University Utrecht

Procedural game environment for training player skills

A Game and media technology Master's Thesis

Author: Joel van der Werf, 7007094

Supervised by Dr. Sander Bakkes and Prof. Dr. Albert Ali Salah

ABSTRACT

Wanting to learn someone to play a video-game is tightly tight to the concept of literacy, which means the competence or knowledge in a specified area. In video-games, we call this game literacy, which means we want people to increase their skill in playing video games. The common way of teaching a user game literacy is through the use of tutorial's which teaches the player the base mechanics of the game. Tutorials in most situations are created by designers which forces players to complete some simple tasks in a certain order that learn them the base mechanics of the game. After completing the tutorial, the player plays through a series of levels, in which slowly the difficulty of the game scales up. The problem with this approach is that we have players of all kind of different skill levels in playing video games. We have to take the different skill levels of player's into account. If we do not take the different skill levels of players into account, they might quite the game because of boredom or being too frustrated because of not making any progress in the game. To solve this problem, we propose a method that adaptively changes the generation of game levels based on the player's performance in the game and try to target the difficulty of the levels based on the player's skill in playing video games. Our method focuses on generating game levels with game mechanics the user needs improvement on. Results show that our method is enabled to train people in their game literacy better than a baseline adaptive environment.

Keywords; Procedural Content Generation; Player modeling; Game literacy;

Contents

List of Tables

List of Figures

Listings

ACKNOWLEDGEMENTS

This thesis would not have been possible without the contribution and help of some people, so I would like to express my gratitude to them. First, I would like to thank my supervisor Dr. Sander Bakkes for his help, patience, and the insights he offered me in our weekly meetings. I would also like to thank my family, friends, and colleagues for their never-ending support throughout my studies, which helped alleviate all the stress and tension that appeared in the last 2.5 years.

Joel van der Werf

Chapter 1

Introduction

Teaching someone how to play a video game is closely linked to the concept of literacy [46] which refers to competence or knowledge in a specified area. In the past, literacy was mostly used to describe the competence and knowledge of writing and reading, but nowadays its meaning also extends to other research areas. For example, the book What Video Games Have to Teach Us About Learning and Literacy by James Paul Gee [12] introduces the concept of video game literacy, which involves not only the ability to decode and understand meanings with respect to the domain of video games, but also the ability to produce meanings. Understanding has therefore become equivalent with the ability to access content, that is, to play video games.

1.1 Game Literary and Video Games

In this section, we will discuss game literacy and video games we do this based on our test bed of the popular video game Super Mario Bros, since it is a well-known subject of literacy studies related to video games [28]. Super Mario Bros was developed by Nintendo [37] [32] [45] and falls under the video game genre [8] called platformers. A video-game genre is a specific category of games related by similar gameplay characteristics. Most platformer games use the same game play elements as presented in the game Super Mario Bros.

The main goal of each level in the video game Super Mario Bros is to get the player character to the end of the level. This is achieved by moving the player character from left to right through the game level. Within each level, there are several obstacles and challenges the player must overcome. The auxiliary goals of the video game Super Mario Bros are to collect all the coins and to get the highest score in

each level. The score given to the player for each completed level is dependent on the number of coins collected, how fast the player has completed the level, and the number of enemies killed. Super Mario Bros also allows players to make up their own goals throughout the game. An example of such a goal would be to beat the game as fast as humanly possible, a concept called speed running [21]. Another example would be to collect all collectibles (coins) in the level.

The gameplay elements for the player character are to move either to the left or the right and the ability to jump and run. The player character can also shoot fireballs depending on the character's state. The player character state depends on how many power-ups have been collected. The player character can be in one of the following states:

- Small at the beginning of a game;
- Large, in which the player can crush some objects by jumping into them from below;
- Fire state, in which the player character can shoot fireballs;

The main obstacles for the player character to overcome are gaps and a variety of patrolling enemies in each of the levels. If the player characters fall down a gap, they lose a life. If the player character touches an enemy, they lose a life when they are still in the small state. However, if they are in the big or fire state, they shift back to the small and large state respectively. Most patrolling enemies can also be beaten by the player character if the character jump in such a way that they land on the enemy from above or hit the enemy with a fireball. However, the outcome of this action is dependent on the type of enemy: Goombas die and are removed from the level. Static enemies such as piranha plants are not vulnerable to this action and will still hurt the player character. Finally, turtle enemies withdraw into their shells. These shells can then be used to throw at other enemies and kill them.

The last important thing to know about Super Mario Bros is that there are items spread across the level, either hidden in blocks or visible. The player character can find these items in blocks marked with a question mark and gain them by jumping and hitting the blocks from below. Available items include:

- Coins, which can be collected to increase the player's score and to unlock extra lives for every hundred coins collected;

- Mushrooms, which change the player character's state from the small to the large and finally to the fire state;
- Flowers, which change the state of player characters to the fire state if they are already in the large state;

In Figure 1.1 we can see an example of a Super Mario Bros level.

Figure 1.1: Example of Super Mario Bros level

1.2 Procedural content generation

Procedural content generation [39] [16] refers to the methods used to generate content for games and simulations algorithmically, which normally would have been hand-made by people. A large variety of procedural content generation techniques exist, which have been applied to the content and styles of a wide variety of video games. Procedural content generation has been particularly studied for the generation of video-game levels [45] [3].

One of the first games to use procedural content generation was the game Elite [4]. Released in 1984, Elite is a space trading and combat simulator developed by David Braben and Ian Bell and originally published by Acornsoft. For the game design, Braben and Bell needed to create numerous galaxies in which trading was possible.

Instead of having a designer create all the galaxies, they used procedural content generation to algorithmically create all galaxies with planets, stars, and other spaceships to trade with.

One of the most famous open-source projects involving procedural content generation for Super Mario Bros, is the project Infinite Super Mario Bros [32] by Markus Persson. This project is a public domain clone of Nintendo's classic platform game Super Mario Bros. The game Infinite Mario Bros is playable on the web, and the source code of the project is also available online. Infinite Super Mario generates all its levels using procedural content generation.

The project Infinite Super Mario Bros has been used as a starting point for procedural content generation projects related to game levels for Super Mario Bros. In 2009 there was a competition [3] to create the best agent in terms of level generation with the Infinite Super Mario Bros project as a starting point.

Another use of procedural content generation is to change game play parameters with relevant real-time player data and game play context information data while a game is in progress. This creates a way for developers to adjust their content generation process in run time to better suit the needs of individual players. This type of procedural content generation is called adaptive procedural content generation [31]. Changing content based on procedural content generation driven by computational models of user experience is a subsection of adaptive procedural content generation called experience-driven procedural content generation [49].

An example of this would be to adaptively change the generation patterns of a video game to increase or decrease its difficulty [27]. For example, in the Infinite Super Mario Bros game levels are generated based on a difficulty value. The difficulty value can be changed based on the output of computational models of user experience to give the player the right balance between player skill and difficulty in the game.

1.3 Rage quitting

Rage-quitting [7] is defined by psychologists as the act of quitting a game when you are losing, about to lose or feel that you will lose. Within the game Super Mario Bros this can happen when a player is stuck at a certain point in a level and is unable

to progress beyond it. This can be because the user is not yet skilled enough to overcome certain challenges. The examples given in the previous paragraph about adaptive procedural content generation could be a solution to the problem of rage-quitting, since it allows developers to adapt the difficulty of the game's content based on the player's skill level in playing video games. It is specifically important to not let users rage-quit if the intention is for them to acquire game literacy.

1.4 Research question

The main goal of this research is to create a method used for adaptive procedural content generation for the use case of training the user in acquiring basic game literacy, and to determine which concept would be most conducive to achieving competence in playing a game. This will be accomplished by letting users play game levels generated by procedural content in a Super Mario Bros [28] clone. To reach this main research goal, the following sub questions need to be answered:

1. How does a user acquire game literacy?
2. How can users be kept engaged throughout the learning process and prevented from wanting to quit?
3. To what extent is an adaptive procedural content generation-system able to train a user's game literacy in playing platformer games?

Chapter 2

Related Work

One of the key challenges of video game design is teaching new players how to play the game. Although game developers frequently use tutorials to teach game mechanics, little is known about how tutorials affect game learn-ability and player engagement. In most games, tutorials are structured levels that are the same for all players. In their paper The Impact of Tutorials on Games of Varying Complexity [33], the authors investigate the impact tutorials have on player engagement by letting 45000 players play eight tutorials for different games of varying complexity. Results showed that tutorials increased play time by as much as 29 percent by teaching more complex game situations; they did not, however, improve player engagement if the situations were too easy to overcome. One of the problems to solve is how to keep all players engaged since player skill can be of varying levels.

2.1 Game literacy

Game literacy is the ability to access, critically evaluate, analyze, and create games. The article The power of game literacy [11] by Joshua Gad explains that game literacy can be broken down into three levels: basic, advanced, and creative "literacy".

- Basic literacy refers to the ability to consume media;
- Advanced is the ability to analyze media;
- Creative is the ability and knowledge to make media;

Basic game literacy in video games is taught by letting players play something that is called a tutorial level. A tutorial level forces the player to employ one or more of the game's mechanics to overcome pre-designed scenarios. Advanced game literacy is

taught by exposing the user to different kind of challenges in the game, which require insight into one or more of the game mechanics to be overcome. An example of creative game literacy is the game Super Mario Maker [30] by Nintendo. In this game, users can make their own game levels for the game Super Mario. These game levels can be built with different kinds of pre-made game assets.

The distinction in the levels of game literacy shows that it is necessary to create a method that can teach users basic and advanced game literacy in the game Super Mario Bros, basic game literacy because it involves being enabled to play the game, while advanced literacy is used to analyze the game and the situation the user is in. For the purposes of this research, the user is not required to be capable of making a game or game level for Super Mario Bros, since we assumed that it would not increase the user's proficiency in playing the game.

2.2 Mechanics

Before beginning to teach someone basic game literacy, it is necessary to first be aware of the game mechanics involved in a game of Super Mario Bros. This is needed, so we know which mechanics we need to teach a user, so the user can successfully overcome the challenges presented in the game. We use the game mechanics presented in the game Super Mario Bros, since other platformer games include the same type of mechanics as presented in Super Mario Bros.

Extensive research has been done into the game mechanics of the game Super Mario Bros. For example, the paper Mario Level Generation From Mechanics Using Scene Stitching [19] generates an artificial intelligence-based on the mechanics of Super Mario Bros and describes the game mechanics, as can be seen in Table 2.1.

Game mechanics	
Name	Description
Jump	Tracks if Mario left the ground (has a small y axis change less than a certain threshold).
Low Jump	Tracks if Mario made a low jump (has a high y axis change greater than a certain threshold).
High Jump	Tracks if Mario made a high jump (has a large x axis change greater than a certain threshold).
Long Jump	Tracks if Mario made a long jump.
Stomp Kill	If Mario kills an enemy by jumping on top of it.
Shell kill	If Mario kills an enemy by pushing a Koopa shell.
Fall kill	If a enemy dies because it fell off the game screen.
Mode	If Mario changed his mode (small, big, and fire).
Brick Block	If Mario bumped into a brick block.
Question Block	If Mario bumped into a question mark block.

Table 2.1: Name and description of the mechanics in an Super Mario Bros game.

Another fundamental aspect of Super Mario Bros is the physics of the player character named Mario. The article A Complete Guide To Super Mario Bros Physics Engine [18] explains the parameters and values involved in the physics engine of the games Super Mario Bros and Super Mario Bros 2.

It is important for use to implement the physics and game mechanics of Super Mario Bros correctly to give the player the feeling of playing an actual, real version of Super Mario Bros.

2.3 Game and Level design theory

An important aspect to keep in mind is the flow of the content of the game, which helps keep the player optimally engaged [24] in the training process. In order for a player to be optimally engaged in the training process, the challenge should be presented at a level equivalent to or slightly higher than the current skill level of the player. Not too easy, but also not too hard. The challenges of the game should then be increased as the player's skill increases. The designer should be aware that drastic changes in the game's difficulty may result in the loss of player engagement. If the player is optimally engaged in the content, then the player is considered to be in a flow state. The concept of flow was introduced by Mihaly Csikszentmihalyi [25].

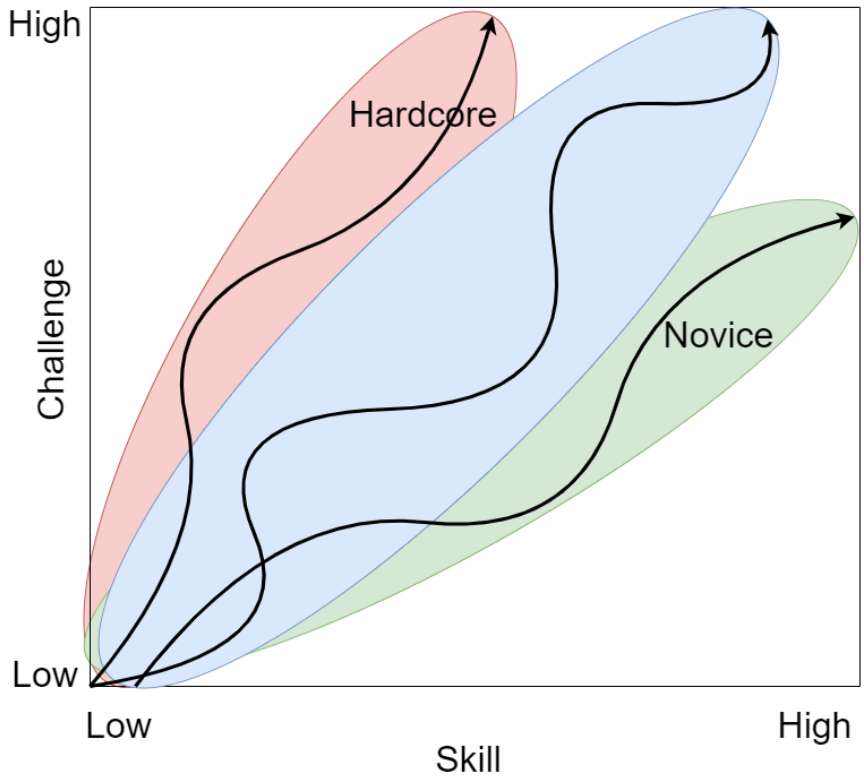


Figure 2.1: Different players have different flow zones

Figure 2.1 shows an example of how the flow should look for a game where different types of skilled players are supposed to be kept engaged in the content. From the graph, we can see that the line is not linear. The reason for this is that if players are kept continuously challenged, they might get exhausted.

Figure 2.3 shows that the curve operates into what is known as the flow channel. This indicates a person's tolerance for a temporary lack of or spike in engaging or challenging content, while being given hope that more challenging or relaxing content is on the way. That is why it is important for the flow of the content to also keep into account some sort of short cooldown period after the completion of a challenge (learning goal).

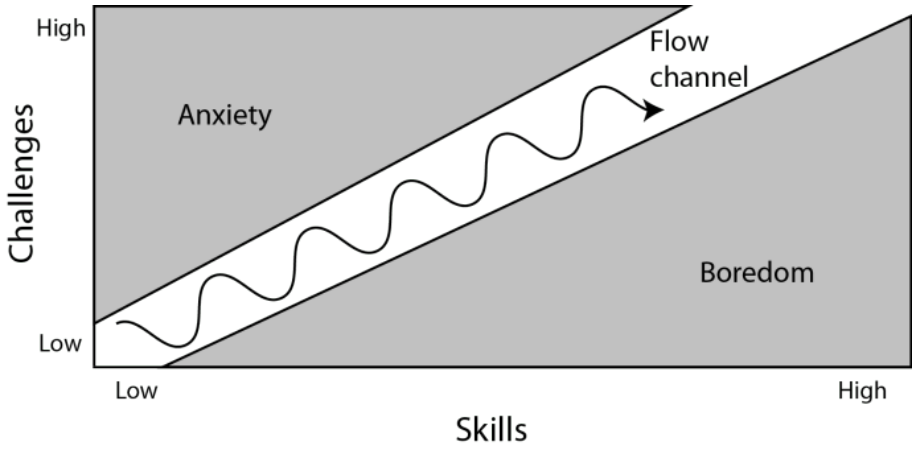


Figure 2.2: Example chart of the flow introduced by Mihaly Csikszentmihalyi

Figure 2.3

The article Flow in Games (and everything else) [6] mentions that the flow channel and the curve may be different for different types of players. To design an engaging experience for a broader audience, the experience cannot be the same for all users. This means that the gameplay experience for each user should be adapted to his or her personal flow zone. An example of personalized flow zones can be seen in Figure 2.4 where the flow rate of an average player is indicated with red arrows, while the blue arrows indicate different skill levels of players, which need to be taken into consideration for adaption to get that user into the flow zone.

The article The effectiveness of adaptive difficulty adjustments on students' motivation and learning in an educational computer game [17] mentions that tying adaptive difficulty to learning goals will result in significantly higher learning rate outcomes. The authors have tested this by comparing games that make adaptive difficulty changes with two equivalent non-adaptive learning activities.

For our method, it is important to keep the concept of flow and balancing player skill with game difficulty in mind. This is so we can keep the player optimally engaged in the training process of obtaining game literacy in playing platformer games.

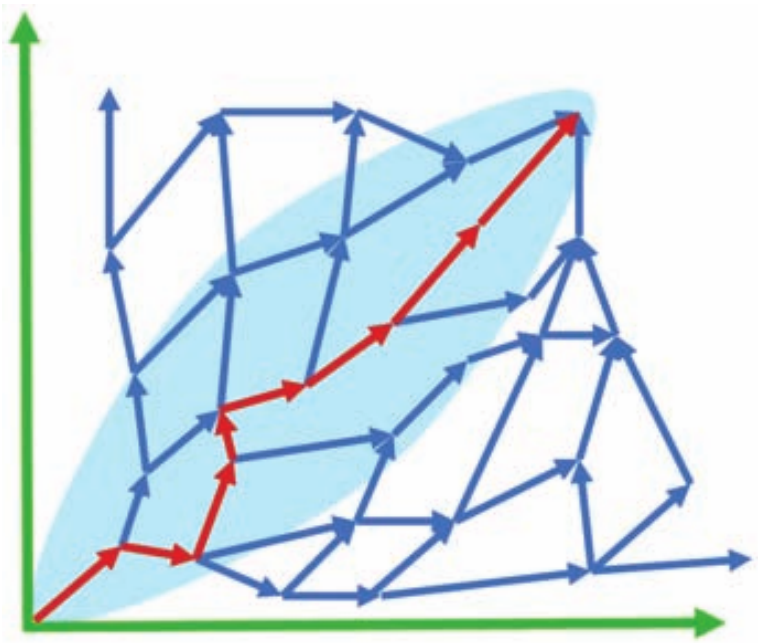


Figure 2.4: Adapting to different users personal flow zone

2.4 Player modeling

The paper An inclusive view of player modeling [38] explains the concept of player modeling as describing all aspects of a predictive model of player actions in games. This includes the detection, modeling, prediction, and expression of the human player characteristics that are manifested through cognitive, affective, and behavioral patterns. These predictive models are constructed based on data obtained from the interaction of a human player with a game [48]. Player models are built on dynamic information obtained during game-player interaction, but they could also rely on player profiling information.

In the paper Player modeling: Towards a common taxonomy [23] the authors discuss some commonly used methods for player modeling. These are the following methods:

1. Genetic algorithms [14], use the process of natural selection. Genetic algorithms are part of a subgroup of evolutionary algorithms.

2. Coevolutionary algorithms [34] evaluate the fitness of an individual in a population. The individuals in the population are evaluated based on their interactions with other individuals. The interaction between individuals in the population can be either a competitive or Cooperative interaction.

3. Evaluation functions, are concerned in understanding and modeling how players evaluate the different game states. Once we have modeled the way a player sees the game, we can better predict the player's future actions

4. Neural networks [29] can be seen as a "black box" that maps inputs to outputs. This black box is implemented through a set of interconnect nodes (neurons) that are activated according to their weighted inputs. A training process is used to adjust the weights to the desired objective.

5. Rules-based models consist of a set of conditions that, when satisfied, generate a series of actions.

6. Probabilistic Models are a statistical technique used to consider the impact of random events or actions in predicting the potential occurrence of future outcomes.

7. Monte-Carlo Tree Search [5] uses game simulations that are used to estimate the values of game states and actions. This information is used to progressively improve the quality of the simulations. Can be specifically useful for games like poker to decide the best option based on previous simulation outcomes.

8. Machine learning, uses learning algorithms applied to player modeling.

In the article player modeling, [22] the authors give a definition of the taxonomy of player modeling and the three major components that player modeling consists of: Input, a computational model, and the output. This flow can be seen in Figure 2.5.

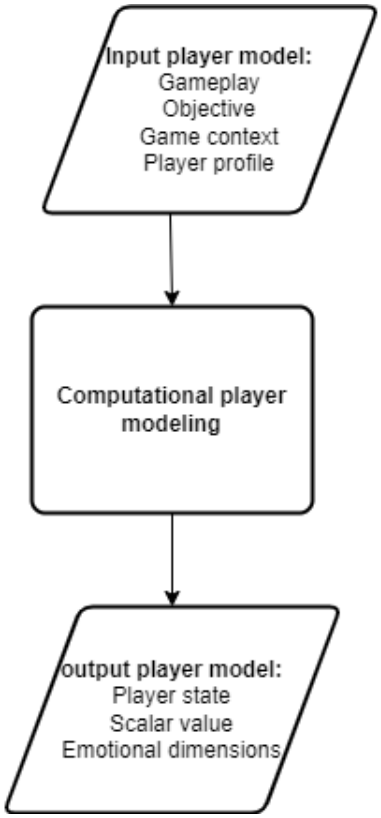


Figure 2.5: Three major components of player modeling [22]

The input for a player model is data regarding the game and the player, this can be one or more of the types of input explained below:

1. Anything that a human player is doing in a game environment (behavioral data).
2. Objective data collected as bodily responses to game stimuli, such as physiology and body movements.
3. The game context consisting of any player-agent interactions, but also any type of game content viewed, played, and created.

The last type of input refers to player profile information and includes all the information about the player that is static and is not directly (nor necessarily) linked to gameplay. This may include information related to a player's personality, such as cultural variables and general demographics such as gender and age.

The second step includes the actual modeling; this can either be a top-down (model-based) or bottom-up (model-free-based) approach. The top-down approach concerns itself with theoretical fields of knowledge such as humanities and social sciences, which

hypothesize models to explain concepts. This is then followed by an empirical phase, in which the researchers experimentally determine to what extent the hypothesized models suit the researcher's observations. Bottom-up (model-free) approaches refer to the construction of a mapping model between player input and a player state representation. As such, model-free approaches follow the approach of the exact sciences, in which observations are collected and analyzed to generate models without a strong initial assumption on what the model looks like or even what it captures. The model-free and the model-based approach can also be combined. The paper A systematic review of data-driven approaches in player modeling of educational games [15] analyse and reviews data-driven player modeling approaches. The authors state that it would be preferable to use a combination of model-free and model-based approaches, since it could lead to a better understanding of player behavior.

Finally, we come to the output of player modeling, which is usually a set of particular player states. Such states can be represented as a class, a scalar (or a vector of numbers) that maps to a player state, such as the emotional dimensions of arousal and valence [1] or a behavioral pattern or a relative preference.

The paper Personalised gaming: a motivation and overview of literature [42] states that techniques can be applied to automatically adapt the challenge that a game poses to the skills of a human player. This is called difficulty scaling [40] When applied to game dynamics, difficulty scaling is usually aimed at achieving a game wherein the human player is challenged neither too little nor too much. Some methods mentioned on how to achieve this are:

- Controlling the game environment;
- Establishing a discrete parameter that determines how difficult the game should be;
- Coevolutionary algorithms [34];
- Estimating and tracking the 'emotional intensity' and mapping this to the emotional dimensions of arousal and valence [1];

The purpose of difficulty scaling is to allow both novice and experienced players to enjoy the appropriate challenge (learning goals) that the game offers. The paper Personalised gaming: a motivation and overview of literature explains this based on the concept of *player modeling*.

The paper Player Modeling for Intelligent Difficulty Adjustment by Olana Missura and Thomas Gartner [26] has attempted to solve a similar problem regarding balancing difficulty and player skill in real time. The authors investigated the use of machine learning for dynamic difficulty adjustment. The author's aim was to create a difficulty adjustment algorithm that does not bother the actual players while they are playing the game. The author's approach to building a difficulty model consists of clustering different types of players, finding a good difficulty adjustment for each cluster, and predicting the cluster for short traces of gameplay. The author's results suggest that dynamic adjustment and cluster prediction together outperform alternative machine learning algorithms to determine game difficulty adjustment for individual players.

An example of player modeling in Super Mario Bros is given in the papers Modeling player experience in Super Mario Bros [44] and modeling player experience for content creation [43]. In these papers, the authors investigate the relationship between level design parameters of platform games, individual playing characteristics, and player experience using player modeling. For their research, the authors collected and analyzed the following data:

- Controllable features of the game the parameters used for the procedural content generation of the levels;
- Gameplay characteristics related to how the user plays the game. They measured statistical features such as how often and when the player jumped, ran, died, and so on;
- The player's experience of playing the game, measured through a 4-alternative forced-choice questionnaire;

The results of the data show that various emotions are at play when playing a game and how users perform under these emotions. The main focus of the study was on the emotions of frustration, challenge, boredom, anxiety, predictability, and fun. It was shown that the different emotions can have positive or negative effects on the gameplay of the user. In future work, they want to use the results of their study to create Super Mario Bros game levels based on the emotions the user is experiencing. This means it is important to keep players' emotions in mind when designing procedurally generated content for video-game levels.

In the paper Player-Centred Game Design: Player Modelling and Adaptive Digital Games [20] the authors state that player modeling and adaptive procedural content generation technologies may be used alongside existing approaches to facilitate improved player-centered game design and in doing so provide a more appropriate level of challenge, smooth out the learning curve, and enhance the gameplay experience for individual players.

For our method, we wish to use the concept of player modeling [38] [23] to target the difficulty of the game for an individual user based on his or her player skill. We also wish to use player modeling for the user case of determining which training goals we have to present to the user in the game in order to improve the user's game literacy in playing platformer games. This goal can be reached by letting the player model establish a discrete parameter for the targeted difficulty of the game and the learning goals. The concepts presented on player modeling in the papers: Modeling player experience in Super Mario Bros [44] and modeling player experience for content creation [43] provide us with useful information for reaching our goal of targeting difficulty based on player skill and providing the user with training goals that help them increase their skill in playing platformer games.

2.5 Procedural content generation for Super Mario Bros

There has been substantial research done on generators of Super Mario levels; in 2010 there was even a competition. In the paper, The 2010 Mario AI Championship Level Generation Track [3] a group of authors present the results of this competition. The goal of the competition was that competitors participated by submitting level generators that could generate levels for the game Super Mario Bros. The goal of these level generators was to generate new levels for Super Mario Bros tailored to an individual player's playing style and player skill level. In the following paragraphs, we will discuss some level generators presented in the paper The 2010 Mario AI Championship Level Generation Track that could be of interest to our work.

The Notch algorithm is the algorithm that is used in the game Infinite Super Mario Bros [32]. Notch builds the levels from left to right, iterating over each tile on an x by y grid. At each tile, there is a probability of adding components like enemies,

platforms, and gaps to the level. Adding these components is based on the following parameters:

- Seed: is a unique identifier to the generated level;
- Difficulty: is how difficult the generated level should be;
- Type of level: is the visuals used for the generated level;
  - Overground;
  - Underground;
  - Castle;

When the generation of components is completed, basic checks are done to make sure the generated levels are playable. Basics checks include the difference in height between rows of blocks and the width of gaps made in the level.

The downside of the Notch algorithm is that it has no control over the generation of levels, meaning it is completely random. For this reason, the paper Feature Analysis for Modeling Game Content Quality [36] by Noor Shaker and Georgios Yannakakis introduces a method that expands on Notch. In this method, they introduced generating content based on the controllable feature of the game Super Mario Bros. This algorithm based on controllable features is explained in the paper Comparative Evaluation of Procedural Level Generators [37] in which Shaker and Yannakakis name this method Parameterized Notch. The controllable features are parameters that can be adjusted to guide the generation of levels. These are the following parameters:

- The number of gaps;
- Width of the gaps;
- Number of enemies;
- Placement of enemies;
- Number of power-ups;
- Number of boxes(coins);
- Type of block
  - Type rock (breakable when Mario is in powered up state);

– Type solid;

When a level is generated, the same checks are done that are used in Notch to ensure that a level is playable from beginning to end. The Parameterized Notch method can also use randomization for the controllable parameters, the method is then called Parameterized Notch-Randomized.

Hopper is another interesting method introduced in the paper The 2010 Mario AI-Championship: Level Generation Track [3] by Tomoyuki Shimizu and Tomonori Hashiyama. Hopper generates its levels by writing them from left to right and placing features with specific probabilities in much the same way as Notch and Parameterized Notch do. It was built with adaptability in mind so that the probabilities could easily be altered depending on the player's skill and prior performance. The difference between Notch and Hopper is that the generated parts are alternated with pre-designed parts.

In the paper, A Comparative Evaluation of Procedural Level Generators in the Mario Artificial Intelligence Frameworkk [37] the authors evaluate different Mario level generators that were used in the 2010 competition. In their paper, the authors compare the generated levels with the levels made for the original game of Super Mario Bros. For these, they measured the following statistics for each procedural level generator: The leniency shows how difficult a level is. The linearity is how many height differences there are in the generator; the higher the height difference is in a generated level the lower the linearity score gets. Density is a measure of how many platforms are stacked on top of each other. Pattern density measures how many patterns (parts) of the original Mario levels are used inside the generator. Pattern variation measures the unique occurrences of patterns used within the generator. The more unique the pattern is, the higher the score is. In Figure 2.6, we can see the results of the methods we previously discussed.

generator	leniency	linearity	density	pattern density	pattern variation	compression
Notch	0.67 (0.06)	0.1 (0.11)	0.4 (0.16)	0.13 (0.02)	0.27 (0.08)	0.53 (0.03)
Notch param	0.85 (0.06)	0.04 (0.05)	0.81 (0.08)	0.08 (0.03)	0.24 (0.07)	0.36 (0.08)
Notch param ran	0.86 (0.08)	0.08 (0.06)	0.8 (0.1)	0.08 (0.03)	0.17 (0.09)	0.47 (0.08)
Hopper	0.72 (0.04) 0	0.15 (0.16)	0.6 (0.15)	0.1 (0.02)	0.29 (0.05)	0.65 (0.05)

Figure 2.6: Overview comparison of level generators: mean value (standard (deviation) of each metric on the output of each generator [37]

The results show that parameterization in level generators plays a large role in changing the nature of generated levels compared to the other level generators from the competition. This would make Parameterized Notch the ideal candidate for an adaptive procedural content generation level generator because it allows the researcher to change the parameters from Parameterized Notch to the needs of an individual user.

2.6 Procedural content generation and difficulty adjustment

The online article The Mario Game that Gets Harder the Better You Are (Infinite Adaptive Mario) [47] [35] by Ben Weber proposes a method to adapt the difficulty of the game Infinite Mario Based on the difficulty parameter for each individual user.

Infinite Adaptive Mario scales up the difficulty by increasing the frequency of gaps, the average size of gaps, variation of ground height, and the number of enemies. Scaling up the difficulty also tends to result in a larger number of possible paths through a level. The player begins at difficulty level 50, which produces levels with a moderate degree of challenge. Upon successful completion of a level, the challenge is increased. The faster the user completes a level, the higher the increase is for the difficulty score for the next generated level. At the moment of death, the difficulty is decreased based on the amount of progress made by the player. If the player is close to the goal at the moment of death, then the difficulty is decreased only a small amount. If the player is close to the beginning of the level at the moment of death, then the difficulty is decreased a large amount. A new level is generated when the player either completes a level or fails to complete a level after three deaths.

The paper Challenge Balancing for Personalized Game Spaces [13] has attempted to solve a similar problem in regard to balancing difficulty and player skill. In this paper, the authors propose an approach for personalizing the game space in which an altered version of Infinite Super Mario Bros is played in terms of levels to the end of tailoring the experienced challenge to the individual user during the actual play of the game. The authors' approach specifically considers two design challenges, namely implicit user feedback and a high risk of user abandonment. The method the authors propose is to make use of the concepts explained previously, such as an altered version of Infinite Super Mario Bros for the level generation and player modeling to adjust

the difficulty of the procedural content generation. The author's approach works in three steps:

- Implement a policy that is appropriate in expectation across users to be used for initializing the online game;
- Implement a mapping from gameplay observations to the player experience to be used for guiding the online game personalization;
- Rapidly establish an appropriate policy for the individual user in online gameplay employing the learned feedback model and a straightforward model of user abandonment;

The method used for the research described in this thesis also intends to balance player skill and the difficulty of the generated levels. However, our method also needs to take into account the different training goals that are set for the individual player in order to gradually learn the user's basic and advanced game literacy. For this, different parameters would need to be tuned rather than just the difficulty, and this would also lead to a more complicated player model.

There are also ways to increase the difficulty based on measurements taken outside the game environment. For example, In the paper Modeling and Adjusting in-game Difficulty Based on Facial Expression Analysis by Blom et al [41] the authors make use of facial expressions to determine if they should increase or decrease the difficulty of the game. The authors record the user's facial expression and use facial expression analysis to adapt to the difficulty. For example, if the player is angry, it lowers the difficulty, while if the player has a neutral face possibly indicating boredom, it increases the difficulty of the game.

The papers Infinite Adaptive Mario [35] and Challenge Balancing for Personalized Game Spaces [13] are of importance to our work. This is because both papers aim to balance player skill and the difficulty of procedural generated game levels, using Super Mario Bros as a test bed. That is why these papers will be used as starting point for our work.

2.6.1 Evaluation of gameplay agents

Hoyle [10] is a program that learns to play a broad variety of board games very well, in real time. Hoyle is not a traditional game-playing artificial intelligence: it searches

and tolerates, even encourages, inconsistent and incomplete knowledge. Hoyle minimizes search, focusing instead upon learning a set of reasons or a logical basis for a course of action or belief in the board game. Hoyle learns during competition, and demonstrates substantial, learned expertise after relatively little training.

The paper the Ideal Trainer [9] by Susan L. Epstein talks about how they evaluate learning for different type of agents. They evaluate the different types of agents (Artificial intelligence players) by letting the agents play x type of board games and seeing what the agents learn from the different type of board games. The results suggest that teaching a program by leading it repeatedly through the same restricted paths is an overly narrow preparation for the variations that appear in real-world experience. The results also demonstrate that variety introduced into training by random choice from the agents does not constitute reliable preparation, and that a program that directs its own training may overlook important situations. The results argue for a broad variety of training experience with play at many levels. This variety may either be inherent in the game or introduced deliberately into the training as noise. A blend of expert guidance, knowledge-based and directed learning for a group of agents or one by itself, is shown to be particularly effective for learning during competition.

This paper deals with training people in acquiring game literacy in playing platformer type games like Super Mario Bros instead of artificial agents but the results shown by the study done by Susan L. Epstein are still relevant to this research regarding expert guidance, knowledge-based and self-directed elaboration, which are effective methods in learning.

Chapter 3

Method

The main problem to be solved by this research is: a method used for adaptive procedural content generation for the use case of training the user in getting basic game literacy, which concept is most conducive to achieving competence in playing a game? An attempt to solve this has been made by letting users play procedural content-generated game levels in a Super Mario Bros game clone. A variety of methods have been used to create, analyze, and evaluate the following research questions, already stated in section 1.4:

1. How does a user acquire game literacy?
2. How can a user be kept engaged in the learning process and prevented from wanting to quit?
3. To what extent is an adaptive procedural content generation-system able to train a user's game literacy in playing platformer games?

To answer these research questions, literature research has been employed. A user-centered experiment has also been conducted. In this experiment, some users were invited to play variations of a game clone of adaptive procedural content generation Super Mario Bros and data is measured regarding gameplay and level generation, while the user is playing the game. The data gathered in this experiment is then used to analyze and discuss the method presented in this paper.

3.1 Answering research question one

The first sub-question is: *how does a user acquire game literacy?* Which is the concept most conducive to achieving competence in playing a game? To answer this question,

literature research into the concept of game literacy and the mechanics of Super Mario Bros has been done. For game literacy this literature research has shown that there are three types of game literacy:

- *Basic literacy* refers to the ability to consume game media. A user will be enabled to use the default mechanics of the game.
- *Advanced literacy* is the ability to analyze media. This means the user has insight to determine which game mechanics available in the game need to be used to overcome different scenarios with the combined mechanics in the game.
- *Creative literacy* is the ability and knowledge to make levels for a certain game;

This has led to the following definition of game literacy for this thesis project: the ability of a user to play and analyze a level of the game Super Mario Bros. This means the user needs to learn the base mechanics of a game of Super Mario Bros and be enabled to execute them. This is known as basic game literacy. The user also needs to learn how to overcome different combined obstacles in the game with the game mechanics learned from the user's own perception. This is known as advanced literacy.

The list of game mechanics present in Super Mario Bros can be seen in the first paragraph of previous work in section 2.2. These mechanics involve the different types of blocks, enemies, and obstacles a user might come across in a normal game of Super Mario Bros and which actions to take to overcome these obstacles. Each mechanic can be associated with a training goal; if a player can overcome all the training goals the player can be said to have reached basic game literacy. If the player can overcome differently combined training goals to a certain level of proficiency, the player can be said to have reached advanced game literacy, since the player can now identify which action to use to overcome obstacles in the game. Table 3.1 shows the training goals with the corresponding actions the user has to take to complete these training goals. If a player can create levels for Super Mario, the player has achieved creative literacy, however this is not one of the goals of this thesis project, which instead focuses on training the user in basic and advanced game literacy.

Training objectives	
Training goals	Completion
Walking	Move Mario from left to right.
Small jump	Jumping over or on a block of elevation one in height.
Medium Jump	Jumping over or on a block of elevation 2/4 in height.
Chasm jump	Jumping over a hole in the level.
Enemies	Jumping over or on top of enemy without hitting it.
Platforming	Jumping on top of a platform.
Static obstacle	Mario overcomes fire chain and piranha plant by timing a jump over these obstacles.
Question Block	Mario picks up item hidden in block.

Table 3.1: Goal and objective of the training goals assigned in a Super Mario Bros game.

3.2 Answering research question two

The second sub-question is: *how can a user be kept engaged in the learning process and be prevented from wanting to quit?* For this, literature research has been employed regarding the game and level design in section 2.3 and the concept of flow in section 2.3 in video games. Flow is the concept of keeping players optimally engaged in the content they are currently experiencing. This means that the game needs to offer some challenge to the player but should also have certain moments in the game where the player can relax a bit.

Adaptive methods and player modeling may be used to get people into the flow state. Player modeling is the study of computational models of users in games. This includes the detection, modeling, prediction, and expression of human behavior. In short, this means the adjustment of the game's content based on the needs of individual players. To adjust the game's content, a method called adaptive procedural content generation can be used. Procedural content generation is a method of creating data algorithmically. In the game Super Mario Bros, this would mean generating challenging game levels for the user to play. The adaptive part is changing the level of difficulty and training goals of the game based on the performance of the individual user in previous levels. The player model can then determine if the challenge of training goals for a certain player needs to be increased or decreased based on the tracked data.

3.3 Answering research question three

The third sub-question is: *To what extent is an adaptive procedural content generation system able to train the "platformer" game literacy of a user?* To answer this research question first some research into the pros and cons of traditional structured tutorial levels for video games is conducted. Secondly, the major components needed to create an adaptive game environment for training player skills are discussed. Lastly, a user-centered experiment is conducted. In this experiment, an adaptive procedural content generation game environment is created based on a Super Mario Bros clone to see if the chosen method would be well suited to training the game literacy of a user.

Most games use some sort of structured level [33] to teach a player the basics of a game. These structured levels are created by a designer and are the same for all players. The problem with structured created levels regarding training the user in-game literacy is that the individual learning curve for each user is not considered in pre-made levels. In pre-made game levels, the challenges of each level are determined by the game designer. This can lead to a decrease in the learning rate since the content might either be too challenging or too easy for the user's current skill level and knowledge in game literacy. This is where adaptive procedural content generation can help by adapting the training goals and challenges of the levels to the skill level of an individual user. Using adaptive methods could lead to an increased learning rate in-game literacy since the user can now be kept optimally engaged in the training process by adapting the difficulty of the game to their skill level.

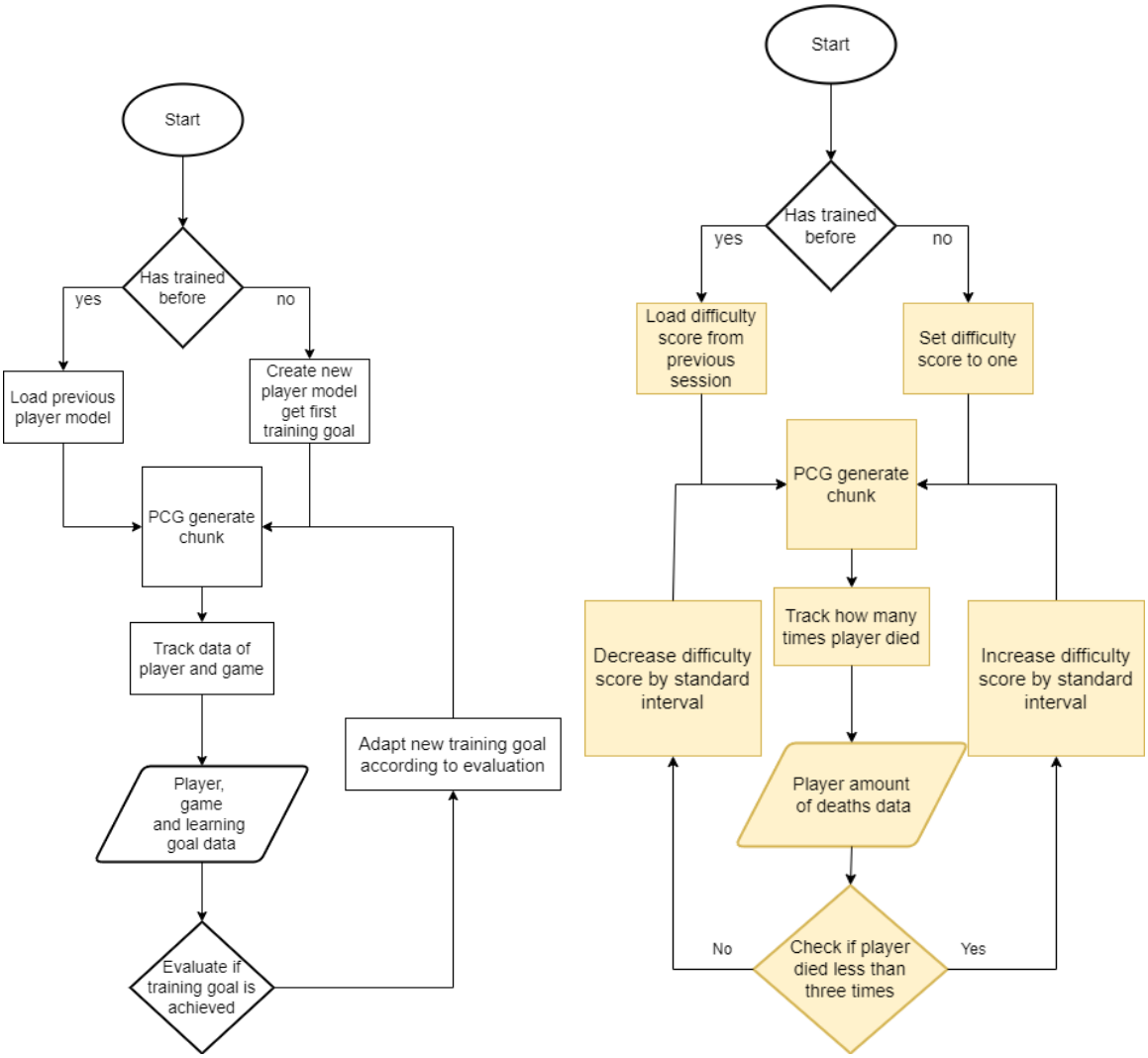
3.3.1 Major components of an adaptive environment

The answer to research question two shows that player modeling can be used to determine if the training goals are either too easy or too hard for the user. But how can the procedural content generation be adjusted to increase or decrease the difficulty of the training goals?

First, literature research was done on different procedural content generation algorithms to weigh the pros and cons of existing work for the chosen method. The literature research done on procedural content generation explained in section 2.5, showed that parameterization plays a significant role in changing the nature of generated levels. Parameterization also gives control to the creator related to what should and should not be included in a generated game level. For this research's goal of

generating levels that consist of challenging content with individual training goals for a user to improve their game literacy, a method with parameterization would be preferable. The method Parameterization-Notch-Random uses parameterization, which allows researchers to decide what to include and what not in a generated level regarding obstacles set by training goals. Parameterization-Notch-Random also uses a form of randomization so that not all levels with the same training goals would feel the same, thus keeping the levels interesting for the user. Because of these two major reasons, Parameterization-Notch-Random has been chosen as a starting point for the generation of game levels for a game clone of Super Mario Bros.

Based on literature research on player modeling and adaptive procedural content generation, a flow chart can be made that shows the components needed for our adaptive procedural content generation environment for training a player's skill compared to a baseline adaptive procedural content generation environment like infinite adaptive Mario. This flow can be seen in Figure 3.1a and Figure 3.2b. The parts that are different between the base version and our adaptive version are marked yellow in the flow chart of the base adaptive environment.



(a) Adaptive version flow (b) Baseline version flow

Figure 3.1: Adaptive flow versus baseline version flow.

The main difference between our adaptive environment and a baseline version of the adaptive environment is that our adaptive environment targets the difficulty for an individual player instead of using default intervals to increase or decreasing the difficulty score. Our adaptive environment also makes use of gameplay, level, and player information to presents the players with challenging levels based on training goals that focus on increasing the user's game literacy in playing platformer games instead of generating completely random obstacles solely based on the difficulty score for the player to overcome.

The flow chart for our adaptive environment consists of the following steps:

1. Create or load player model;
2. Generate new part of level (chunk) based on this data;
3. Track progress of the player while playing the game;
4. Evaluate the player's result at the end of the generate part of the level;
5. Adapt the procedural content generation based on the outcome of the evaluation of the played part of a game level;
6. Then repeat steps 2 to 5 for each completed section or level;

The diagram shows that there are, however, still some unanswered questions.

- What should be done with a new player, whose proficiency in playing platformer (Super Mario Bros type) games has not been determined? This is called the hot cold problem.
- What should be done with players that get stuck in a certain part of a level?
- Can an adaptive procedural content generation game environment help in training players' game literacy?

3.4 Experiment

To answer the questions stated at the end of the previous section, a user-centered experiment will be conducted. After conducting the experiment, an empirical analysis of the results will be done. For the experiment, a user study in an adaptive procedural content generation game environment will be done, where the aim is to generate levels targeting the user's skill level. For the experiment, a group of participants are asked to play two variants of an adaptive procedural content generation platformer game. These games are meant to be clones of the popular game Super Mario Bros that will be developed for this experiment.

Two variants of the game are made to see if the participant's game literacy learned from one game can be used in a similar other game. This will be accomplished by letting the user play both versions of the game for a certain amount of time. While

the user is playing the game, some data regarding the user's performance will be tracked. When the player is done playing both versions of the game, the tracked data will be analyzed to see if the user's game literacy learned from the first game can be applied in the second game.

The games will employ the mechanics described in section 2.2. The notable differences between the two versions will be the visual graphics and the player character's physics. The visual graphics of the games are different, so the game expresses a visual distinction between the two versions of the game. There are two reasons for the differences in character physics between the two versions of the game. The first is that the mechanics described in section 2.2 are present in all platformer games, since these mechanics describe the core of what a platformer game is and thus it would not make sense to change those mechanics. The second reason is that it would be easier to add more versions and variants of the game if needed. This is because it would be easier to just change a few parameters for either the character physics or the level generation parameters than it would be to add or remove entire game mechanics.

The game should be designed as an Infinite Mario Runner type of game. This means that the game will forever generate small parts of Super Mario Bros-type of game levels, while the player is traversing through the game level. This is done to provide the player with an experience like that of playing an actual level of the game Super Mario Bros, while also being able to adapt each part of the generated level to the individual user's needs. The game needs to include the following major components, already discussed in section 3.3.1 and shown in Figure 3.1a, these are the following elements:

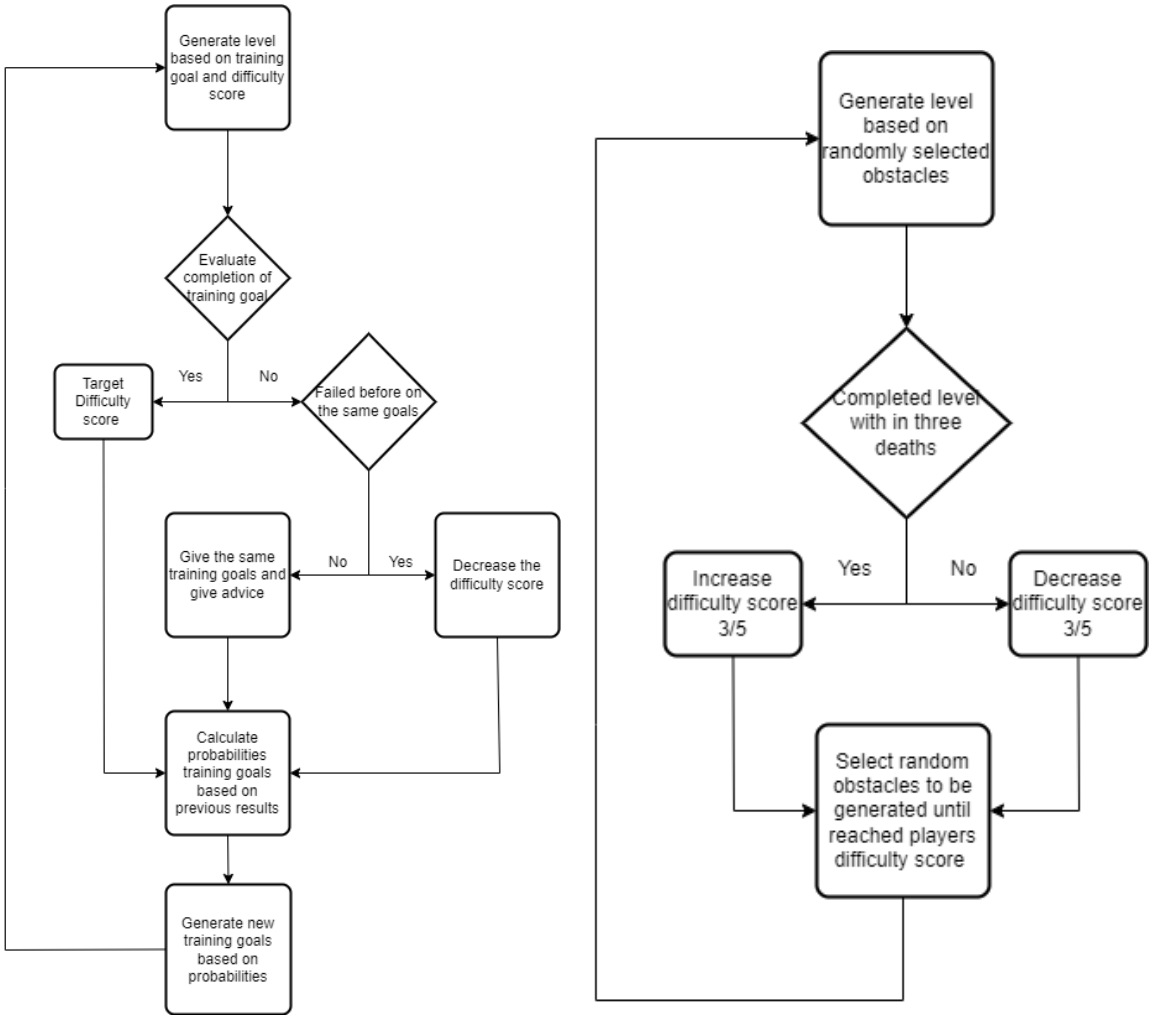
- The basic mechanics for a platformer-type game like Super Mario Bros, as shown in Table 2.2;
- Adaptive procedural content generation algorithm for a game of Super Mario Bros based on training goal;
- A player model that can determine if the difficulty of the training goals is either too high or too low;
- A way to generate training goals, based on player modeling;
- A way to evaluate the player based on the set training goals;

The first step of the design shown in Figure 3.1a is to check if the player has played the game before. If that proves to be the case, the user's data will be loaded in from the previous session and the game will be started from where the player left off. If the player has not played the game before, a new user will be created. The user will then be taught the basic movement and mechanics of Super Mario Bros in generated levels before adaptively targeting the player's skill level. Teaching every user, the base mechanics of platformer games is done to ensure that the user has the basic game literacy required to play platformer games.

The second part of the design shown in Figure 3.1a is tracking player and level data. User and level-related data will be tracked while the user is playing the game. This data will include:

- Number of deaths caused by enemies;
- Number of deaths because of failed jumps;
- How long it took the user to complete a level;
- The average velocity with which the player is moving through the level;
- Current estimated difficulty score of a level; this influences the number of paths and obstacles generated in the level;
- Training goals attached to the generated level;
- Play style of the user (speed running, collecting);

This data will then be used inside the evaluation step shown in Figure 3.1a to determine whether the player has completed the level or not. The outcome of the evaluation step is a true or false value that determines if the player has completed the level and the assigned training goals to that level.



(a) Adaptive environment player model

(b) Baseline adaptive environment version player model

Figure 3.2: Adaptive vs baseline player model.

After the evaluation step, the player model is used to target the user's difficulty score and to determine which training goals should be assigned for the generation of the next part of a level based on the outcome of the evaluation step. The flow of the player model for our adaptive environment and a example of the baseline adaptive environment can be seen in Figure 3.2a And Figure 3.2b. The baseline adaptive environment has one flow either increase or decrease the difficulty score based on the amount of player character deaths. While in the adaptive environment of our method the player model can increase and decrease the difficulty score based on the players performance. The player model can also take different types of actions to guide the player in improving their game literacy. when the user has failed the training goal, the player model can initiate one of the following actions:

1. Give the player a message on the screen with a tip on how to overcome an obstacle and let the user try again with the same training goals;
2. Give the player a message on the screen with a tip on how to overcome an obstacle and lower the difficulty of the training goals;
3. Give the player a message on the screen with a tip on how to overcome the obstacle the player is currently facing;
4. Regenerate a level if a player is unable to complete a level and lower the difficulty of the newly generated level;

When it is determined that a player can complete a level with reasonable ease, we can increase the difficulty drastically for the next generated level. When it is determined that a player can complete a level with some challenge and difficulty, we will only increase the difficulty slightly for the next part of a generated level.

With the different actions in place that the player model can take based on the user's performance, we make sure user's can never get fully stuck at a certain part in a level, answering the question: *What should be done with players that get stuck in a certain part of a level?* By faster increasing or decreasing the difficulty score based on a user's performance, we also reach content in the game faster that is around the skill level of the player and thus getting the player closer to a flow state [25].

The training goals that the player is presented with are based on which mechanics the player has failed, meaning that if the player has, for example, died multiple times at the hands of a certain enemy, the chance of a level being generated with that type of enemy would be really high. However, within the assignment step of new training goals, all mechanics or types of enemies are possible, only the mechanics that the player has failed more will have a higher probability of being generated in the upcoming levels. This is done so that not all levels feel the same for the user.

The player model selects different types of obstacles or combined obstacles until it has reached the player's current difficulty score. The selected obstacles are based on the training goals selected by the player model. Each obstacle in the game has a difficulty score assigned to it, indicating how hard it would be for a player to overcome. The procedural level generator is then responsible for the generation of levels with the obstacles and training goals provided by the player model.

The last part of the design seen in Figure 3.1a is the generation of the levels. This will be done using Parameterization-Notch-Random as a starting point for the generation of game levels, where the parameters are based on the training goals. The following parameters can be adjusted through the adaptive procedural content generation algorithm:

- Number of chasms to generate;
- Number of enemies to generate;
- Which type of enemies are allowed to be generated
  - Goomba;
  - Shell;
  - Flying shell;
- Number of platforms;
- Number of fire chains;
- Number of elevations (height of blocks) in the level;
- Difficulty score of the generated level, which influences the frequency of generated obstacles. It also leads to multiple ways through the levels [35] [47];
- Assigned training goals to the generated level;

The levels are generated in small parts called chunks. In the method chosen for this research, two types of chunks are defined: the training chunk and the cool down chunk. The training chunk is where the user will be challenged with certain training goals based on the player's skill level in playing platformer games targeted by the player model. A training chunk can present the user only with one new mechanic in each chunk. This creates an onboarding flow at the start of the game where the user can learn the mechanics of a game of Super Mario Bros in sequential order instead of having multiple new mechanics presented to the user all at once. The second type is the cool down chunk: here, the user will not face any significant challenge in the level. This is done for two reasons:

1. The first reason is that users can catch their breath and are not under continuous pressure. This is based on the research done into flow in games [6] [25], which states that a play under continuous pressure can have a negative impact on keeping a player optimally engaged.

2. The second reason is that the cool-down chunks provide the possibility to run calculations in the player model so that the user's skill level can be targeted before generating the next training part of the level.

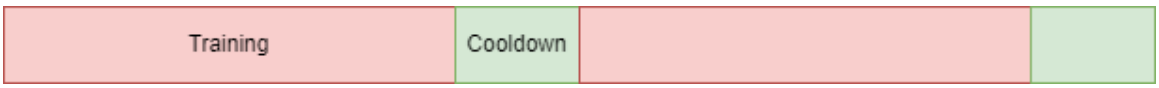


Figure 3.3: Flow of generation of levels.

The cool-down chunks are significantly smaller than the training chunks. The reason for this is that users might get bored if they are unchallenged for too long. Figure 3.3 shows an example of the general flow of the generation process of the level generator. Figure 3.4 shows an example of a generated level containing some obstacles generated based on training goals. In the top left corner of Figure 3.4, the training goals that are presented in the generated level can be seen.

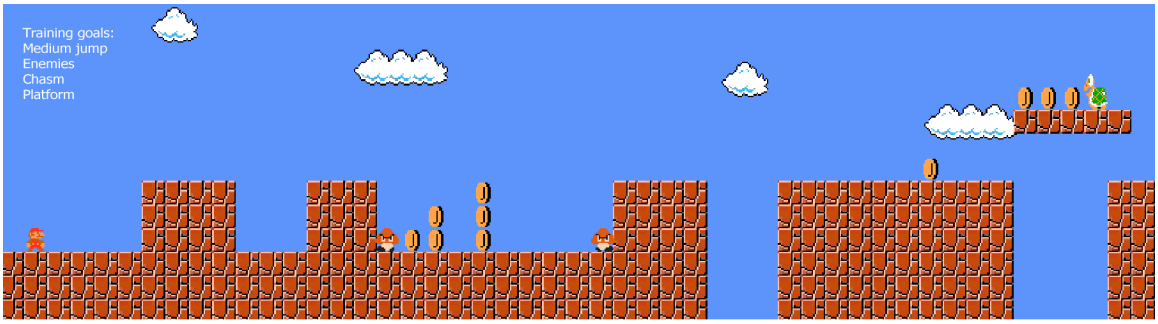


Figure 3.4: Example of training chunk.

After partaking in the experiment, the user will be asked some questions by filling in a questionnaire. The questions in the questionnaire will focus on the generated levels of the game and the player's experience while playing the game. From the information received from the experiment and the questionnaire, conclusions will be drawn to determine whether the presented method increases the game literacy of the user and to identify what the player liked about the method and what could be improved upon this method in future research.

3.5 Implementation details

The method has three main goals. The first one is to teach the player the basic mechanics of the game. The second one is to generate levels that target the user's current skill level in playing platformer games. The third one is to generate levels that focus on mechanics the user has difficulty with overcoming while playing the game.

This means this method needs to facilitate the way in which the user learns the basic mechanics of the game. The method also requires some way to track player data and to adapt the level generating process to target the user's skill level based on the tracked data. The method would need to decide what type of training goals should be assigned to users so that they can increase their overall game literacy in playing platformer games. The method should be able to generate new levels or parts of levels called chunks based on the training goals and player's skill level. This leads to the following flow of our method:

1. Tracking player data;
2. Analyzing the player's data to target the player's skill level and training goals;
3. Lastly, generating a new section of a level based on the targeted player skill level and training goals;

The result of each generated level should be a Mario clone game level that is the right difficulty for an individual player and contains game mechanics the user has difficulty overcoming while playing the game. The goal of playing these generated levels would be to improve the player's game literacy in playing platformer games.

3.5.1 Tracking player data

While the user is playing the game, data is tracked for each chunk the player is playing by tracking in-game events. The data of the five last played chunks the user has played is used to adapt to the player's skill level and adapt training goals accordingly. We use the last five chunks to determine the training goals and target the difficulty for the user, since it would be inappropriate to consider mistakes, users made before them. The information tracked for each chunk can be seen in Listing 3.1.

```
public class ChunkInformation
{
    public int jumpDeaths = 0;
    public int enemiesDeaths = 0;
    public int goombaDeaths = 0;
    public int shellDeaths = 0;
    public int flyingShellDeaths = 0;
    public int fireBarDeaths = 0;
    public int timeCompleted;

    public int totalCoinsInChunk = 0;
    public int totalCoinsCollected = 0;

    public int difficultyScore = 0;

    public int index = 0;

    public float averageVelocity = 0;

    public bool completedChunk = false;
    public bool outOfTime = false;

    public List<TrainingType> tranningTypes;

    public int GetTotalDeaths()
    {
        return jumpDeaths + enemiesDeaths + fireBarDeaths;
    }
}
```

Listing 3.1: Data tracked for each generated chunk.

A user's data is also tracked over the duration of an entire gameplay session, to solve the hot-cold problem when a user is playing the game for a second time and to start them off using game-play info from previous sessions. This data is tracked for each of the different versions of the game. The data tracked over an entire session can be seen in Listing 3.2.

```
public class GlobalPlayerResults
{
    public int HighestScoreVersionOne;
    public int HighestScoreVersionTwo;

    public int jumpDeathsVersionOne;
    public int jumpDeathsVersionTwo;

    public int EnemyDeathsVersionOne;
    public int EnemyDeathsVersionTwo;

    public int totalDeathsVersionOne;
    public int totalDeathsVersionTwo;

    public int fireBarDeathsVersionOne;
    public int fireBarDeathsVersionTwo;

    public int timeCompletionIntroductionVersionOne;
    public int timeCompletionIntroductionVersionTwo;

    public bool DidRegenerateLevelVersionOne;
    public bool DidRegenerateLevelVersionTwo;
    public bool DidFailTraingVersionOne;
    public bool DidFailTrainingVersionTwo;

    public bool DidSpeedRun;
    public bool DidCollectAllCoins;
}
```

Listing 3.2: Global data tracked for each version of the game.

The global data includes the highest difficulty score reached by the user in each version of the game. That data can be used as a starting point for the next session the player will play. Data regarding the user's gameplay style is also tracked, such as data indicating whether the user is speed running (completing levels as fast as humanly possible) or is behaving more like a collector collecting all the coins that are spread out across the level.

The data in Listing 3.1 and Listing 3.2 will be used for the evaluation of the method, after gathering all the results from the participants that partook in the experiment.

3.5.2 Player model

The purpose of the player model is to target training goals and determine a difficulty score for an individual user based on previous results the user has achieved in levels played beforehand. This is done using the tracked data specified in the previous section. The player model has the following flow:

1. Evaluate the training goals;
2. Adapt difficulty score based on player skill;
3. Adapt training goals to player skill;
4. Generate new training goals;

The subsequent sections will go into more detail regarding the implementation of these steps.

Evaluate training goals

After the completion of a generated part of a game level, the training goals associated with that level are evaluated. This is done by checking if the user has completed the level and the training goals associated with it. An example of how training goals are evaluated can be seen in listing 3.3, which shows that the player is allowed to make at least one mistake since any human can make an unintended error for several reasons. Users are also allowed to make slightly more errors if they are attempting to speed run [21] the levels, which is the concept of trying to complete the levels as fast as possible, which also increases the chance of making mistakes.

```
1  private bool DidCompleteEnemiesTrainingGoal()
2  {
3      var isSpeedRunning = _playerModel.IsSpeedRunning();
4      var enemyDeaths = _playerModel.chunkInformation.enemiesDeaths;
5
6      if (isSpeedRunning == false && enemyDeaths > 1
7      ||  isSpeedRunning && enemyDeaths > 2)
8      {
9          return false;
10     }
11         return true;
12 }
```

Listing 3.3: Example evaluate training goal.

The completion of training goals for each mechanic or combined mechanic in the game is also checked. A combined mechanic is a mechanic that consists of multiple training goals that need to be overcome together. For example, the training goal of enemies and platforms can be combined and would mean having an enemy that is patrolling on top of a platform.

Difficulty score

Every game mechanic has a difficulty score attached to it. The difficulty score reflects how difficult the mechanic is supposed to be for a player to overcome. The difficulty score influences the following parameters during generation:

- The amount of game mechanics placed in the generated level;
- The type of mechanics that are selected; a higher difficulty score results in more difficult mechanics being selected;
- The number of available paths the player can take in the level;

The difficulty scores for each mechanic can be seen in Table 3.2.

Training goals difficulty score	
Training goals	Difficulty score
Walking	1
Small jump	2
Medium Jump	2
Question Block	2
Chasm jump	6
Enemies	6
Platforming	6
Static obstacle	10

Table 3.2: Training goals and their difficulty score in a Super Mario Bros game.

The different mechanics can also be combined, which will result in a higher overall difficulty score. An example of a combined mechanic with an increased overall difficulty score is one where enemies walk on a generated platform: this combines the mechanic of enemies with the mechanic of platforming. The difficulty score for each mechanic is determined based on the number of actions the player must take to overcome it, and whether the action can result in a player's death. For example, walking only requires the player to perform one action, namely pressing one button. While

overcoming a chasm jump requires players to use their insight on when to jump and multiple inputs from the user to overcome the mechanic, it is also possible for the player to die if the user fails to complete the chasm jump mechanic.

Figures 3.5, 3.6, and 3.7 present the various levels generated with different difficulty scores and show that higher difficulty scores introduce more obstacles in the level and present different paths through the level because of the added verticality in the levels.

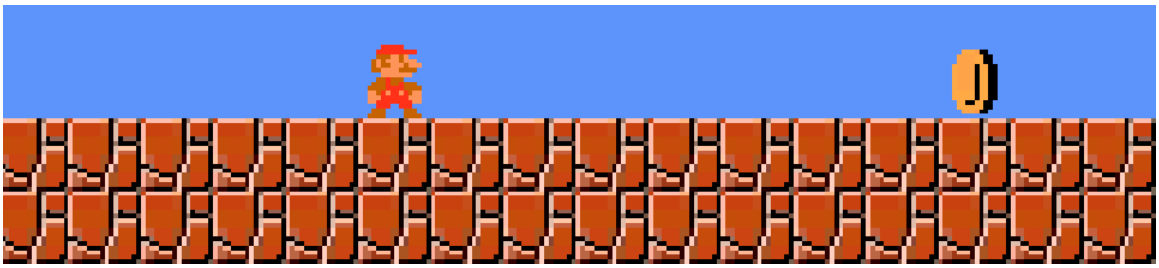


Figure 3.5: Example of level with difficulty score 1.

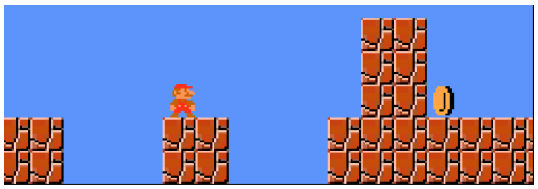


Figure 3.6: Example of level with difficulty score 20.

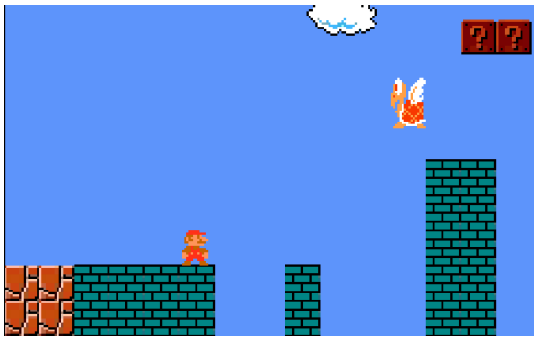


Figure 3.7: Example of level with difficulty score 40.

Listing 3.4 and 3.5 show the variables that influence the decision to increase or decrease the difficulty score. The difficulty score can be increased to a maximum of 100 and a minimum of one. Changing the difficulty occurs once a player has reached the end of a part of a generated chunk (part of a level). If the user has completed the training goals successfully, the increasing difficulty Listing 3.4, will be used; if the user has failed a training goal, the decreased difficulty Listing 3.5 will be used.

```
private int IncreaseDifficulty(ChunkInformation chunk)
{
    var increaseDifficulty = 0;
    var totalDeaths = chunk.GetTotalDeaths();

    increaseDifficulty += chunk.collectedAllCoins ? 5 : 0;
    increaseDifficulty += chunk.completedChunk ? 5 : 0;
    increaseDifficulty += chunk.timeCompleted < 5 ? 5 : 0;
    increaseDifficulty += totalDeaths == 0 ? 5 : 0;
    // 5.86 max speed walking.
    increaseDifficulty += chunk.averageVelocity > 5.86f
    && totalDeaths < 3
    && timeCompleted < 4
    ? 10
    : 0;

    return increaseDifficulty;
}
```

Listing 3.4: What influences increasing the difficulty score.

```
private int DecreaseDifficulty(ChunkInformation chunk)
{
    var decreaseDifficulty = 0;
    var totalDeaths = chunk.GetTotalDeaths();

    decreaseDifficulty += chunk.completedChunk ? 0 : 5;
    decreaseDifficulty += chunk.timeCompleted < 30 ? 0 : 5;
    decreaseDifficulty += totalDeaths > 5 ? 10 : 0;

    return decreaseDifficulty;
}
```

Listing 3.5: What influences decreasing the difficulty score.

The time it takes the user to complete the level, the average velocity with which the player character moves through the level determines if the user is trying to complete the levels as fast as possible, also known as speed running [21]. When the user is speed running, the penalty for failing the training goal is less severe than if the player is not speed running. When the user successfully completes the training goals attached to a level and has not been speed running the level, it will result in a higher increase in the difficulty score. The difficulty score will also increase more substantially if the player has completed the level with zero player deaths and if the player has collected all the collectibles (coins) and has thus taken more risk in completing the level. By

allowing different intervals in the amount of increase or decrease, of the difficulty score we target the user's skill level in playing platformer games faster than using default intervals in the increase or decrease, of the difficulty score thus considering the question: *What should be done with a new player, whose proficiency in playing platformer (Super Mario Bros type) games has not been determined? This is called the hot cold problem.* that was stated at the end of Section 3.3.1.

Adapt training goals and difficulty

Training goals are targeted by calculating a distribution based on the data specified in the previous sections for each mechanic. This distribution is based on which mechanics the user has failed in previously played parts of the game. For example, if a user has died five times at the hand of enemies and died once because of falling through a gap in the level, the training goal enemies will be five times more prevalent in the distribution than the mechanic chasms (gaps in the world). There are lower and upper bound values in the distribution between mechanics, so that at least every mechanic has a chance to be present in a newly generated level. This is done so that the levels have some randomization in them, so the user doesn't get bored from continuously playing the same type of level. From the created distribution of mechanics, we take x random training goals until we have reached the targeted difficulty score for a specific user. In listing 3.6 we can see a version of the flow discussed in the text above.

```
int currentDifficultyScore = 0;
List<TranningType> tranningTypes;

CalculateDistribution();

while (difference > 0)
{
    // Difference between targeted and current selected mechanics
   difficulty.
    var difference = targetDifficultyScore - currentDifficultyScore;
    // Get one mechanic from the distribution.
    var type = MechanicFromDistribution();

    tranningTypes.Add(type.tranningType);
    currentDifficultyScore += type.difficultyScore;
}
return tranningTypes;
```

Listing 3.6: Simplified version of targeting training goals.

3.5.3 Adaptive procedural content generation

The adaptive procedural content generation is responsible for the generation of the levels based on the training goals and difficulty score generated by the player model. Each training goal corresponds with one or more obstacles/mechanics in the level.

The level generator generates an x by y grid and from there on generates obstacles within the level based on parameters set by the player model. The level is generated in the following stages:

1. Generate Floor;
2. Generate elevations in the level;
3. Generate chasms in the level;
4. Generate platforms in the level;
5. Generate enemies in the level;
6. Generate static obstacles in the level;
7. Generate Coins in the level;

Obstacles are placed in random positions within the grid based on some rules that ensure that the level is playable and looks believable. Some exceptions are in place for combined mechanics. For example, with the combined game mechanics of platforming and enemies, firstly an attempt is made to find an existing platform created in the level and an enemy is spawned on that already existing platform. The same is done for the mechanic platforming and chasms: an attempt is made to find an already generated platform in the level and a chasm is generated underneath the platform.

The adaptive procedural content generation is also responsible for making sure the levels are playable. While generating obstacles randomly in the world, some rules have been put in place to make sure the levels are playable. These rules are listed below:

1. The difference in height between two neighboring tiles can be a maximum of the highest jump height possible for the player character;
2. A chasm can only be a maximum of the furthest possible jump position in the X direction by the player character;

3. The random obstacles being generated need to be within the grid size;

4. Each tile in the generated world can contain one mechanic, so we do not have overlapping elements on the map;

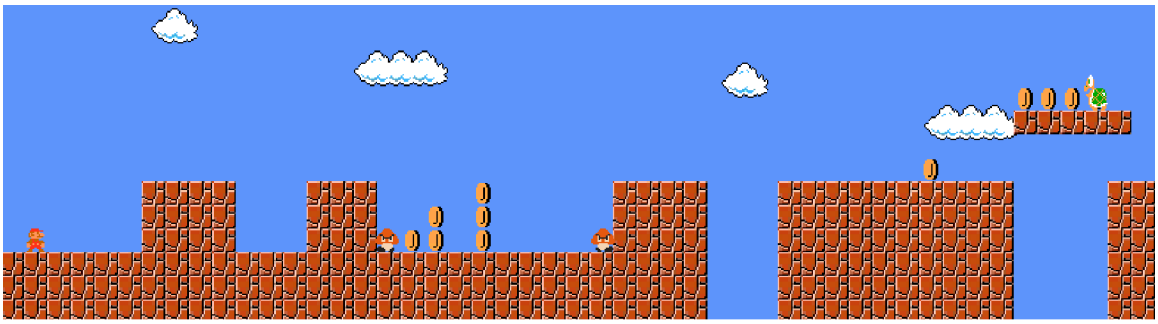


Figure 3.8: Example of generated level.

Figure 3.8 shows the result of a generated level with a difficulty score of 40 from our discussed method. This level contains the following training goals:

- Platform + enemies + chasm;
- Medium elevation;
- Medium Elevation + enemies;
- Chasm jumps;

To answer the last question stated at the end of Section 3.3.1, *Can an adaptive procedural content generation game environment help in training players' game literacy?* We will do a user-centered experiment comparing our adaptive procedural content generation environment used for training the user in platformer game literacy against a basic infinite Mario level generator with scaling difficulty. The results of this experiment will be discussed in the next Section.

Chapter 4

Results

This chapter discusses the results of the developed games. To evaluate the games, a user testing period was scheduled, during which the testers played the games and filled out a questionnaire. In addition to the questionnaire, the users' in-game behavior was objectively analyzed from data gathered during the gaming session. The participants were volunteers with different game experience backgrounds, the only condition being that they had not played the game before, meaning testers who had played the game during the development phase to help test the mechanics and the game's difficulty were not eligible.

4.1 Experiment process and goals

For our experiment we had two different types of environments, the first one being the adaptive procedural content generation game training environment using adaptive methods to target training goals and the user's difficulty score as discussed in Section 3. The second one being a baseline infinite Mario runner that increases or decreases difficulty with a standard interval and randomly selects obstacles based on the user's current difficulty score.

In the experiment, users first were presented with either our adaptive procedural content generation game training environment or the baseline adaptive procedural content generation training environment. Participants played only one of the two environments, so the results would not be influenced by learning game literacy from the other environment. Both the base Mario runner and the adaptive Mario runner had two different types of training games that would each be played for 5 minutes.

Note that when we talk about environments we are talking about the adaptive or baseline method implemented for the experiment. And when we are talking about game version we mean the different version of a clone of the popular game Super Mario Bros with the second version being slightly altered regarding the physics and visuals.

Each different type of training game is slightly altered from the other version in terms of physics and game visuals. The goal is to verify if a user can use the game literacy learned from playing the first training game in another similar training game, which is the second training game. The intention is to verify this by tracking gameplay and level generation data while the player is playing the game. After playing the game, the participant is also asked to fill in a questionnaire regarding their experience with previous versions of Super Mario Bros and the environment they just played.

4.2 Baseline adaptive environment

First we will show the results for the baseline environment of the game. This is the version of the game that does not use adaptive methods to target training goals for the users or try to target the user's difficulty score. Instead, this version increases or decreases difficulty with standard intervals based on if the user can complete the level within 3 player character deaths. The baseline adaptive environment flow can also be seen in Section 3 and illustrated in Appendix A.2 and Appendix A.3. While participants are playing the game, player and level generation data is tracked, as discussed in section 3.5.1. Some results on individual user level can be found in Appendix A.7, However some statements made in this section will be based on the data found in Appendix A.7.

The highest difficulty score reached in each version of the game is tracked and considered. A participant can reach a minimum of one and a maximum score of one hundred. The results can be seen in Table A.6. Table A.6 shows that overall, players performed slightly better or slightly worse in the second version of the game compared to the first version of the game. It is important to note that all players first played version one and then version two. The results show that most users stay around the same difficulty score between the two versions.

Version one and two statistics							
Version	Min	Max	Mean	Median	Mode	SD	VAR(x)
1	17	38	26	25.5	30	7	49
2	19	38	27.5	25	22	6.8	46

Table 4.1: Statistics of highest difficulty score between the two versions of the game.

A paired T-test based on the highest difficulty scores reached between the two versions of the game was also done, using a paired T-test, since data of two groups containing the same population in different scenarios is being compared. The results of the paired T-Test are: The two-tailed P value equals 0.2159. By conventional criteria, this difference is considered to not be statistically significant. The T-test was calculated using the mean presented in A.1. The mean of Group One minus Group Two equals -1.67 95 percent confidence interval of this difference: From -4.46 to 1.13 Intermediate values used in calculations: t = 1.313 and df = 11. The standard error of difference is 1.216. This means there is no clear difference in learning between session one and two.

Data regarding player character deaths was also tracked, which corresponds with the different mechanics generated in a level for the user. Table 4.5 shows the results related to causes of player character death and how many times they occurred.

Player character deaths				
Version	Total	Enemies	Jump	Fireball
1	95	41	40	14
2	91	37	41	13

Table 4.2: The number of times the player character has died and to which mechanics.

The results show that around the same number of player deaths occurred in version one compared to version two.

4.2.1 Survey

After playing both versions of the game, the participants were asked to fill in a questionnaire about their experience with the baseline procedural content generation game training environment. The questionnaire was filled in anonymously by each participant.

The first question was: *How much experience do you have in playing the game Super Mario Bros?* This question was asked to determine how skilled the overall group of participants initially was in playing Super Mario Bros.

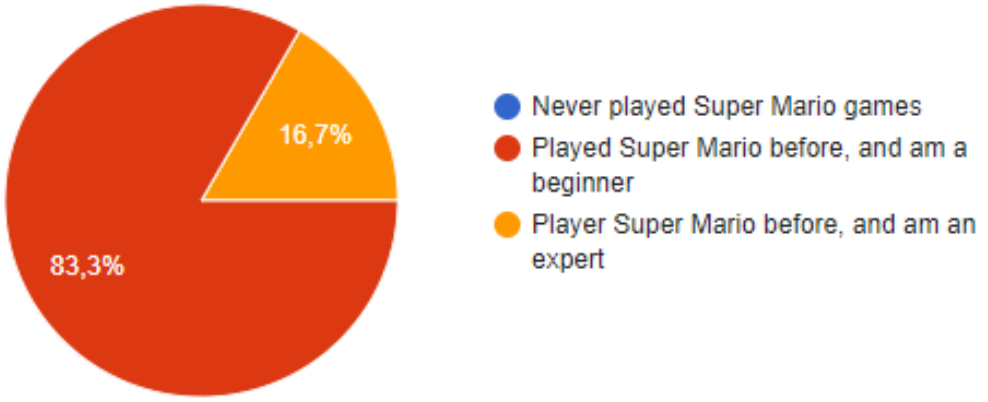


Figure 4.1: Played Super Mario before and I am an expert.

Figure 4.1 shows that two participants consider themselves experts in playing Super Mario Games. Ten participants mention having played Super Mario Bros before and consider themselves beginners. Zero participants stated that they had never played Super Mario Bros before and are thus new to the game.

The second and third questions asked the participants how they felt about the difficulty of the content at the beginning and the end of the game, using the five-point Likert scale as measurement. Question two was phrased as follows: *How challenging would you describe the content at the beginning of the level?* The result can be seen in Figure 4.2.

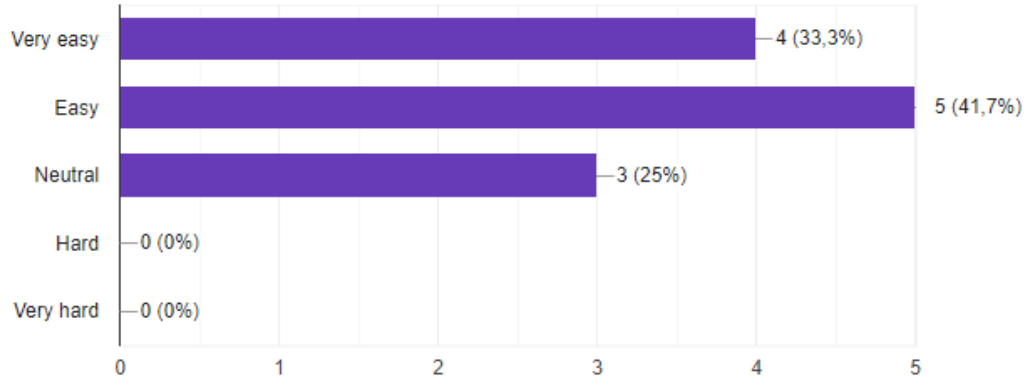


Figure 4.2: Difficulty of the game at the start of the game stated by the participants.

The results show that for most of the participants, the difficulty level was seen as being easy at the beginning of the game.

Question three was phrased as follows: *How challenging would you describe the content at the end of the level?* The result can be seen in Figure 4.3

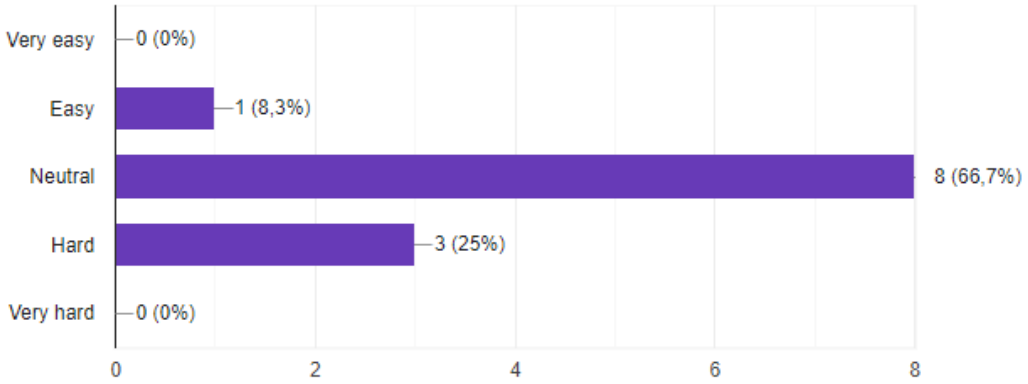


Figure 4.3: Difficulty of the game at the end of the game stated by the participants.

The results show that one participant interpreted the content easy. Eighth participants found the challenges the game presented neutral in terms of difficulty. Three participants considered the content as hard and zero participants considered the content as very hard.

The fourth question asked the participants about their initial impression of the game. It focused on whether the player felt bored, neutral, or intensive during the game

play. This question was asked to ensure that the game does not fall into the boredom or too intensive side of the flow channel. Question four was stated as follows: *What impression did the game have on you?* The results can be seen in Figure 4.4.

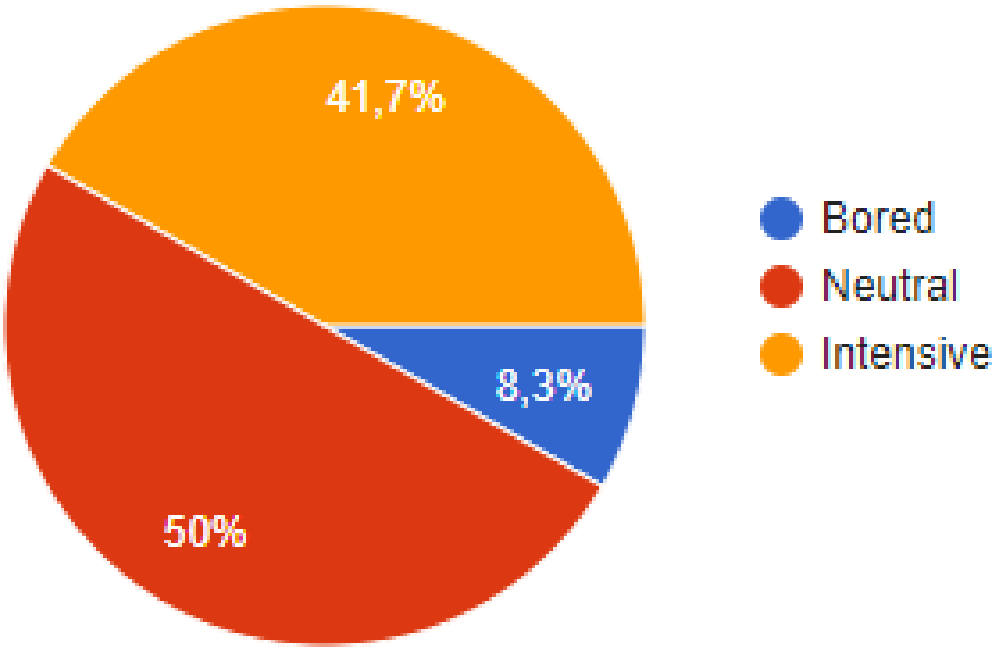


Figure 4.4: Initial impression during gameplay following flow.

Out of the group of participants, six users felt neutral while playing the game. Five users felt intensive during gameplay and one user felt bored while playing the game.

The fifth question focused on the feelings the game evoked in the users, and specifically on whether the game evoked a negative, neutral, or positive feeling in the user. Question five was phrased as follows: *what feeling did the game give you?* Figure 4.5 shows the results.

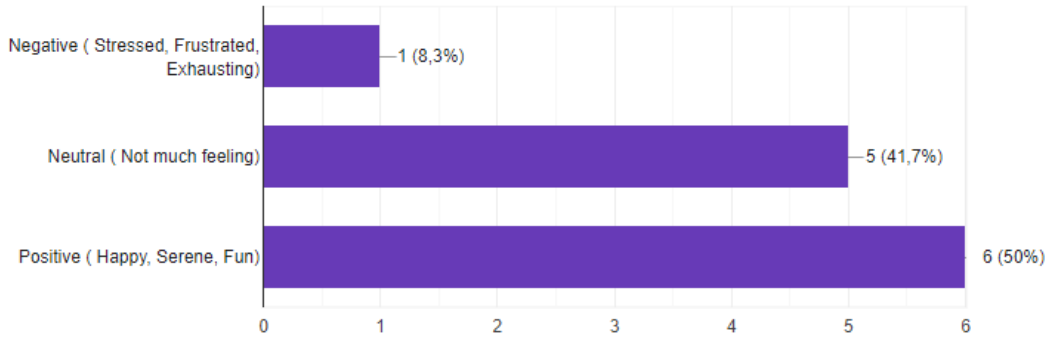


Figure 4.5: Feeling game invoked on users.

The results show that in most participants, the game evoked a positive or neutral feeling. One user experienced negative feeling during gameplay.

4.3 Adaptive training environment

Now we will show the results for the adaptive procedural content generation environment as discussed in section 3. This is the environment that does use adaptive methods to target training goals for the users and try to target the user's difficulty score in playing platformer games. The difficulty score is targeted by increasing or decreasing the difficulty score based on the player's performance in playing a game level, instead of using default intervals for increasing or decreasing the difficulty score after completion of a chunk (part of a level). While participants are playing the game, player and level generation data is tracked, as discussed in section 3.5.1. Some results on individual user level can be found in Appendix A.6, However some statements made in this section will be based on the data found in Appendix A.6.

The highest difficulty score reached in each version of the game is tracked and considered. A participant can reach a minimum of one and a maximum score of one hundred. The results can be seen in Table A.6. Table A.6 shows that overall, players performed better in the second version of the game compared to the first version of the game. It is important to note that all players first played version one and then version two. The results show that only two users performed slightly worse in the

Version one and two statistics							
Version	Min	Max	Mean	Median	Mode	SD	VAR(x)
1	10	85	41	36	30	21.5	463
2	15	100	61	65	65.7	23	534

Table 4.3: Statistics of highest difficulty score between the two versions of the game.

second version of the game than in the first version, with a score of 80 in the first version and 75 in the second version. There was one player who was able to reach the highest difficulty score of 100.

A paired T-test based on the highest difficulty scores reached between the two versions of the game was also done, using a paired T-test, since data of two groups containing the same population in different scenarios is being compared. The results of the paired T-Test are: The two-tailed P value equals 0.0001. By conventional criteria, this difference is considered to be statistically significant. The T-test was calculated using the mean presented in A.1. The mean of Group One minus Group Two equals 18.76 95 percent confidence interval of this difference: From 4.78 to 32.74 Intermediate values used in calculations: $t = 2.7121$ and $df = 40$. The standard error of difference is 6.918.

Both versions of the game start with an introduction to the game mechanics. This is a small section of the game that is meant to teach the player the base mechanics of the game. Table 4.4 shows the difference in statistics regarding the introduction. It should be noted that the introduction for both games has the same flow.

Statistics base mechanics				
Version	Completed	Failed	Mean	standard deviation
1	21	1	67.1	98.3
2	22	0	36.0	15.2

Table 4.4: Statistics teaching base mechanics.

The results show that one participant was incapable of completing the introduction in version one but was able to do so in version two of the game. The data also shows that participants were faster in completing the tutorial in the second version of the game. It also shows that there is a significant difference in the standard deviation between the two versions. This is due to the range being higher in version one than

in version two since one player spent five minutes in the tutorial in version one.

Data regarding player character deaths was also tracked, which corresponds with the training goals generated for the user. Table 4.5 shows the results related to causes of player character death and how many times they occurred.

Player character deaths				
Version	Total	Enemies	Jump	Fireball
1	196	104	79	13
2	224	100	106	18

Table 4.5: The number of times the player character has died and to which mechanics.

The results show that in the second version of the game more player character deaths occurred than in the first version. It also shows that in the second version, more players died from jumping and platforming than from enemies, while in version one it is the other way around.

4.3.1 Survey

After playing both versions of the game, the participants were asked to fill in a questionnaire about their experience with the adaptive procedural content generation game training environment. The questionnaire was filled in anonymously by each participant.

The first question was: *How much experience do you have in playing the game Super Mario Bros?* This question was asked to determine how skilled the overall group of participants initially was in playing Super Mario Bros.

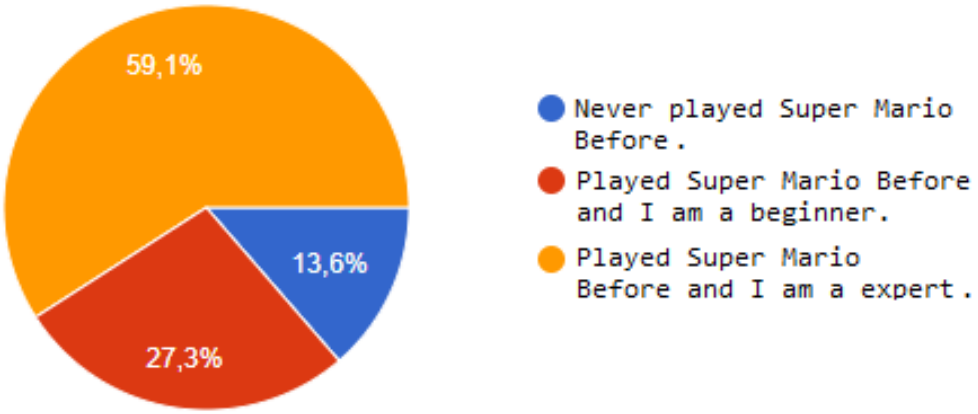


Figure 4.6: Played Super Mario before and I am an expert.

Figure 4.6 shows that 13 participants consider themselves experts in playing Super Mario Games. Six participants mention having played Super Mario Bros before and consider themselves beginners. Three participants stated that they had never played Super Mario Bros before and are thus new to the game.

The second and third questions asked the participants how they felt about the difficulty of the content at the beginning and the end of the game, using the five-point Likert scale as measurement. Question two was phrased as follows: *How challenging would you describe the content at the beginning of the level?* The result can be seen in Figure 4.7

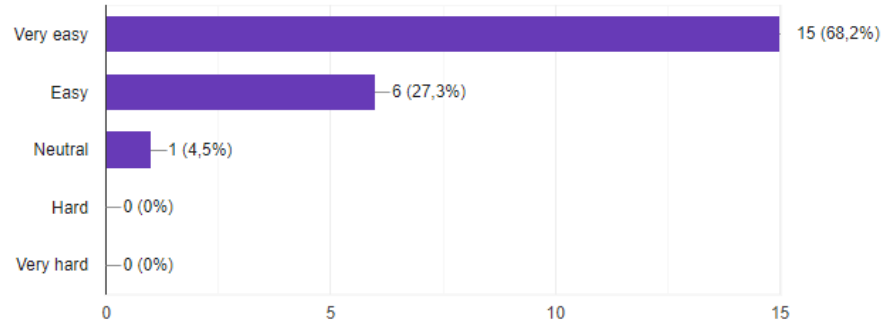


Figure 4.7: Difficulty of the game at the start of the game stated by the participants.

The results show that for most of the participants the difficulty level was seen as being easy at the beginning of the game.

Question three was phrased as follows: *How challenging would you describe the content at the end of the level?* The result can be seen in Figure 4.8

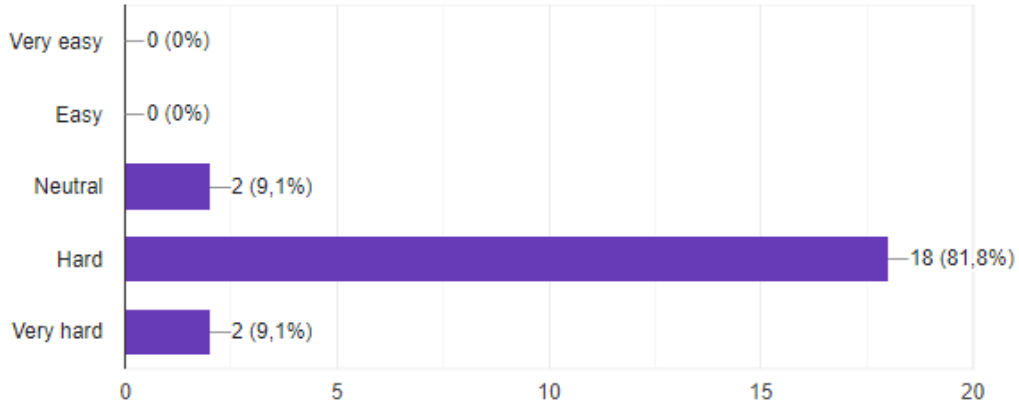


Figure 4.8: Difficulty of the game at the end of the game stated by the participants.

The results show that the game can offer some difficulty to users of all skill levels.

The fourth question asked the participants about their initial impression of the game. It focused on whether the player felt bored, neutral, or intensive during the game play. This question was asked to ensure that the game does not fall into the boredom or too intensive side of the flow channel. Question four was stated as follows: *What impression did the game have on you?* The results can be seen in Figure 4.9.

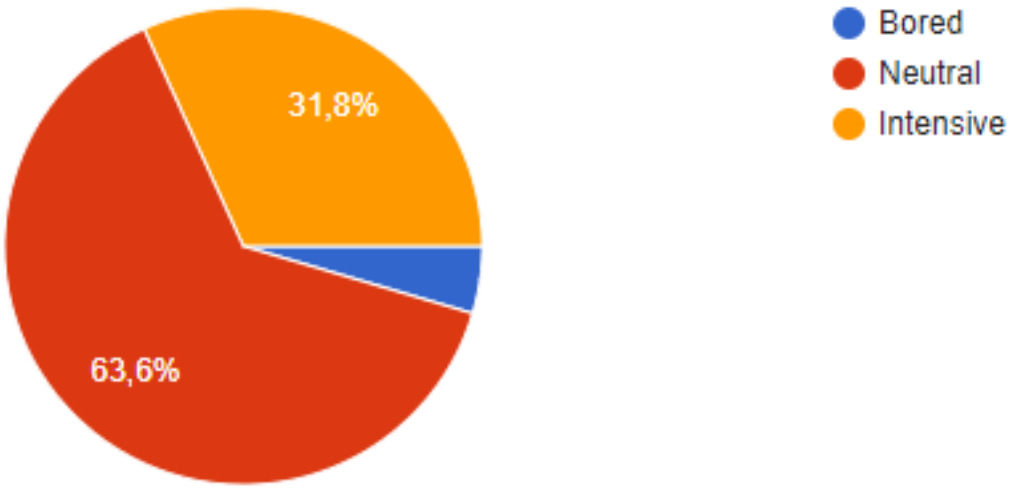


Figure 4.9: Initial impression during gameplay following flow.

Out of the group of participants, fourteen users felt neutral while playing the game. Seven users felt intensive during gameplay and one user felt bored while playing the game.

The fifth question focused on the feelings the game evoked in the users, and specifically on whether the game evoked a negative, neutral, or positive feeling in the user. Question five was phrased as follows: *what feeling did the game give you?* Figure 4.10 shows the results.

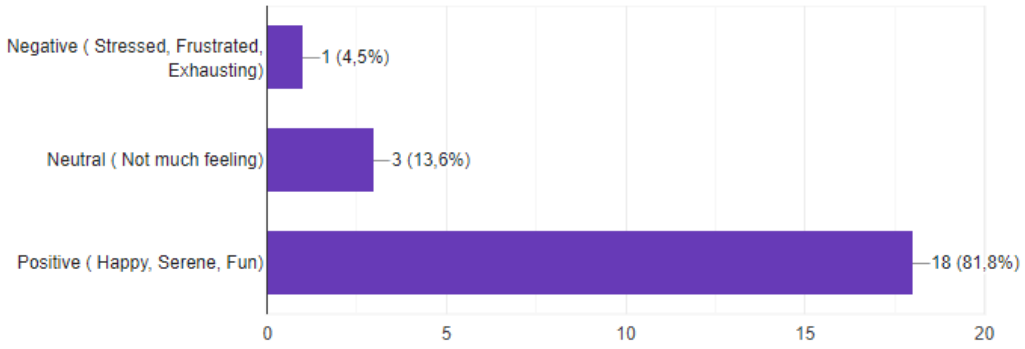


Figure 4.10: Feeling game invoked on users.

The results show that in most participants the game evoked a positive or neutral feeling. One user experienced negative feeling during gameplay.

The last question the participants were required to fill in focused on whether the user would be motivated to use adaptive procedural environment games to increase their skills in playing video games. The question was phrased as follows: *After playing the game for a while, it uses adaptive methods to change the difficulty of the game and suggest training goals in certain game mechanics the player can improve in, for each individual participant. Would playing video games like the ones you just played motivate you to increase your skills in playing video games?* The results can be seen in Figure 4.11.

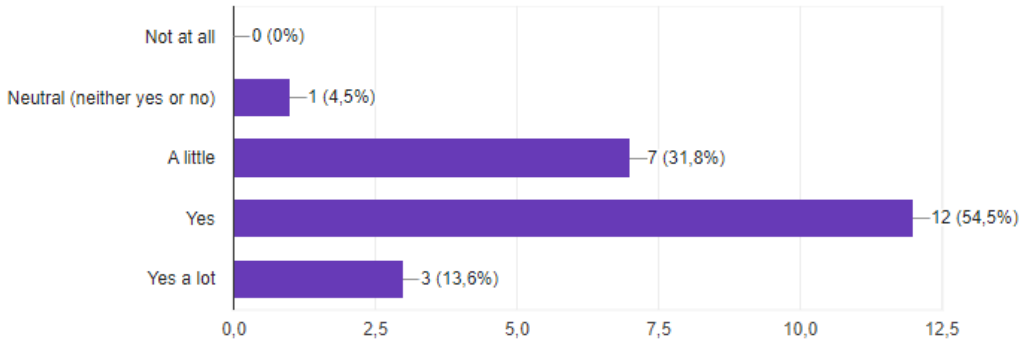


Figure 4.11: Motivation to use adaptive game environments for increasing player skill.

The results show that one player noted that they felt it would neither discourage them nor encourage them to use procedural game environments to increase their skill in playing video games. The other 22 participants would be motivated a little to a lot to use an adaptive procedural game environment to increase their skill in playing video games.

4.4 Differences between baseline and adaptive version

In this section we will present the major difference found in the results between the base and adaptive game environments. When we are talking about environment we talk about the base version and the adaptive version. When we are mentioning games

we are talking about the two different type of games in regards to physics and visuals for both the environments.

4.4.1 Gameplay results differences

In Table 4.6 we can see the results for the base version marked with type b and adaptive version marked with type a regarding highest difficulty score reached between the different games.

Base versus adaptive statistics								
Type	Version	Min	Max	Mean	Median	Mode	SD	VAR(x)
b	1	17	38	26	25.5	30	7	49
b	2	19	38	27.5	25	22	6.8	46
a	1	10	85	41	36	30	21.5	463
a	2	15	100	61	65	65.7	23	534

Table 4.6: Statistics of highest difficulty score between the adaptive and base versions of the game.

From the results we can notice that on average the adaptive version participants are enabled to reach higher difficulty scores compared to the base version. We can also notice that there is a big difference between the max score that has been reached between both versions.

We did a T-test based on the highest score both for the base version and the two sub games and the adaptive version and its two sub-games. The results for were the adaptive version are 0.001 and the base version are 0.2159 we can see that there is a significant difference in the adaptive version with it being lower then 0.005 with a score of 0.001. While there is no significant different between the two games in the base version with it being higher then 0.005 with a score of 0.2159. Indicating their is a higher learn rate in the adaptive version in regard to reaching higher difficulty scores.

4.4.2 Survey differences

In this section we will discuss the major differences between the base and adaptive version in regards to the survey filled in by the participants.

The question regarding *How challenging would you describe the con- tent at the end of the level?* we can see the results of the base (b) and adaptive (a) version in Table 4.7.

Base versus adaptive statistics					
Type	Very easy	Easy	Neutral	Hard	Very Hard
b	0	1	8	3	0
a	0	0	2	18	2

Table 4.7: Difference between difficulty.

The results show that more people experiences difficult content in the adaptive version compared to the base version. While in the base version more people experienced neutral difficulty content.

The question regarding *What feeling did the game give you?*we can see the results of the base (b) and adaptive (a) version in Table 4.8.

Base versus adaptive statistics			
Type	Negative	Neutral	Positive
b	1	5	6
a	1	3	18

Table 4.8: Feeling the game invoked on participant adaptive vs baseline

From the results we can see more people have percentage wise a positive experience in the adaptive version compared to the base version.

4.5 Difference between player types

In the game, each player is also assigned a variable that indicates whether the user has a play style that corresponds with speed running, collector, or casual [2]. speed runners being players who wish to complete the level as fast as possible, and collectors being users willing to collect all the collectibles (coins) in the levels. Finally, casual players are those who are neither collectors nor speed runners and are content with just overcoming the challenges of the game. Table 4.9 shows the distribution of collectors, speed runners and casuals detected in this manner during the experiment. The results show that, on average, speed runners reach a higher difficulty score compared to

Difference between player types				
Type	Total	Highest Score	Total Deaths	Failed Training goal
Casual	8	70	100	3
Collector	6	100	145	4
Speed runner	10	100	284	6
Combo	2	100	109	2

Table 4.9: Statistics for the different player Types presented in this method.

casuals and collectors. However, they also show that, on average, speed runners make more mistakes in the game, based on the number of deaths and the training goals they failed. This can also be seen on an individual player level in Appendix A.6. Note that from Appendix A.6 shows that some players are marked as being both a speed runner and a collector.

4.6 Player feedback

The last question was optional and was an open question where participants could give their overall impression of the game. The question was phrased as follows: *What kind of impression gave the adaptive game?* Most responses indicated that the game had left a good overall impression. Below are some of the more noteworthy comments received from the participants, which will be discussed in the next section.

1. On higher difficulty scores the difficulty can ramp up quite significantly even if the difference in difficulty score is only slightly higher or lower.

2. I like being challenged in a game and made me want to perform better. When a game is too easy, you are not really paying attention and just cruising through. I think it will be important to find the right balance so that when it's getting too difficult it also adapts to making it easier.

3. Apart from that the impact of the adaptive difficulty is very hard to gauge because I cannot make a comparison with how it would have played if it were harder/easier, and I would not want to play a game like that multiple times to find out. Because of this the message "Good job! Difficulty increasing" felt useless to me because I do not know what changes and games always get more difficult the further you get into a level. The system would be better suited as

a subtle background thing that can alter difficulty without the player knowing it.

4. In my opinion, it altered the level rather fast. In the first few stages that wouldn't be a problem. At the end however, it became pretty savage. Overall it was a pretty nice way to get used to the pc controls instead of an old school controller!

All the responses users gave to our game can be seen in Appendix A.1.

Chapter 5

Discussion

The results indicate that the chosen adaptive procedural content generation method for training player skill can train people in game literacy, since there is a clear statistical difference in results between version one and version two of the game played by the participants of this experiment. It can also be said that the game is able to generate various levels with various amounts of difficulty. This supports the theory that adaptive procedural content generation algorithms can help in training a person's game literacy.

It can also be noted from the results that the player character died more in the second version of the game than in the first version in the adaptive environment. This can be due to multiple reasons, one being that as people get further into the game, they are challenged more at higher difficulty scores. It can also be that people are more willing to take risks after getting used to the type of game they are playing having already played version one of the game. It can also be noted that participants who are willing to take more risks are able to get further into the game than people that are less willing to take risks. The data supports this conclusion, since participants that are marked as speed runners reach higher difficulty scores than those who are not. It should be noted that this can also be because participants that are speed running are more familiar with Super Mario Bros games. This can, however, not be determined from the data since this familiarity was not tracked in the game.

5.1 Difference between base and adaptive environment

For the experiment, we created two different types of procedural content generation environments, the base and adaptive environment. The adaptive procedural content generation game environment is as proposed in Section 3 that uses adaptive methods to target the user's difficulty score and generates levels based on training goals. The second method being a base procedural content generation environment that increases the difficulty of the game with default intervals after completing a section of a level.

For the adaptive environment there is a statistical difference for the highest score reached between version one and version two of the game, while there is not one for the base environment of the game. This indicates that the adaptive method is better suited for training the user in game literacy for playing player former games than the base environment.

This can be due to multiple reasons. One reason being the adaptive method uses adaptive methods to target the user's difficulty score faster. While the base version slowly builds up the difficulty score with default intervals. This can lead to people not reaching their skill level in the base version while they do in the adaptive version, and thus not being challenged by the content of the game. This can also be supported by seeing the results of the survey where participants mentioned that the difficulty of the content for the adaptive environment was mostly considered hard at the end of the game, while with the base version most participant mentioned neutral difficulty in the game's content.

A second reason can be that the adaptive version focuses on generation obstacles in the level that the user has difficulty with overcoming, and thus increasing the part of the user's game literacy in which the user has the most to learn. While the base version randomly generates obstacles in the game levels. The adaptive version also presents the user with tips on how to overcome certain obstacles and adjust the difficulty of the content accordingly, while the base version does not.

The results indicate that our adaptive method is better suited at training a participant in game literacy than the base method. We should, consider that the five-minute time frame could not be enough time for the base method to reach the difficulty level

of a certain participant's skill in playing platformer games. This could also explain the reason of higher skill level participants not increasing their game literacy between version one and two of the game. However, we could also notice from the result that players stuck on lower difficulty scores in the base environment did not improve much or at all in their second gameplay session while users did in the adaptive environment. We should also note that the different sizes of participants group could have played a part in this, and a large population for both environments might give slightly different results.

5.2 Generalisation of the method to other games

In this section we will discuss how parts of our method could be adapted in to different games.

1. Presenting different mechanics one by one is a good way of teaching someone the different mechanics in a game. In our adaptive method, each generated chunk can only present one new mechanic or one new combined mechanics to the player. This allows for the player to get a gentle introduction to the new mechanics of the game and how to overcome those mechanics.

2. Adapting the difficulty to the user's skill level or slightly above can increase the learning rate for a user, as can be seen from the results of this study. As we can see, a single discrete parameter controlling the type and amount of obstacle in a level can be a successful way of doing it.

3. Allowing a player model to give the user feedback while playing the game can be an effective way to instruct users in how to overcome certain challenges they otherwise would be stuck at.

4. Targeting the user's difficulty score in playing a certain game is an effective way of getting the user fast to a point in the game they can increase that game literacy in playing a certain video game. We can see this from our study where users that did not have the targeting of difficulty score in the baseline version would earlier be bored in playing the game due to the lack of challenging content.

The downside of implementing our method for other games is that the developer needs to be well learned in the mechanics of the game and how the mechanics influence the difficulty in the game.

5.3 Generation of the results to other games

In this section, we discuss some results and observations done during the experiment, that could also be of use when developing games.

Results from the questionnaire indicate that fast alteration of difficulty can lead to unreal feeling for playing video-games. For our goal, that was not much of a problem, since our goal was to create an adaptive game environment with the use case of training the user's game literacy. However, if you want to use our adaptive method solely for the use case of creating a game with difficulty scaling, it might be smart to consider smoothing the curve for increasing and decreasing the difficulty.

Results also indicate that providing the user only with one new game mechanic at a time has a positive influence on the learning rate, especially in the early part of the game. Results also indicate the opposite, providing the player with more than one new mechanic at a time can significantly slow down the learning process, since in most cases it is harder to overcome parts of a level that introduce more than one mechanic at once. In the baseline environment where it could happen that one or more new mechanic was introduced, players got stuck longer than in our adaptive environment that would introduce the game mechanics one by one.

From the results, we could also notice that people are more tolerant to content that was slightly above their skill level than game content that was slightly below their skill level. Content that was slightly below their skill level specially for larger number of times could lead to an increase in boredom and in turn would lead to less effective learning rate.

5.4 Participants' feedback

The last question in the questionnaire was an open question to the participants, asking them to share their experience of playing the game. In this section, some remarks made by the participants are discussed. Overall, the feedback of participants was positive for the method that was created. All feedback given by the participants can be read in Appendix A.1.

5.4.1 Learning keyboard and mouse controls

Some participants noted that they only play games on the console with a controller as input device. Some of those participants noted that the method presented in this thesis is a good way of learning how to use computer controls using a keyboard and mouse for playing video-games and thus gain game literacy in playing video games on a computer.

5.4.2 Method as a background system

One participant stated that the method used in the experiment would be better suited as a background system. Meaning that the user doesn't notice that there is a system active altering the difficulty of the game. In a game developed in normal circumstances this would be the most desirable option, since the method would then be less obvious for the players, but in this case the focus was on testing the system and specifically on using it as a tool for training a player's game literacy.

5.4.3 Fast alteration of difficulty

One person stated that the level was altered a little too fast; in the case of this experiment, this was done to investigate in an explorative manner the game mechanics or combinations of game mechanics that an individual player could be trained in.

5.4.4 Difficulty score version one and two

The results show that two people performed worse in version two of the game than in version one. The two participants that performed worse in version two of the game than in version one, were on higher difficulty scores. An explanation for this can be that on higher difficulty levels, the randomization part of the procedural content generation algorithm has a higher impact on the difficulty of the generated levels than it has on lower difficulty score. This is because the algorithm needs to place more obstacles in the world at random positions. The random placement of obstacles in the game level was not considered for the calculation of the difficulty score of a level. The problem with this is that different obstacles next to each other or combined can be considerably more difficult to overcome than other combinations of obstacles.

It should also be noted that the difference in the difficulty score was only a score of five: the users scored 80 in the first version and 75 in the second version of the

game. It should be noted that in future experiments it would be better to consider the position in the world of already generated obstacles within the procedural content generation algorithm and the calculation of the difficulty score. The problem stated before can also explain why some participants mentioned in the questionnaire that on higher difficulty scores, the levels can sometimes become drastically more difficult.

5.4.5 Higher difficulty scores can result in generation of unrealistic levels

One participant noted that sometimes on high difficulty scores, the level can look slightly random or unrealistic. This has two reasons, reason one being if a participant fails one type of training goal a lot, the adaptive procedural content generation algorithm will focus on generating levels with that training goal a lot. There are upper bound and lower bounds in place for the generation of obstacles. There is however a small chance it would for example only generate enemies if the player has only died to enemies. A part of a level only containing for example 20 enemies in it and nothing else can feel like an unrealistic level to the user. The second reason being the random placement of obstacle within a chunk (small part of a level) one part of the chunk might have considerably more obstacles while another part of the chunk might have no obstacles at all.

5.5 Limitations

One of the limitations of these experiments is that it is difficult to determine whether the chosen method would work for a different type of game genres, since the experiment was limited to one game genre, namely platformer games. It is also not possible to determine whether casual players would perform better than players that employ speed running since the gameplay time was limited to five minutes a session and the participants that used speed running would reach the difficult parts of the generated levels faster than for example a casual or collector type of player. Finally, since only a small group of the participants had never played Mario before, it is not possible to determine whether the chosen method is able to teach a person the basic mechanics of the game. However, the two participants that were beginners were both able to complete the game's introduction, which teaches the players the game mechan-

ics. One person was already able to do this when playing the first version and the other participant was able to complete it while playing the second version of the game.

One of the limitations of the system was that previously generated obstacles in the game level were not considered. For higher difficulties, not considering previously randomly placed objects in the level could lead to a drastic increase in the difficulty of the generated levels.

Chapter 6

Conclusion

The goal of this research was to create a method used for adaptive procedural content generation for the use case of training the user in getting basic game literacy and to determine which concept would be most conducive to achieving competence in playing a game. This was done by letting users play procedural content generated game levels in a Super Mario Bros [28] clone. To reach this main research goal, some sub questions had to be answered.

The first sub-question is: *how does a user acquire game literacy?* Which is the concept most conducive to achieving competence in playing a game? The user needs to learn the ability to play and analyze a level of the game Super Mario Bros. This means the user needs to learn the base mechanics of a game of Super Mario Bros and be enabled to execute them. This is known as basic game literacy. The user also needs to learn how to overcome different combined obstacles in the game with the game mechanics learned from the user's own perception. This is known as advanced literacy. Each mechanic can be associated with a training goal; if a player can overcome all the training goals the player can be said to have reached basic game literacy. If the player can overcome differently combined training goals to a certain level of proficiency, the player can be said to have reached advanced game literacy, since the player can now identify which action to use to overcome obstacles in the game.

The second sub-question is: *how can a user be kept engaged in the learning process and be prevented from wanting to quit?* For this we found the concept of flow which means keeping players optimally engaged in the content they are currently experiencing. This means that the game needs to offer some challenge to the player but should also have certain moments in the game where the player can relax a bit.

Adaptive methods and player modeling may be used to get people into the flow state. Player modeling is the study of computational models of users in games. This means the adjustment of the game's content based on the needs of individual players. To adjust the game's content, a method called adaptive procedural content generation can be used. Procedural content generation is a method of creating data algorithmically. In the game Super Mario Bros, this would mean generating challenging game levels for the user to play. The adaptive part is changing the level of difficulty and training goals of the game based on the performance of the individual user in previous levels. The player model can then determine if the difficulty of the generated levels need to be changed or the training goals need to be adjusted based on the tracked data.

The third sub-question is: *To what extent is an adaptive procedural content generation system able to train the "platformer" game literacy of a user?* To answer this research question we did a user-centered experiment in two different types of procedural content generation environments. The first method being a adaptive procedural content generation game environment as proposed in Section 3 that uses adaptive methods to target the users difficulty score and generates levels based on training goals. The second method being a base procedural content generation environment that increases the difficulty of the game with default intervals after completing a section of a level.

Based on the previous answered research questions we found that an adaptive procedural content generation algorithm needs three major components as listed below:

1. Collect player and level generation data;
2. Use the collected data in a player model to determine training goals and a difficulty score for the user;
3. Adapt the procedural generation algorithm to the needs of an individual user based on the outcome of the player model;

The game that was created to test this method was a clone of the popular game Super Mario Bros. For this version of Super Mario Bros, parts (chunks) of a game level were generated while the player was playing the game. After each completed chunk, the player model would check if it would have to change the difficulty score to target the player skill better based on the player's past performance. It would also

check the performance of the user over the last five played chunks, to see if the user had difficulty overcoming some mechanics. If the player model determined that some mechanics were harder for the user than others, it would lean more to generating levels with the mechanics the user had problems overcoming. These mechanics are referred to in this research as training goals. Once the difficulty level and training goals were set, the procedural content generation algorithm called Parameterized-Random-Notch was responsible for taking this information and generating believable and playable Super Mario Bros game levels. Parameterized-Random-Notch works by generating obstacles on an x by y grid, where each obstacle generated can be controlled by setting parameters based on the game mechanics of the game like number of enemies, number of platforms and number of chasms.

An experiment was conducted with 34 participants. In this experiment, participants were presented with either the adaptive environment as presented in Section 3 or the baseline adaptive environment of a Super Mario Bros game clone as presented in Section 3. The participants played two games created for each version of their assigned procedural content generation game environment that had been created for this purpose. Each version of the game was slightly altered from the other version in terms of physics and game visuals. The games are slightly different from each other, making it possible to verify if a user can use the game literacy learned from playing the first version of the game in another similar game, which is version two. The participants were also asked to fill in a questionnaire regarding their experience with the game.

Results indicate that the adaptive method is better suited in training the user in game literacy regarding platformer games than the base procedural content generation method. The adaptive method also showed that it can generate various game levels of the game Super Mario Bros of various difficulty for different kind of players and their skill level in playing video games. This supports the theory that adaptive procedural content generation algorithms can help in training a person's game literacy.

Finally, since only a small group of the participants had never played Mario before, it is not possible to determine whether the chosen method is able to teach a person the basic mechanics of the game. However, the two participants that were beginners were both able to complete the game's introduction, which teaches the players the game mechanics. One person was already able to do this when playing the first version and

the other participant was able to complete it while playing the second version of the game.

One of the downsides of the experiment is that it has been tested on one specific game genre, namely platformer games. For future work it would be interesting to test the method on one or more different types of game genres. This would be useful to determine whether the method can be used with several types of game genres. Another aspect that could be improved is to take into consideration previously placed obstacles in the world when calculating the difficulty score. For the experiment presented in this paper, this was not done and for higher difficulties, this could lead to a sometimes-drastic increase in the difficulty of the generated levels, because of the randomized placement of obstacles in the world. One way to solve this could be by limiting the number of obstacles that can be placed within an x by y space and increasing this limit when the difficulty score of the user increases.

Appendix A

Additional Information

A.1 Statements made by participants

- nice game.
- funny and challenging
- Overall: good impression.
- Got harder as i played better
- Some surprising spawns vs the original
- Nice, been familiar with the game style.
- Very challenging, wanted to play again to get a better highscore!
- Seeing the increase in difficulty gave a feeling of satisfaction and challenge. Interesting to play.
- At the end, the map felt more unnatural. Especially in version 2, it didn't look and feel like a regular mario level.
- In my opinion, it altered the level somewhat fast. In the first few stages that wouldn't be a problem. At the end however, it became pretty savage. Overall it was a pretty nice way to get used to the pc controls instead of an old school controller!
- got a good quick impression of the game, it's easy to play in the beginning and start with the basics. It interact how you play the game. When you get further

in the game and completed you're goals the game will be harder so that you can increase you're skills.

- like being challenged in a game so the change in difficulty got me more into the game and made me want to perform better. When a game is too easy you're not really paying attention and just cruising through. I think it will be important to find the right balance when it's getting too difficult that it also adapts to making it easier.

- Mario felt awkward to control because his movement speed scales while I am used to Mario having a constant movement speed. Apart from that the impact of the adaptive difficulty is very hard to gauge because I can't make a comparison with how it would've played if it was harder/easier and I wouldn't want to play a game like that multiple times to find out. Because of this the message "Good job! Difficulty increasing" felt useless to me because I don't know what it changes and games pretty much always get more difficult the further you get into a level. Maybe the system would be better suited as a subtle background thing that can alter difficulty without the player knowing?

- It's fun to see that the difficulty level gets increased. Makes it exciting to see what is coming next.

- Nice.

- I got a very pleasant feeling playing the game.

A.2 Base version overarching design

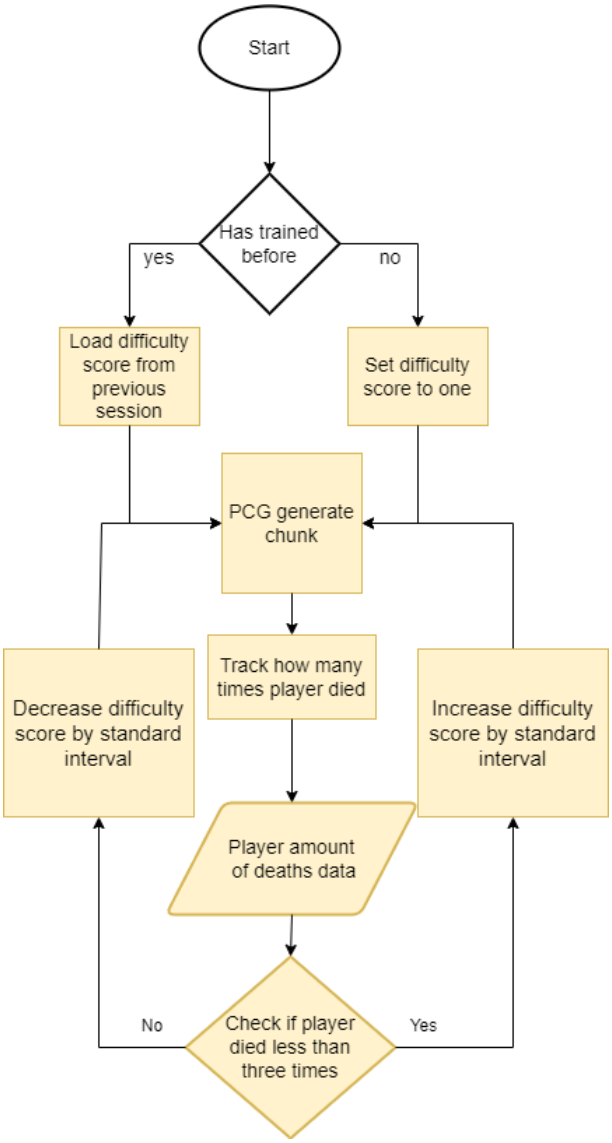


Figure A.1: Base version overarching design

A.3 Base version player model design

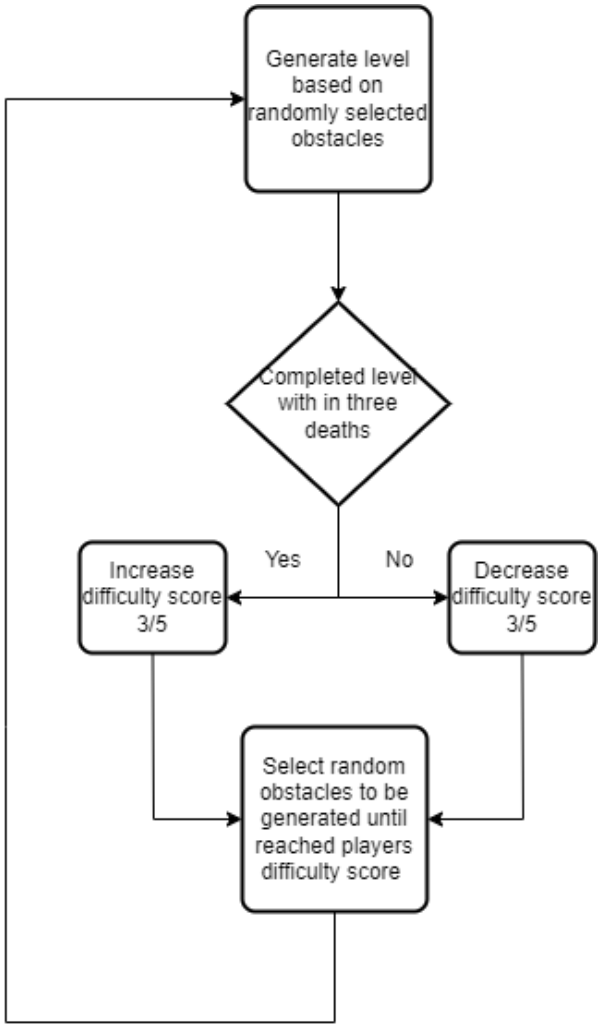


Figure A.2: Base version player model

A.4 Adaptive version overarching design

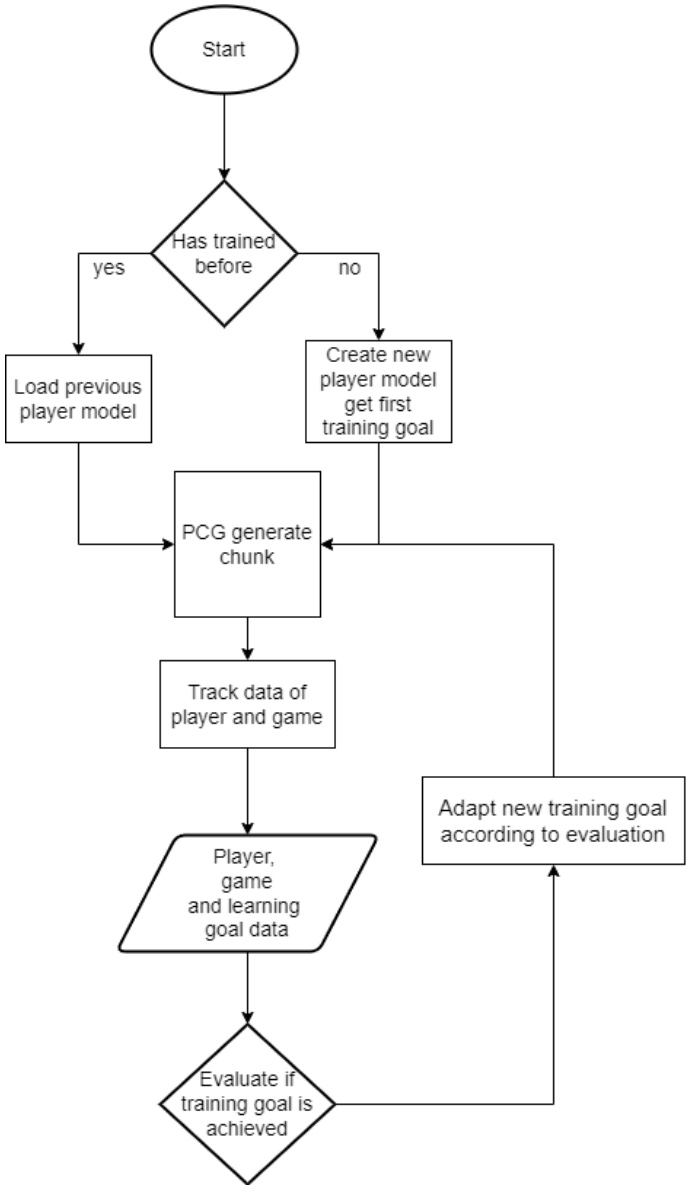


Figure A.3: Overarching design

A.5 Adaptive version player model design

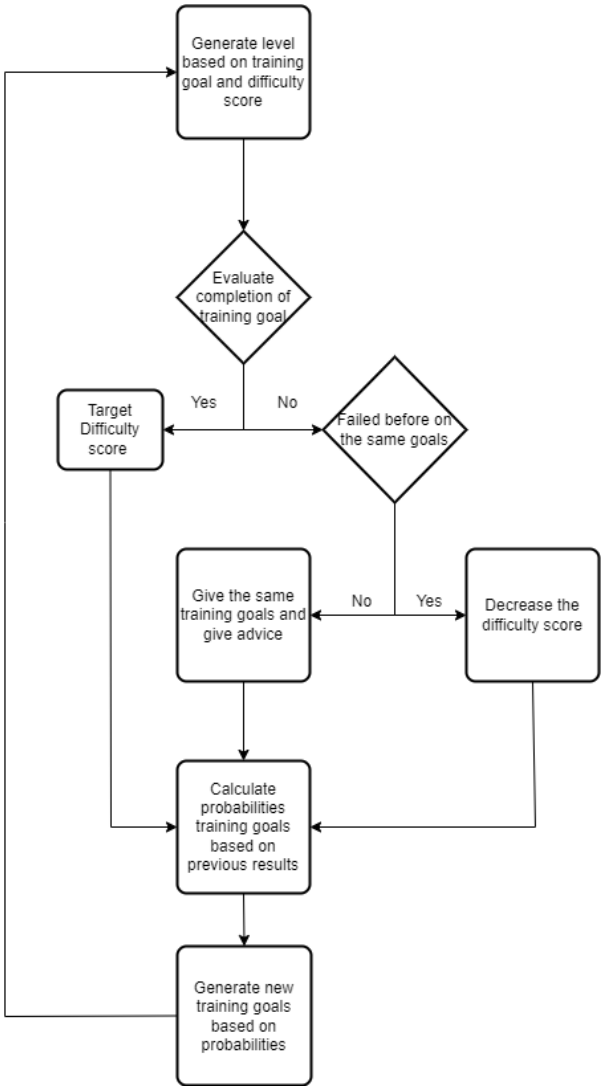


Figure A.4: Player model design

A.6 Adaptive individual player results

User	Version	Intro completed	Highest score	Speed runner	Collector
1	1	true	55	false	false
1	2	true	65	false	false
2	1	true	30	false	false
2	2	true	45	false	false
3	1	true	30	false	false
3	2	true	40	false	false
4	1	true	10	false	false
4	2	true	15	false	false
5	1	true	80	true	false
5	2	true	75	true	false
6	1	true	45	true	false
6	2	true	50	true	false
7	1	true	60	true	false
7	2	true	90	true	false
8	1	true	20	false	false
8	2	true	25	false	false
9	1	true	30	false	false
9	2	true	70	false	false
10	1	true	36	true	false
10	1	true	80	true	false
11	1	true	40	true	false
11	2	true	70	true	false
12	1	false	10	false	true
12	2	true	25	false	true
13	1	true	25	false	false
13	2	true	35	false	false
14	1	true	40	true	false
14	2	true	65	true	false
15	1	true	60	true	false
15	2	true	90	true	false
16	1	true	85	true	true
16	2	true	100	true	true

Continued on next page

Table A.0 – continued from previous page

User	Version	Intro completed	Highest score	Speed runner	Collector
17	1	true	80	true	true
17	2	true	70	true	true
18	1	true	30	false	false
18	2	true	60	false	false
19	1	true	60	true	false
19	2	true	80	true	false
20	1	true	15	false	true
20	2	true	50	false	true
21	1	true	35	false	true
21	2	true	65	false	true
22	1	true	30	false	true
22	2	true	55	false	true

A.7 Base individual player results

User	Version	Intro completed	Highest score	Speed runner	Collector
1	1	true	25	false	false
1	2	true	28	false	false
2	1	true	22	false	false
2	2	true	22	false	false
3	1	true	28	false	true
3	2	true	22	false	true
4	1	true	21	false	false
4	2	true	19	false	false
5	1	true	38	false	false
5	2	true	38	false	false
6	1	true	30	false	false
6	2	true	38	false	false
7	1	true	26	false	false
7	2	true	24	false	false
8	1	true	20	false	false
8	2	true	23	false	false
9	1	true	30	false	false
9	2	true	34	false	false
10	1	true	17	false	false
10	1	true	23	false	false
11	1	true	18	false	false
11	2	true	26	false	false
12	1	false	38	true	false
12	2	true	36	true	false

Bibliography

[1] Lisa Feldman Barrett. Valence focus and arousal focus: Individual differences in the structure of affective experience. In *Journal of Personality and Social Psychology*, pages 153–166, 1995.

[2] Richard Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. *Interaction design foundation*, 1996.

[3] Noor Shaker; Julian Togelius; Georgios N. Yannakakis; Ben Weber; Tomoyuki Shimizu; Tomonori Hashiyama; Nathan Sorenson; Philippe Pasquier; Peter Mawhorter; Glen Takahashi; Gillian Smith; Robin Baumgarten. The 2010 mario ai championship: Level generation track. In *IEEE Transactions on Computational Intelligence and AI in Games*, pages 332–347, 2011.

[4] D. Braben and I Bell. Elite (bbc micro). In *Acornsoft*, 1984.

[5] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] Jenova Chen. Flow in games(and everything else). *Communications of the ACM*, 2013.

[7] Dr Natalie Coyle. The psychology of rage quitting. `http://platinumparagon.info/psychology-of-rage-quitting/`, 2018. (accessed: 18.10.2021).

[8] Arsenault Dominic. Video game genre, evolution and innovation. *Eludamos: journal for computer game culture*, 3:149–176, 2009.

[9] Susan L. Epstein. Towards an ideal trainer. *Mach. Learn.*, June 1994.

[10] Susan L. Epstein. Learning to play expertly: A tutorial on hoyle. *""*, January 2001.

[11] Joshua Gad. The power of game literacy. *superjumpmagazine*, 2019.

[12] James Paul Gee. *What Video Games Have to Teach Us About Learning and Literacy.* St. Martin's Griffin, 2007.

[13] Sander Bakkes; Shimon Whiteson; Guangliang Li; George Viorel; Visniuc; Efstathios Charitos; Norbert Heijne and Arjen Swellengrebel. Challenge balancing for personalised game spaces. In *IEEE Games Media Entertainment*, 2014.

[14] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[15] Danial Hooshyar, Moslem Yousef, and Heuiseok Lim. A systematic review of data-driven approaches in player modeling of educational games. In *Artificial Intelligence Review volume 52 2017*, 2017.

[16] Ricardo Lopes; Ken Hilf; Luke Jayapalan and Rafael Bidarra. Mobile adaptive procedural content generation. In *Proceedings of the fourth workshop on Procedural Content Generation in Games*, 2013.

[17] Sandra Sampayo Vargas; Chris J.Cope; Zhen He; Graeme J.Byrne. The effectiveness of adaptive difficulty adjustments on students' motivation and learning in an educational computer game. In *Computers & Education Volume 69*, pages Pages 452–462, 2013.

[18] Jdaster64. A complete guide to super mario bros physics engine. `https://web.archive.org/web/20130807122227/http://i276.photobucket.com/albums/kk21/jdaster64/smb_playerphysics.png`. (accessed: 28.02.2022).

[19] Michael Cerny Green; Luvneesh Mugrai; Ahmed Khalifa and Julian Togelius. Mario level generation from mechanics using scene stitching. In *Foundations of Digital Games, September 2020*, pages 266–274, 2020.

[20] Darryl Charles; Michael McNeill; Moira McAlister; Michaela Black; Adrian Moore; Karl Stringer; Julian Kücklich; and Aphra Kerr. Player-centred game design: Player modelling and adaptive digital games. In *Digital Games Research Group*, 2005.

[21] Manuel Lafond. The complexity of speedrunning video games. In *9th International Conference on Fun with Algorithms*, pages 27:1–27:19, 2018.

[22] Georgios N. Yannakakis; Pieter Spronck; Daniele Loiacono and Elisabeth Andre. Player modeling. *Artificial and Computational Intelligence in Games*, 2013.

[23] Marlos C Machado, Fantini Eduardo P.C., and Luiz Chaimowicz. Player modeling: Towards a common taxonomy. In *2011 16th International Conference on Computer Games (CGAMES)*, pages 50–57, 2011.

[24] Thomas Malone. What makes things fun to learn? a study of intrinsically motivating computer games. In *Pipeline*, 1981.

[25] Csikszentmihalyi Mihaly. *Flow: The psychology of optimal experience*. Harper & Row New York, 1990.

[26] Olana Missura and Thomas Gartner. Player modeling for intelligent difficulty adjustment. In *International Conference on Discovery Science*, 2015.

[27] Olana Missura, Thomas, and Gartner. Player modeling for intelligent difficulty adjustment. In *International Conference on Discovery Science*, pages 197–211, 2009.

[28] Shigeru Miyamoto. Super mario bros. In *Nintendo*, 1985.

[29] Berndt Müller, Joachim Reinhardt, and Michael T Strickland. *Neural networks: an introduction*. Springer Science & Business Media, 1995.

[30] Nintendo. Super mario maker. In *Nintendo*, 2015.

[31] Sérgio Oliveira and Luís Magalhães. Adaptive content generation for games. In *Encontro Português de Computação Gráfica e Interação*, 2017.

[32] Markus Persson. Infinite mario bros. `https://creatorcsie.github.io/NotchGame/Applet&JNLP/Mario/`. (accessed: 22.11.2021).

[33] Erik Andersen Eleanor O'Rourke; Yun-En Liu; Richard Snider; Jeff Lowdermilk; David Truong; Seth Cooper; Zoran Popovic. The impact of tutorials on games of varying complexity. In *Department of Computer Science & Engineering, University of Washington*, 2012.

[34] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. Coevolutionary principles., 2012.

[35] Ben Weber; Robert Quigley. The mario game that gets harder the better you are. `https://www.themarysue.com/infinite-adaptive-mario/`. (accessed: 01.03.2022).

[36] Noor Shaker and Georgios Yannakakis. Feature analysis for modeling game content quality. In *Computational Intelligence and Games (CIG) 2011 IEEE Conference*, pages 266–274, 2011.

[37] Steve Dahlskog; Britton Horn; Noor Shaker and Gillian Smith. A comparative evaluation of procedural level generators in the mario ai framework. In *Foundations of Digital Games*, 04 2014.

[38] Adam M. Smith; Chris Lewis; Kenneth Hullett; Gillian Smith and Anne Sullivan. An inclusive view of player modeling. In *Proceedings of the 6th international conference on foundation of digital games*, pages 301–303, 2011.

[39] Gillian Smith. An analog history of procedural content generation. In *FDG*, 2015.

[40] Pieter Spronck; Ida Sprinkhuizen-Kuyper and Eric Postma. Difficulty scaling of game ai. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation*, 2004.

[41] Paris Mavromoustakos Blom; Sander Bakkes; Pieter Spronck. Modeling and adjusting in-game difficulty based on facial expression analysis. *Entertainment Computing*, 31, 2019.

[42] Sander Bakkes; Chek Tien Tan and Yusuf Pisan. Personalised gaming: a motivation and overview of literature. In *Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System*, pages 1–10, 2012.

[43] Christopher Pedersen; Julian Togelius and Georgios Yannakakis. Modeling player experience for content creation. In *IEEE Transactions on Computational Intelligence and AI in Games 2*, pages 54–67, 2010.

[44] Georgios Yannakakis; Julian Togelius and Chris Pedersen. Modeling player experience in super mario bros. In *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, 2009.

[45] S. Karakovskiy; Julian Togelius; and Robin Baumgarten. The 2009 mario ai competition. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, 2009.

[46] Zamel V and Spack R. *Negotiating Academic Literacies Teaching and Learning Across Languages and Cultures*. Routledge, 1998.

[47] Ben Weber. The mario game that gets harder the better you are. `http://alumni.soe.ucsc.edu/~bweber/dokuwiki/doku.php?id=infinite_adaptive_mario`. (accessed: 01.03.2022).

[48] Geogios N. Yannakakis. Game ai revisited. In *Proceedings of the 9th conference on Computing Frontiers*, pages 285, 292, 2012.

[49] Georgios Yannakakis and Julian Togelius. Experience-driven procedural content generation. In *IEEE Transactions on Affective Computing*, 2011.