data Science and Knowledge Engineering

March 11, 2020 (v1.0)

Abstract

In recent years video games have become larger in the amount of unique content that they contain (e.g. levels, characters, ...). Creating this content can often be expensive and time-consuming. PCG (Procedural Content Generation), which is a set of methods that can be utilised to create content with little or no human interaction, could be a solution to this problem. In this thesis we propose a PCG method that can generate new 3D platformer game levels, without having human designed levels from previous games to learn from. We evaluate that these levels are playable and calculate a variety of metrics to evaluate how interesting these levels are. These metrics are then used to create a diverse set of levels, those levels are later used in a survey to establish how interesting levels are and thus how interesting their characteristics are based on these metrics. Using this information we construct a fitness function whose output is how interesting any given level is. Finally we conclude that we can effectively create levels with significantly different 'interestingness'. We also show that the metrics from levels have some impact on user ratings, but users' behaviour explains much more about why certain levels may be more interesting then others.

Chapter 1

Introduction

As video games have become larger in content over the years. That content has become more expensive and time-consuming to make, due to the demand for longer games with higher graphical fidelity [1]. Large games are therefore a big investment and thus also come with a certain risk with respect to 'Return on Investment'. Therefore, there are a few options for developing games that can reduce the risks.

One possibility—with a lower risk—for game developers is to make smaller games using free-to-play principles, repetitive gameplay and sometimes gambling tactics to keep players invested [2]. In this case, functional PCG is used to provide users with sometimes endless challenges. PCG (Procedural Content Generation) is a set methods that algorithmically generate (part of) the content of a game. These could be functional or cosmetic aspects of the game [3]. Functional content could be puzzles or mazes, while cosmetic content could be landscapes (with mountains and rivers) or any arbitrary object.

Another possibility is to let users make their own and each others game content in 'sandbox-style' games (e.g. *Dreams*, *Little Big Planet*, *Super Mario Maker*). While for these games PCG plays less big of a role, it can be important to evaluate the interestingness of the content that users add to the game (e.g. finding the most interesting levels designed by other players). This approach has recently become very popular and brings an element of creativity that is still difficult to achieve algorithmically.

Lastly, it's possible to develop a large high-quality video game. However, to reduce the costs and risks of making such a game, procedural content generation can help. E.g. hydraulic erosion simulation can help construct a rich authentic landscape with mountains and rivers [4] and being able to generate arbitrary objects without having to go far back into the design process. Other examples are Genetica by Spiral Graphics or Substance Designer. They make it possible to save a texture maps as a function tree of operations that lead up to the resulting texture. This allows us to easily change the the parameters or even change random seeds.

The trade-off is that a publisher either spends a lot of money to make a small set of high-quality levels or spends less money on simple generators that generate less intuitive levels. Therefore, it would be interesting to know how games could be created using PCG and make verifiably interesting gameplay.

1.1 Open Issues

As will be further discussed in Section 2, most of the PCG algorithms have a learning part. Some learn from a large set of human-made levels [5] and others learn from the players experience through passive (e.g. movement in a level) [6] and active (e.g. ratings) feedback [7].

When the algorithm learns from previously designed levels, the pitfalls could be that the levels will look (too) similar to its source (e.g. this becomes clear when looking at *Super Mario Maker* where users often do something different entirely from what is considered to be *Mario* level even though the building blocks are (mostly) the same [8]). But an even bigger issue is that we don't know whether the algorithm is optimising for interesting gameplay or optimising for levels that just match the ideals of the source content. The assumption is that the source content is interesting.

When learning from player experiences the challenges are about what metrics to take into account to establish a fitness function for what is interesting (i.e. how to properly quantify a video game level). In 2D games there are already a lot of factors the play a role in the 'interestingness' and in 3D games there are even more (e.g. camera behaviours) [9].

What we want to investigate is if it's possible to intelligently generate 3D game levels that are interesting to human players. For the game that we'll be generating levels for, we assume that a set of rules and the characteristics are defined. The idea behind this is that we still have some of the creative control—through the components we introduce and the parameters of our search algorithm—and also have some of the advantages of automation.

1.2 Objective

In this thesis we want find out (1) how we can use PCG in 3D space without keeping a grid-based structure, (2) how we can help the (human) player navigate in an unknown 3D environment without disclosing where the goal is, (3) how we can identify what aspects of levels are important to overall ’interestingness’ of a level and (4) if we can use continuous human feedback to algorithmically improve the results.

For this thesis we will choose our scope to be 3D ’platformer’ games. We define a 3D platformer game as one where the character is controlled in the 3rd person perspective, where the player can move around on a horizontal plane while being able to jump vertically and where the objective is to avoid obstacles across a level until a certain goal is reached. For this type of game, we want to define a model for levels and use PCG to generate levels with platforms and obstacles.

Given these level definitions we want to guide the player through them. In 2D games it’s usually easier to define where to go (e.g. many 2D platformer games like *Super Mario Bros.* move from left to right). However, in 3D platformer games the path may not always be straight (linearly) ahead. To solve this issue the game needs to communicate a path to the player (i.e. not knowing where to go, may be boring).

After having defined and generated a large amount of playable level data, we want to define metrics that may distinguish them. These can then be used to generate a sample pool of playable levels, where we can optimise each level to have a large distance in metrics-space compared to other levels in the sample pool (i.e. each level is as unique and different as possible). Having these levels tested by human players provides us with player data (e.g. including ratings) that can help us establish a baseline of what levels are interesting. This data can then again be used to construct a fitness function, for all or clusters of human players.

If we can find a proper fitness function that maps from the features of the level (including the metrics) to the (weighted) average/median rating human players, we can use this to continuously find better level data and improve the human players' experience.

1.3 Research Questions

The research question for this thesis is: how can we automatically generate 3D game levels that are interesting for human players?

1. *LevelDefinition*: How can we define game levels in 3D space?
2. *PlayerCommunication*: How can actions—that the player has to perform towards the goal—be optimally communicated to the player?
3. *EvaluationMetrics*: What are good methods to evaluate such the 'interestingness'? And which metrics are the most important?
4. *FeedbackLoop*: Can human interaction feedback be used to improve the generated content?

Chapter 2

Background

There have been many approaches towards creating interesting games using various methods of PCG. We distinguish a number of different dimensions in PCG. Some approaches try to replicate human designed levels [7, 10], while others try to generate levels from scratch [11] and sometimes levels are generated based on a flexible rule set [12]. Some methods are mainly focused on being functional (e.g. generating puzzles and challenges) [7, 12, 11], though other methods are about cosmetic generation (e.g. texturing, shading, animation) [13, 14]. Most papers that describe functional PCG, generate 2D grid-based levels, though some take a step towards creating 3D grid based levels [15]. Many cosmetic papers discus 3D generation; this is probably due to how labour intensive designing 3D games can be. One final aspect of PCG is whether or not to use human feedback for parameter tuning of the generation process.

From all papers related to 'Mario AI Championship' [7, 10] or that otherwise try to use PCG for generating *Super Mario Bros.* levels, there are several types of approaches. Earlier work mostly focuses on manual analysis and building a model from the identified structures [16]. E.g. S. Dahlskog tries to identify patterns by analysing and classifying vertical slices from world 1-1 of the original *Super Mario Bros.* Dahlskog continues his work on this and defines patterns of subsequent vertical slices as interesting patterns that are then used for the evaluation in a fitness function [17, 18]. The assumption that Dahlskog makes here is that having more patterns of subsequent vertical slices entails more interesting levels.

Later we start to see some machine learning in this field. Dahlskog focuses on finding logical orders for all distinct possible layouts through Markov chains [19]. This finally brings a higher level of automation and reliability to the generation of *Mario* levels. The Markov model is created from *n* prior vertical slices. With low values for *n* the outputs of the model ”produce a haphazard mess”, while for large values of *n* the levels are very similar to the ”corpus” (source content).

A. Summerville proposes a similar approach to Dahlskog, however he doesn’t use the same representation of the level data [20]. Summerville uses a representation where each tile is represented by a number. What makes it more interesting is that Summerville uses MCTS for verifying playability and the method proposed also allows for making it possible to tweak parameters (e.g. how many coins, gaps, ...).

V. Volz takes the idea of a convolutional GAN (Generative Adversarial Network) and learns with two different phases [5]. In the first phase, the discriminator learns from the map of the level, where each tile is represented by a unique number, while the generator is given a Gaussian noise input. In the second phase, CMAES (Covariance Matrix Adaptation Evolutionary Strategy) is used to create levels with specific design properties (e.g. number of ground tiles). This method also allow for creating levels with increasing difficulty, unlike the previously discussed methods.

The last *Mario*-related paper we discuss aims to identify ”High Interaction Areas” from gameplay videos, with the assumption that an area where a player spends more (non-idle) time is a more interesting area (i.e. more engagement entails it’s more interesting) [6]. This method allows not only to take into account tile data, but also allows for identifying how the player interacts with each tile.

Furthermore, in a more general approach, it been investigated how certain level structure can lead to interesting gameplay [21]. This shows that there are various structure levels at which we can look at gameplay. E.g. coming back to *Super Mario Bros.* at the highest level we have the game level that focuses on teaching one strategy for beating the level. Then one level lower, there are increasingly difficult sections. Until at some point we reach the lowest level which consists of blocks/tiles. This example doesn’t only apply to *Mario*, but a wide variety of games.

There are also approaches that try to model games by introducing a language [12]. Such approaches allow for generating more original content. These approaches focus more on letting humans decide on what rules a game should consist of (e.g. the tiles that a map consist of, the gravity of the world, ...) and not so much on how to use it to let AI generate and evaluate games. The VGDL (Video Game Description Language) is grid-based and every tile receives a value of what object is in that position. Every object has a set of rules and visuals describing it. Using the VGDL several classic 2D games have described (e.g. Lunar Lander, The Legend of Zelda).

For generating general game levels there is some research in the form of ANGELINA [11, 22]. There are various iteration of this program, some of which included a limited form of creative content generation and others that are partially 3D, as 3D projections of what are ultimately 2D grid-based levels [15].

Chapter 3

Methodology

In this chapter we discuss our proposed method, including aspects of level definition, generation, evaluation, verification/testing and adjustment. An overview of the process can be seen in Figure 3.1. All steps the process in this figure are further discussed in detail.

3.1 Level Generation

As discussed in Section 2 there are two categories of level generation. However, with machine learning approaches there are several issues when used for creating 'new' games. The 'Super Mario Bros.' series has many examples to learn from, because by now there have been released many games for it. New games with 'novel' mechanics don't have that. However, levels for such games may be generated according to certain design rules and principles. Besides the rules of the defined objects, the atomic behaviours can be combined in various ways. The level generation is thus essentially random, however with certain rules and parameters to tighten the scope.

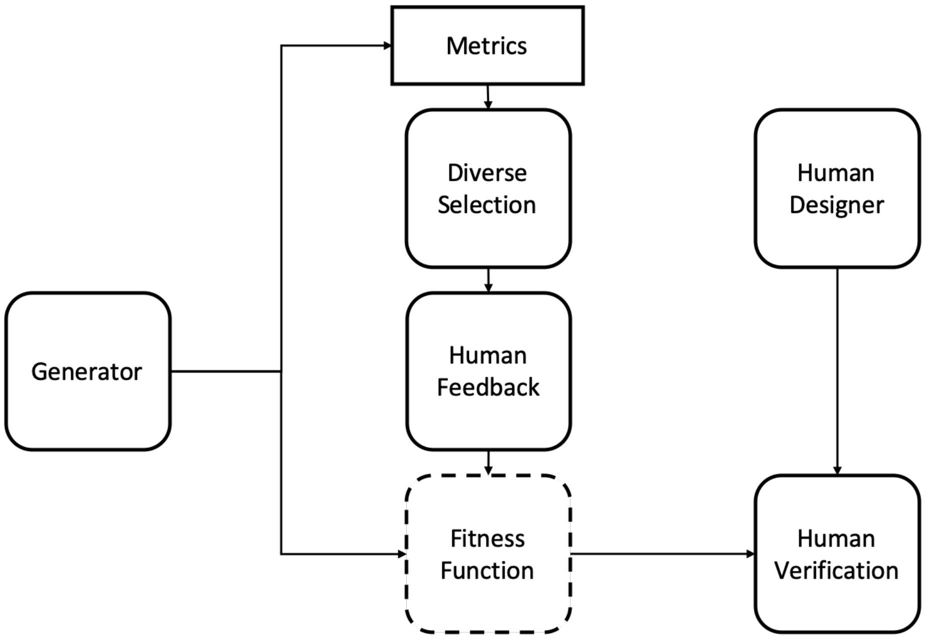

Figure 3.1: An overview of the method

3.1.1 Gameplay Definition

We intend to define as accurately as possible what the game's rules are, as well as the set of available gameplay elements (e.g. the ?-block in Super Mario Bros.). Elements can be defined with a logical component to it, as behaviour of objects can be designed as a combination of atomic behaviours. For example, a ground platform provides the logic for the player to able to stand on it; this can be combined with moving logic, rotating logic or even both. While other combinations may be contradictory (e.g. a enemy that makes the player earn points upon touching it, while also decreasing the health of the player). The game levels that we want to generate will obviously contain other gameplay elements to make the levels more interesting (i.e. with just platforms we would only generate simple mazes).

3.1.2 Environment

In general we define the environment as elements that don't change their state during the level (i.e. they are static).

Camera and Camera Zones The camera is a virtual device that captures and displays the game world to the player, such that the player can interact with the game and can make correct decisions on what actions to perform next in the game world. Camera zones are areas in the game where the camera should differ from it's default following behaviour, adjusting it's perspective to better communicate what the game wants the player to do (E.g. showing that a moving platform is on its way back to the player, even though with the default camera behaviour it would have been too far away to see).

Platforms Platforms are defined as a flat 2D shapes that are projected on the XZ plane (where positive Y points toward the sky), extruded downwards and translated somewhere into the 3D space. The player is able to stand on these and consecutive platforms provide a path towards the goal.

Goal The goal is a platform that is *n* platforms away from the starting platform of the player. Touching the goal platform will finish the level.

3.1.3 Entities

In general we define entities as elements that change their state during the level (i.e. they are dynamic).

Checkpoints In many games there are various checkpoints throughout the game. Usually at the end of a level, but also often at various locations in a level through a checkpoint. Upon touching a checkpoint the player's fallback will be reset (i.e. the player will return to the latest activated checkpoint). This is ensure that the player won't have to redo (all) previous challenges.

Moving Platforms In our implementation a moving platform moves between two points A and B where it at point A it connects to a prior platform and at point B it connects to the next platform.

Teleporters In most games there are doors/teleporters that translate the player to a new area. In our implementation every teleporter has a counterpart and is thus bidirectional.

Hidden Platforms Hidden platforms are platforms that are hidden until the player is in close proximity to them.

3.1.4 Player Communication

An important part of the level generation is how the camera will be moving, because this plays a major role in communication to the player. The camera's intent is to show the following action of the player. For example by centering the next platform or enemy to jump on.

Another way to show the player how to continue is to show what the AI player did to continue. Though there are possible complications, due to the fact that levels might not be linear (i.e. there may be more than one next goal). However, using a simple A* search we were able to guide the player through the levels, each time showing the human player his/her next objective (i.e. a key, a teleporter or the goal).

3.1.5 Language and Encoding

As discussed in Section 2 there are already some languages for designing video games. However, they don't translate well beyond a 2D grid based game. 3D games are in most cases represented by triangulated meshes in continuous space [1], which will be further described in Section 3.1.6.

To be reproducible and playable as part of our survey, we need to unambiguously encode the levels' data. Encoding is also important for evaluation and inspection by hand. We chose to encode the data of each level as a set of platforms. Each of which contains a type, position, rotation and mesh data. This ensures that everyone gets precisely the same level as when it was saved.

Something that we also explored is the possibility to encode the level as graph denoting every platform as a vertex and each connection as a directed edge UV (i.e. the player can from platform U to V). Though this is more ambiguous, it allows for easier manipulation. The algorithm would be able to generate a complex graph and from that it can be calculated where platforms with a certain size and shape should be. However, this is much more difficult because the graph can have contradictions that aren't be possible in euclidean space.

[1] as precise as a continuous space can be using floating-point numbers

3.1.6 PCG Algorithm

We first construct a path the goal while randomly picking interesting (dynamic) platforms to challenge the player (e.g. moving or rotating platforms). The first platform is always static to ensure that the player can study the surroundings without dying. After the main path is generated we generated alternative paths that may lead to the goal or a dead end. Ideally paths with a dead end would have some kind of reward, though this is currently not included in the implementation.

In order to generate platforms that are form a path, we must generate them in such a way that they are close to enough to each other. We can do that by calculating the maximally feasible jump arc that a player can make (i.e. platforms can be further away when the next platform is lower). This due to the physics in the game. The upwards velocity is defined by $v_i = v_{i-1} - a$, where a is the gravity. v_0 is the strength of the jump. The horizontal movement speed is a constant [2], due to the fact the force of movement and the force of the air friction are the same at the maximum speed m. Now that is laid down, we can find the direct formula for the arc.

Vertical speed will be $v_y = v_{y,0} - a * t$, if integrated with respect to time we get $y = v_0 t - \frac{1}{2}at^2$. Horizontal speed is $v_x = m$, if we integrate with respect to time we get $x = m * t$. Hence if we substitute, we get equation (3.1).

$$y = v_0 \left(\frac{x}{m}\right) - \frac{1}{2}a\left(\frac{x}{m}\right)^2 \tag{3.1}$$

In Algorithm 1 we show we put the basic concepts of PCG and physics together into a simple algorithm. Here we only show how to create the path and the branches, excluding the details about different functionality that each platform might have (e.g. moving).

[2]as long as we move forward without changing the direction of horizontal input

```
Data: Level size $N$ and number of branches $B$
Result: The set $S$ of platform locations
Let $P$ be a vector with value $\{0,0,0\}$;
Let $S$ be an empty set;
Let $n = N$;
for $i = 0$, $i$++, while $i <= B$ do
    if $S > 0$ then
        Let $r$ be a random number from the uniform distribution $[1, N]$;
        $n$ += $r$;
    end
    while $|S| < n$ do
        Let $P'$ be a new location at one of the maximally most reachable
          arcs from $P$;
        Let $S'$ be a new platform containing $P'$;
        if no collision between a platform in $S$ and $S'$ then
            Add $S'$ to $S$;
            Let $P$ = $P'$;
        end
    end
end
```

Algorithm 1: A simplified version of the generation

3.2 Evaluation

When doing evaluation we want to minimise the amount of human intervention. At first we want a perfect (A*) AI player to verify that the level's goal can be reached. We looked into having more realistic (machine) 'learning' AI players to get statistics on the levels (e.g. learning curve, difficulty, etc.) and use a heuristic to define the 'interestingness'. For this we tried using an AI player that learns to play a game level (e.g. Deep Q-learning [23], NEAT [24]). Unfortunately, there were issues with using these approaches, (1) they weren't representative of a human player and (2) they don't function as well in 3D as they do in 2D (due to the added complexity). For evaluation, the program should be able to create levels with the following criteria [25]:

1. Minimum Travel, the minimum amount of platforms that the player has to touch in order to get to the goal (as calculated by A* search).
2. Main Path Travel, the amount of platforms that the level has on its main path from start to the goal.

3. Ratio Moving Platforms, *M/P*, where *M* is the number of moving platforms and *P* is the total number of platforms.

```
return levelData.platforms.Where(p => p.GetType() ==
    typeof(MovingPlatform)).Count() / (float)
    levelData.platforms.Count;
```

4. Ratio Hidden Platforms, *H/P*, where *H* is the number of hidden platforms and *P* is the total number of platforms.

```
return levelData.platforms.Where(p => p.GetType() ==
    typeof(HiddenPlatform)).Count() / (float)
    levelData.platforms.Count;
```

5. Forgivingness, *C/P*, where *C* is the number of checkpoint platforms and *P* is the total number of platforms. As the ratio of checkpoints is lower, the level is less forgiving (i.e. sending the player back to the beginning each time, when a mistake is made).

```
return levelData.platforms.Where(p => p.GetType() ==
    typeof(CheckpointPlatform)).Count() / (float)
    levelData.platforms.Count;
```

6. Ratio Door Platforms, *D/P*, where *D* is the number of teleporter/door platforms and *P* is the total number of platforms.

```
return levelData.platforms.Where(p => p.GetType() ==
    typeof(DoorPlatform)).Count() / (float)levelData.
    platforms.Count;
```

7. Optimal Coverage, *O/M*, where *O* is the number of platforms on the optimal path and *M* is the number of platforms on the main path (i.e. the path as it was procedural generated from the starting platform to the goal platform).

```
return (float)levelData.pathLength / levelData.
    platforms.Count;
```

8. Margin of Error, a value that is used to create levels by. A value of 0 mean every jump has to be exactly perfect according to equation (3.1). As the margin of error increases the game becomes easier (i.e. the platforms are placed more closely and are therefore easier to reach).

9. Average Time to Complete, an estimate value of how long it would take a human player to complete the level. Speed is not taken into account into this formula, because the speed of the player is the same in each level. We assume that the level takes longer if the linearity is lower (i.e. because the player requires less time to search), therefore for every three vertices U, V and W, we multiply by $UV \cdot UW$ and add them all together. A low value for this sum entails high linearity.

```
return OptimalCoverage * AveragePlatformSize * Mathf.
    Abs(levelData.nonlinearity);
```

10. Average Outgoing Edges, the average number of reachable platforms from a certain platform.

```
Graph g = Graph.MakeGraph(levelData.platforms);
return g.Edges.Average(v => v.Count);
```

11. Average Platform Size, the average size that a platform has. Generally, we expect levels with larger platforms to be easier, due to less risk of (overshoot) falling.

```
return levelData.platforms.Average(p => p.SurfaceArea
    );
```

3.2.1 High-Level A* with Perfect Information

The A* method that we propose for evaluation will work on two levels of detail. On the highest level we will be planning the route, where we represent all the game level's platform as nodes in a directed graph. We represent the possible transitions as edges in a graph.

For high-level A* we simply consider all platforms to be vertices in our directed graph and let the edges (U, V) denote that a vertex V is accessible from vertex U. Considering the directed graph we calculate the path to the goal G. If a door D is on that path we calculate the A* path from the starting position S to the key's position K and concatenate with the calculated A* path from K to D. Then we set S to D and redo this process from the new S to G. Finally we concatenate all sub-paths we found.

To be certain of optimality $d(S, K) + d(K, D)$ should be smaller then an alternative path that doesn't go through D.

Using this algorithm we can measure the following properties:

1. Length of the shortest path
2. Singularity (i.e. is there only one path to the goal)
3. Coverage, the ratio of the number of platforms that were visited divided by the total number of platforms
4. Visits, the amount of times any given platform in the graph has been visited

3.3 Selection

To be certain that we have a diverse selection of levels for establishing our fitness function, we looked at several clustering approaches. This is useful because methods like k-means have the property that their centroids spread diversely over the data. Our data is in this case the metrics (as described in Section 3.2) on the population of the levels that we've generated. As our data is mostly uniformly/normally distributed—because it's also generated mostly uniformly (except for disregarded levels)—k-means is viable option. Another advantage of this approach is that we can easily control the number of clusters. Selecting too few levels has the disadvantage that we might miss certain interesting levels and less certain statistics due to a small sample size. Selecting too many levels has the disadvantage that players won't play certain levels enough, reducing confidence in the measured ratings. We decided to choose $k = 30$ which is just enough to be significant and not too large for people to stop, halfway through the survey. In Section 4.3 we'll show how many results we have per level and how significant that is, the significance is shown in Table 4.1.

3.4 Human Feedback

With the selection of unique levels that we've made, we can start the survey. This survey serves as a way to establish the baseline of what human players find interesting. For this, we collect active (e.g. ratings) and passive (e.g. movement in a level) feedback data. All other details including results will be discussed in Chapter 4.

3.5 Fitness Function

With the combined information of both the level data and the player data we can start to calculate a fitness function. We evaluate the quality of the fitness function(s) with the coefficient of determination

$$R^2 = 1 - \frac{\sigma_{res}^2}{\sigma_{tot}^2}$$

where σ_{res}^2 is the residual (unexplained) variance from our input X (i.e. the metrics based on level data) to our output Y (i.e. the rating according to player data) and σ_{tot}^2 is the total variance of our output Y. If the data allows it, we will end up with high value for R^2. But, it's very possible to have different fitness functions for different groups of people (i.e. humans players can be clustered based on demographics, skill and their opinion). The quality of the resulting fitness function as part of our survey is discussed in Section 4.3.6.

Chapter 4

Experiments

With the survey experiment—as can be seen in the screenshot in Figure 4.1—we want to establish which of the collected level data is important for the selection of interesting levels (i.e. what entails a good level), with introducing as little bias as possible. To do this we let the generator explore the parameter space and keep $n = 30$ levels that have a high distance to each other in the search space (i.e. levels that are as different as possible). The selection of levels is done according to the method described in Section 3.3. These levels are placed in a survey, in random order, to human players which can rate the levels. The random order is important to reduce any psychological bias (e.g. rating compared to prior levels, rating with no prior playing experience). The user data combined with inherent data is used to predict the ground truth 'interestingness' for any given level, using various (regression) machine learning technique. This will be our fitness function. For the experiment we collect:

1. The path through the level (i.e. recording the time since the start of the level and the location vector, until the goal is reached). This data entails:
   (a) Number of deaths
   (b) Time to reach the goal (or give up)
   (c) Whether it's a win or a loss
   (d) Coverage (i.e. how much of the level has been explored p/P where p is the number of platforms the player has reached and P is the total number of platforms in the level)
   (e) Idle time (i.e. time the player doesn't progress)

(f) Fallback (i.e. how much a player has fallen back, calculated as

$$\sum_{i=0}^{n} d_i - r_i$$

where n is the number of deaths, d_i is the path to the goal at platform last touched before death and r_i is the path to the goal at platform where the player respawned)

2. A rating for each level from 1 to 6, this forces people not to be neutral (i.e. they can choose 3 (slightly negative) or 4 (slightly positive))
3. Age, as this is generally used to target an audience
4. Gender, as this is generally used to target an audience
5. Email (for contact about the prize (i.e. incentive to do the survey))
6. Unique Device ID, to ensure that data belongs to one individual
7. Device Name
8. Device Type (i.e. one of the following: Mobile, Desktop, Console, Web)
9. Device OS

4.1 Biases

With this survey there are several biases to consider due to the fact that this is a more complicated survey than a simple questionnaire.

1. A player has to get used to the game, only after player x levels a player has enough frame of reference to give an accurate score. For this we have chosen $x = 3$.
2. Different environments, even though OS and device name are measured there can be differences in environment of someone taking the survey. In a normal survey this impact may be minimal, however these surveys are a lot more complicated (e.g. slow browser or low battery causing a sluggish game).
3. Personal Bias, every person can have a strong personal bias towards what is considered to be interesting gameplay. Especially more experienced players can have a stronger negative bias due to a (naturally) bigger frame of reference.

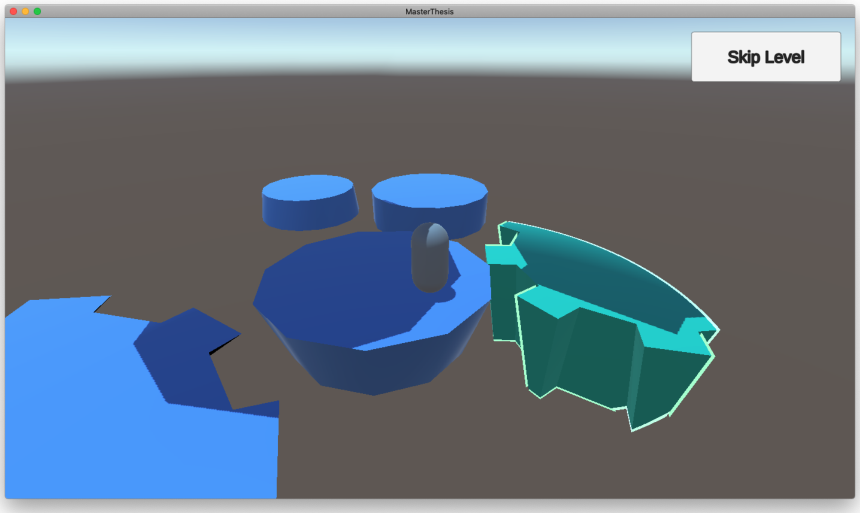

Figure 4.1: The survey with a randomly loaded level from our selection, running on macOS (though also available on Windows, Linux and Web)

4. Too little difference between levels can cause respondents of our survey to not give a clear distinct rating.

We try to improve the results by correcting for these biases. However, as can be seen in Section 4.3, a general fitness function is difficult to achieve due to the described biases. We explore whether we can find a fitness function for certain sets of players.

4.2 Pre-processing

Due to the many biases and noise in the data we carefully did some pre-processing. The steps involved in this process are:

1. Players who played too few levels $n < 5$ likely don't contain any useful data. These are therefore discarded.
2. Levels that were skipped without any attempts of player, measured by number of recordings and 'deaths', are discarded.
3. Players who had no variance in their ratings are discarded (e.g. a player who rates every level with score of 1).

4.3 Results

For the survey we had 547 results divided over 30 levels. Each level was played 18.23 times on average with a minimum of 12 and a maximum of 23. For each level we calculated the average rating without any bias correction. This can be seen in Figure 4.2, Figure 4.3 and Figure 4.4.

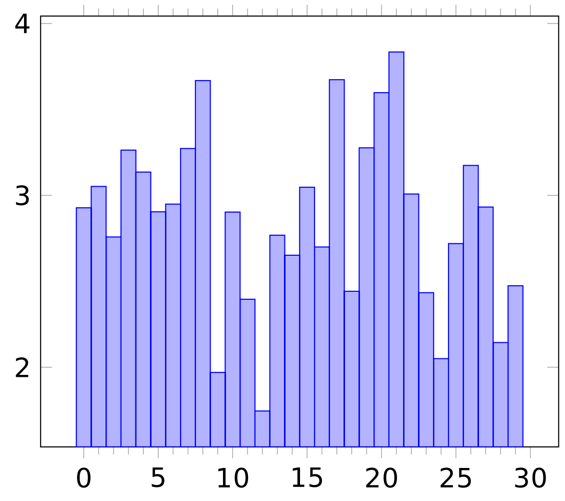

Figure 4.2: Raw Results of the User Ratings per Level

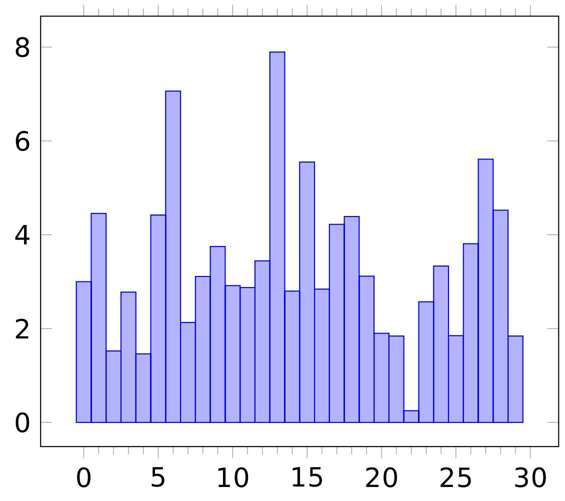

Figure 4.3: Raw Results of the Deaths per Level

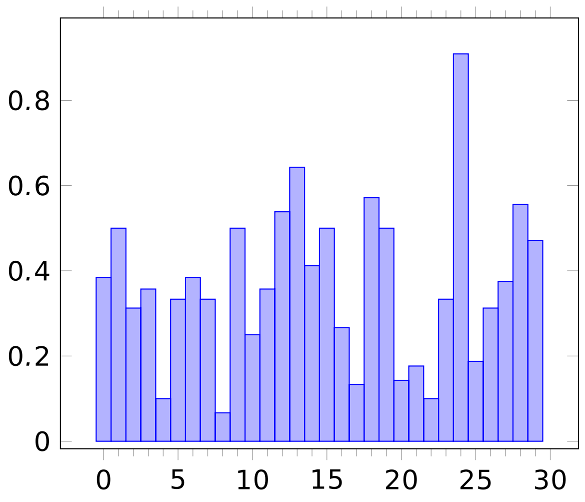

Figure 4.4: The raw percentage of people that stopped a level without reaching the goal

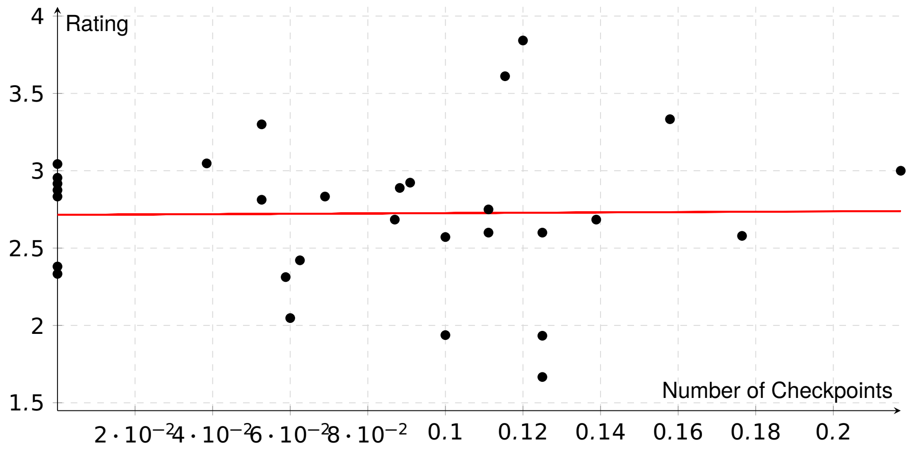

Figure 4.5: The raw correlation between average rating per level and its margin of error

4.3.1 Significance

To understand if the choice of a level selection that we made as described in Section 3.3 was diverse enough, we'll show for each level what the significance is through a T-test. The reason for a T-test rather than a normal test is because our number of respondents per level is bellow $n = 30$ for each level. After reducing noise, we let our DoF (Degrees of Freedom) be equal to the number of serious respondents for that level. For each level we also show the 95% confidence interval with the 2.5% and 97.5% markers. All these results are shown in Table 4.1. Level 4 had the highest upper bound score, but has the lowest DoF. Level 17 is therefore a much better level due to it's highest lower bound score and mean score. That also means that level 17 may be interesting to the largest group of correspondents.

4.3.2 Gender Differences

We tried to find different interests per gender. However as can be seen from Figure 4.6, the survey results from women were limited and this reflects in many discrete numbers.

4.3.3 Heatmaps

To visualise and understand the behaviours of the correspondents we compiled heat maps, as can be see in Figure 4.7. This data tells us where difficult points in the levels are (i.e. where most players die) and this gives us an insight about what we can improve. These are details that aren't directly visible from the predefined metrics, but this manual analysis allows us to define better and/or more metrics. As an example triangle shapes seem to be more difficult and moving platforms with higher speeds are generally more difficult as well.

4.3.4 Textual Feedback

Besides ratings and movement data, we allowed correspondents to give feedback about what they felt was interesting or uninteresting in a level. One of the biggest complaints was that the camera movement wasn't desirable. This is something that wasn't visible in the heatmaps, but could've been simulated, through the movement data. Another complaint was that the jump response was sometimes late at the edge of a platform, causing the player to fall.

Level	2.5%	Mean	97.5%	DoF
0	0.14	0.32	0.49	13
1	0.35	0.53	0.71	18
2	0.37	0.55	0.74	16
3	0.34	0.55	0.76	14
4	0.31	0.58	**0.86**	10
5	0.31	0.46	0.62	15
6	0.12	0.34	0.57	13
7	0.24	0.45	0.67	15
8	0.47	0.64	0.80	15
9	0.10	0.25	0.41	13
10	0.14	0.28	0.42	12
11	0.17	0.35	0.54	14
12	0.01	0.15	0.30	13
13	-0.01	0.16	0.34	14
14	0.21	0.40	0.60	17
15	0.04	0.21	0.39	16
16	0.13	0.33	0.52	15
17	**0.52**	**0.68**	0.83	15
18	0.30	0.49	0.69	14
19	0.11	0.33	0.55	14
20	0.37	0.60	0.83	14
21	0.39	0.58	0.78	18
22	0.26	0.56	0.85	11
23	0.23	0.43	0.63	16
24	-0.06	0.05	0.16	11
25	0.36	0.51	0.66	16
26	0.34	0.51	0.67	16
27	0.28	0.47	0.66	16
28	0.10	0.28	0.47	18
29	0.08	0.22	0.36	17

Table 4.1: Statistical significance for the the weighted normalised ratings. Shown using a 95% confidence interval. Note that a negative values on the lower bound wouldn't be possible in reality as the ratings are normalised to be in the range [0, 1].

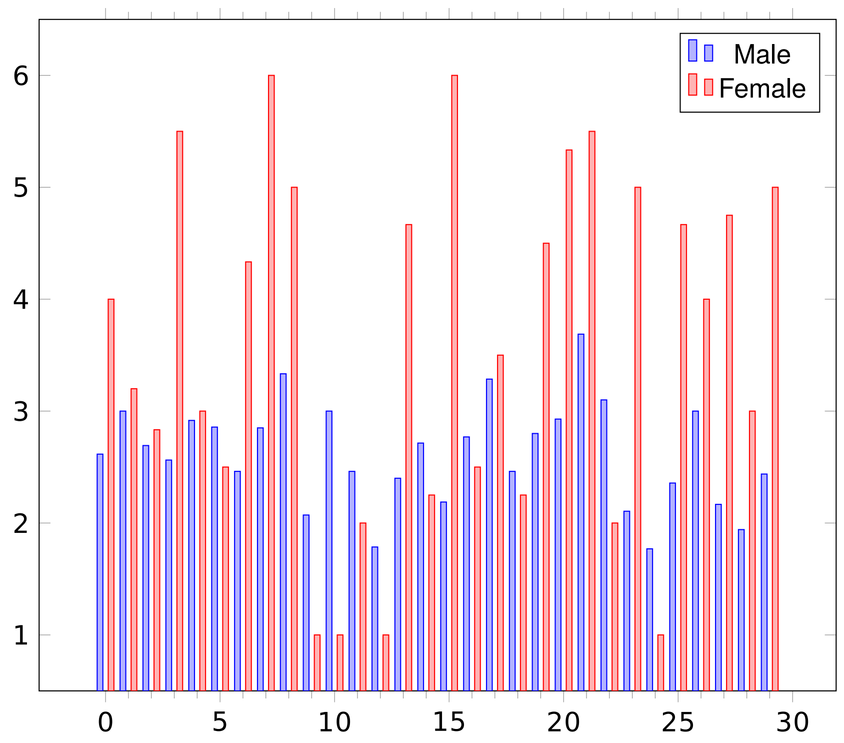

Figure 4.6: Raw Results of the Ratings per Gender per Level

4.3.5 Interesting Metrics

What we are most interested in about our survey results is finding correlation between some of our metrics and the ratings of our correspondents. As can be seen in Figure 4.5, there's very little direct correlation between the metrics and the averaged given ratings.

4.3.6 Fitness Function

From the level data and the ratings we ran a variety of regression algorithms to find a correlation between the metrics—as defined them in Section 3.2—and the ratings as given by our correspondents. Correlations are shown for our best-fitting fitness functions. We evaluated the quality of these fitness functions by R^2. The results are shown in Table 4.2 [1].

[1] the implementations of these algorithms are used from ML.NET 1.5.0-preview and the configurations were modified to improve the results. This was all done in an automated fashion, where the program heuristically tried a variety of configurations.

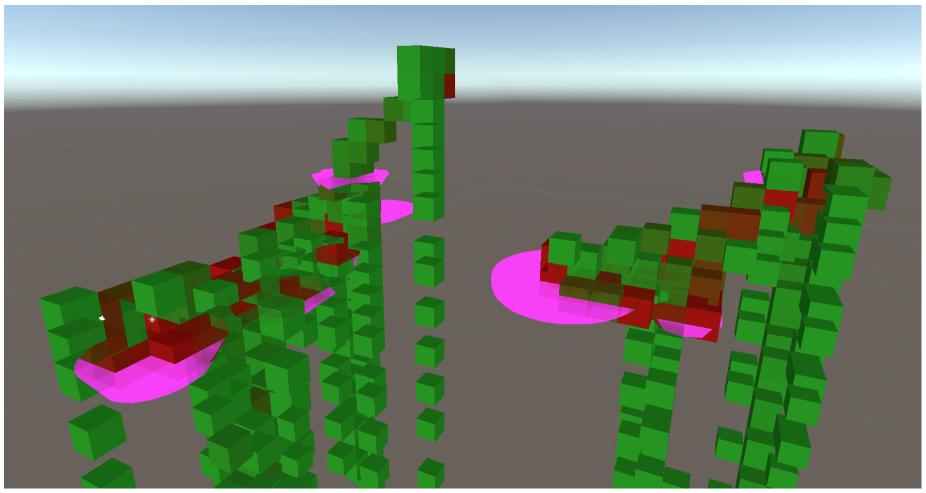

Figure 4.7: An example of what a heatmap looks like for this particular level (from green to red are how many correspondents have entered that cube area). The positions were not interpolated, causing some gaps.

Algorithm	R^2	Loss	Squared-loss	RMS-loss
SdcaRegression	0.1484	0.15	0.02	0.15
FastForestRegression	0.1269	0.14	0.03	0.18
FastTreeRegression	0.1201	0.15	0.02	0.15

Table 4.2: Qualities of the resulting fitness functions without player data

Though the correlations in Table 4.2 are relatively low, there is a stronger correlation possible when we introduce information about the players as a dimension in the data from which we calculate the fitness function. This is however unrealistic, as in reality we'd never have this data at the time of generating new levels. Though it may be possible to simulate human behaviour with an AI agent and collect this data after generating the level using that agent. The quality of the resulting fitness functions with player data input is shown in Table 4.3.

Algorithm	R^2	Loss	Squared-loss	RMS-loss
SdcaRegression	0.4565	0.08	0.01	0.08
FastTreeRegression	0.4111	0.12	0.02	0.15
FastForestRegression	0.3458	0.12	0.02	0.13

Table 4.3: Qualities of the resulting fitness functions with player data

Apart from looking at predicting the average rating, we tried constructing a fitness function from the median ratings, but the quality of these fitness functions were similar to the qualities shows in Table 4.2 and Table 4.3.

Chapter 5

Discussion

As can see from Figure 4.2 there is indeed some variance in how interesting levels are to users based on the average rating that was given. This means that the levels that were selected were indeed also different enough to show some of this variance. However, the correlation between the metrics of the levels are limited. In this chapter we discuss why certain aspects went well and some of the results—mainly the fitness function—aren't quite as significant.

5.1 Limited Level Expression

Currently our implementation only allows for a limited amount of (completely flat) platforms shapes. While more complicated structures like slopes or stairs are not taken into account. Secondly, the number of interesting (dynamic) elements is limited in every level and the variety of different interesting elements is limited as a whole. This also makes later levels less interesting, due to some level of repetition. The lack of diversity reduces the search space when trying to find the optimal generation parameters from rating feedback. This may lead to having less interesting levels as a result.

5.2 Personal Biases

The personal biases heavily influence how levels are enjoyed. However, what can happen in general is that a strong positive preferences of some can cancel out the strong negative preferences of others. That means that on average the rating always ends up somewhere in the middle. Especially because users are already careful due to what we've discussed in Section 5.1. What we could do is look for subsets of correspondents that have similar interests and calculate a fitness function for that group.

5.3 Implementational Issues

As discussed in Section 4.3.4 there are some implementational issues that cause levels to be less interesting than they could otherwise have been. What we could have done to improve upon these points is provide better camera smoothing to get rid of incorrect and abrupt changes. To explain the next point we define a frame as the delta time in which the game state and graphics are updated. For the input lag—that can cause the player not to jump—we can improve physics measurements (currently physics is calculated with discrete time steps, but could be continuous). This removes frames where the player is not exactly on the ground (however hovering slightly—but not visibly-above it). Additionally we can cache the jump input to apply to multiple frames instead of just one.

5.4 Verification

To verify the 'interestingness' of levels and the correctness of the fitness function we could have surveyed human players on a certain amount of levels that were evaluated to be highly interesting in the survey as described in Chapter 4. The human players would also play some levels that were evaluated as highly uninteresting, to be certain that our fitness function works correctly on that end of the spectrum. Afterwards, players would again be asked to rate the level and passive information on how the players moved through the level would then also be collected. All of this information can be used to improve the fitness function and find more viable levels as described in Section 5.5.

5.5 Adjustment and Co-Evolution

After verifying the overall 'interestingness' of levels, it's possible to use the newly found ground truth and use a machine learning approach to predict the 'interestingness' from the input variables (or criteria) as described in Section 3.2. The adjusted evaluation system could then be used to increasingly create better levels in a co-evolutionary fashion. This is similar to current search-based techniques, where they use simple greedy evolutionary algorithms, with a certain content representation and evaluation function, to find the k most interesting levels [26].

5.6 Considerations

We tried looking at what the majority group of players finds interesting. For this we looked at the mode and median scores for each level, however that didn't lead to any significant improvement. We still could've looked at clusters of correspondents; aiming to optimise to either appeal a little to a larger audience or appeal a lot to a smaller audience. This could be an important trade-off. We could also adjust significance of the scores by letting correspondents with higher scores have more weight on the average level correspondents, because correspondents with a higher average rating are obviously also more likely to be interested.

Chapter 6

Conclusion

In this thesis we proposed a method to generate, evaluate and test the interestingness' of 3D platformer video games. It has been investigated how machine learning and other AI methods can help us establish a fitness function for generated levels.

LevelDefinition We have presented a way to describe 3D video game levels with procedurally generated meshes, while being able to describe the generated level data with various metrics.

PlayerCommunication It has been discussed that the camera behaviour can help to improve the focus of the player and help it reach the goal. Camera implementation should be more subtle to avoid annoyances from players, however that requires a lot more research into camera motion estimation.

EvaluationMetrics We've investigated what metrics would impact gameplay and tested this with our survey. Though the outcomes are that the metrics only have a limited influence on the average rating, player behaviour has much impact on ratings. Predicting certain player behaviour would thus help improve our set of metrics.

FeedbackLoop From our survey feedback we can improve the selection of generated levels, by looking at the fitness function from only level data. However, without a realistic AI player that mimics human behaviour we don't have access to player information at generation time (e.g. deaths, coverage).

Finally, though the generated game levels may not be quite as interesting as commercial games, the techniques in this thesis are valuable for developing more interesting games. Using AI to verify the playability and difficulty of levels can be used to minimise the time that is normally required for testing game levels. This is even true if the levels are developed alongside human designers in an evolutionary or CAD-like approach.

Chapter 7

Future Work

Assuming that we want to introduce new gameplay elements, we'd like to know the system should be adapted. A possibility would be to fully retrain the fitness function every time that a new (interesting) element is added. Though this would require a lot of new human ratings each time. Perhaps it would be wiser to decouple the metrics from specific interesting elements and find more structural metrics, because those won't likely change when new elements are added to the game.

The make the generation process more structural, it would be interesting to define level structures in a graph and then generating a level according to that graph. We already discussed this in Section 3.1.5 and though we didn't succeed into developing an implementation for this, it would be very interesting to continue research on.

It may be interesting to start defining a more concise language for mechanics in a level. Currently it's only possible to adjust the amount of elements of a certain type in a level, but it would also be interesting to describe structures of sequential elements. As the number of possible structures and elements increase, the number of parameters for a generating levels will also go up.

Lastly it would be interesting to investigate adjusting the methods of level generation to a CAD-like approach, while keeping the concepts of validation (e.g. the goal can be reached, all score items can be collected) and evaluation (e.g. what rating the level would get, how large the target audience would be). Thus, we'd be integrating automatic and manual (assisted) design into one tool.

Bibliography

[1] “List of most expensive video games to develop,” 2019.

[2] K. Alha, E. Koskinen, J. Paavilainen, J. Hamari, and J. Kinnunen, “Free-to-play games: Professionals’ perspectives,” *Proceedings of nordic DiGRA*, vol. 2014, 2014.

[3] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, “Procedural content generation for games: A survey,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, no. 1, pp. 1–22, 2013.

[4] X. Mei, P. Decaudin, and B.-G. Hu, “Fast hydraulic erosion simulation and visualization on gpu,” in *15th Pacific Conference on Computer Graphics and Applications (PG’07)*, pp. 47–56, IEEE, 2007.

[5] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, “Evolving mario levels in the latent space of a deep convolutional generative adversarial network,” *CoRR*, vol. abs/1805.00728, 2018.

[6] M. Guzdial and M. Riedl, “Toward game level generation from gameplay videos,” *arXiv preprint arXiv:1602.07721*, 2016.

[7] N. Shaker, J. Togelius, G. N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, *et al.*, “The 2010 mario ai championship: Level generation track,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 4, pp. 332–347, 2011.

[8] T. Thompson, “” what is a super mario level anyway?” an argument for non-formalist level generation in super mario bros,” 2016.

[9] M. Hendrikx, S. Meijer, J. Velden, and A. Iosup, “Procedural content generation for games: A survey,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, vol. 9, 02 2013.

[10] J. Togelius, N. Shaker, S. Karakovskiy, and G. N. Yannakakis, “The mario ai championship 2009-2012,” *AI Magazine*, vol. 34, no. 3, pp. 89–92, 2013.

[11] M. Cook, S. Colton, and J. Gow, “The angelina videogame design system—part i,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 2, pp. 192–203, 2016.

[12] T. Schaul, “A video game description language for model-based or interactive learning,” in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, IEEE, 2013.

[13] H.-T. D. Liu and A. Jacobson, “Cubic stylization,” *ACM Transactions on Graphics*, 2019.

[14] S. Starke, H. Zhang, T. Komura, and J. Saito, “Neural state machine for character-scene interactions,” *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–14, 2019.

[15] M. Cook and S. Colton, “Ludus ex machina: Building a 3d game designer that competes alongside humans.,” in *ICCC*, pp. 54–62, 2014.

[16] S. Dahlskog and J. Togelius, “Patterns and procedural content generation,” in *Proceedings of the Workshop on Design Patterns in Games ({{DPG}} 2012), co-located with the Foundations of Digital Games 2012 conference*, 2012.

[17] S. Dahlskog and J. Togelius, “Patterns as objectives for level generation,” in *Proceedings of the Second Workshop on Design Patterns in Games;*, ACM, 2013.

[18] S. Dahlskog and J. Togelius, “A multi-level level generator,” in *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, IEEE, 2014.

[19] S. Dahlskog, J. Togelius, and M. J. Nelson, “Linear levels through n-grams,” in *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, pp. 200–206, ACM, 2014.

[20] A. J. Summerville, S. Philip, and M. Mateas, “Mcmcts pcg 4 smb: Monte carlo tree search to guide platformer level generation,” in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[21] B. Cousins, “Elementary game design,” in *Develop magazine*, vol. 10, Intent Media, 2004.

[22] M. Cook, S. Colton, and J. Gow, "The angelina videogame design system—part ii," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 3, pp. 254–266, 2016.

[23] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, vol. abs/1701.07274, 2017.

[24] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[25] C. Browne, "Lecture 13: Puzzles & procedural content generation," 2018.

[26] J. Togelius and N. Shaker, "The search-based approach," in *Procedural Content Generation in Games*, pp. 17–30, Springer, 2016.

Appendix A

Level Properties

Here you find all the properties that were explored for evaluation of the survey.

A.1 Inherent Properties

There are various properties that are inherent from generation. We can use those properties for the prediction of the 'interestingness'. The inherent properties fit into two categories. The first of which are properties of the generated level graph. These properties that we consider are:

1. Number of vertices (i.e. platforms)
2. Number of edges (i.e. connection from vertex U to V, counted double if V to U exist)
3. Average number of incoming edges
4. Average number of outgoing edges
5. Largest number of incoming edges
6. Largest number of outgoing edges
7. Average vertex centrality (the average distance of any pair of vertices U and V where U != V)
8. Number of cliques
9. Largest clique size
10. Number of branches

11. Number of cycles

The other category consists of functional properties of a generated level. These are:

1. Average Linearity (i.e. for each 3—consecutive in the path to the goal—connected platform positions U, V and W, what is the average cosine between the expected forward vector ($U - V$ and $W - V$).
2. Average platform size
3. Number of platform with mechanic X (where X is any of the implemented mechanics)
4. Number of score items