Received Feb, 14th, 2018. Accepted July, 7th, 2018.

International Journal of Asia Digital Art&Design

Perception of Difficulty in 2D Platformers Using Graph Grammars

Henry Fernández
Tokyo University of Technology
henrydfb@gmail.com

Koji Mikami
Tokyo University of Technology
mikami@stf.teu.ac.jp

Kunio Kondo
Tokyo University of Technology
kondo@stf.teu.ac.jp

Abstract
This paper summarizes the findings of a study on difficulty perception conducted with players of 2D side-scrolling platform games which level design is automatically created. The primary goal of our research, is to offer players a suitable experience that matches their abilities and the difficulty of the games they play. The study presented here was conducted to test the calculations and formulas designed for a method that adapts the difficulty of a level according to the player skills through level design. We created a new method that combines Dynamic Difficulty Adjustment and the principle of Graph Grammars to create levels with a specific degree of difficulty, the novelty of our method lies on the use of Graph Grammars as a tool for creating multipath levels in 2D platformers; an implementation of Graph Grammars applied to 2D platformers; the creation of formulas that numerically estimate difficulty and insights about the perception of difficulty. Results show that the difficulty estimation of our method and the difficulty perceived by participants of our experiment has a correlation coefficient of 0.75, with a linear relationship and a strong correlation between both variables, demonstrating that the approach is heading to the right direction. In addition, difficulty and performance have a correlation of -0.69, which shows an inversely proportional relationship, more difficult levels have lower performance than easier levels; for this particular research this was an expected result, confirming the method is doing the calculations properly.

Keywords: Game Design, Level Design, Game Development, Platformer, Procedural Content Generation, Dynamic Difficulty Adjustment, Player Experience, Graph Grammars, Difficulty, Performance, Experience Design

1 Introduction

Our motivation for starting this study is the existence of imbalance between the skills of players and the difficulty of the games they play. We can see how this issue affects the overall experience for players when the difficulty of a game is not suitable for the player's abilities [1]. High skill players might feel bored and low skilled players might feel frustrated when playing games with unsuitable difficulty. A misleading experience can end up in quitting the game as a result.

In a previous research, we combined electroencephalography (EEG) and player performance with Dynamic Difficulty Adjustment (DDA) and Procedural Content Generation (PCG): specifically Rhythm-Group Theory [2], we obtained results from experiments that indicated that the method we designed enables developers to adapt the level design's difficulty depending on the player's skills.

This particular study consists of a preliminary exploration conducted to test the base calculations of a more general method designed for adapting the player experience. This new general method we are still creating, follows the approach of our previous method but without the EEG component and focuses on a new, more expressive PCG technique: Graph Grammars (which enables the creation of more expressive and varied levels).

In this paper, we introduce our vision on how to apply Graph Grammars to create multi-path levels in 2D platformers with specific degree of difficulty and show the results of experiments designed to test the player's perception about difficulty and how is that related to the effectiveness of our method and the calculation of difficulty in general.

Results show that there is a strong linear relationship between the difficulty perceived by participants of our experiments and the difficulty calculated by the algorithm we created, with a correlation coefficient of 0.75.

In addition, we calculated the linear regression and correlation coefficient of the computed difficulty and the player's performance, using formulas designed from our method, showing a correlation of -0.69 which is a moderate correlation. Having a correlation of -0.69 demonstrates that both variables are inversely proportionals, they decrease or increase in opposite directions. This is an expected result and is an indication that the calculations are heading to the right direction.

2 Related Work

One of the reasons for imbalance between skills and challenge to exist is the lack of harmony in game design, which could be solved by changing the rules of the game [3]. The problem with

this solution is that to match every kind of player's skills is very difficult.

A common way to tackle this issue is to include Dynamic Difficulty Adjustment (DDA) techniques in their games [1][4][5]. By doing this, the game adapts itself to the player, creating a suitable experience. Previous researchers combined DDA and PCG techniques in 2D platformers [6] which demonstrated to be a good approach.

In addition, there are researchers like Noor Shaker and Georgios Yannakakis who work towards personalizing the player experience automatically using PCG and Artificial Intelligence (AI) techniques [7].

As an improvement of our previous research [2], we decided to choose a more expressive PCG technique to create levels that could be constructed with more than one path to increase diversity and reduce the monotony and entertain the player with new content.

The new method we decided to use, Graph Grammars, was originally designed for automatic dungeon creation by David Adams [8] for his Bachelors thesis research. We found in previous research that Graph Grammars shows how this technique has been tested and compared against other methods for automatic content creation [9], having interesting results creating dungeon maps driven by gameplay. In addition to this, the Graph Grammars method was used in a recent research about a learning game to teach parallel programming [10], which was partially successful showing positive results in automatic puzzle creation.

Some researchers have explored the idea of appliying the same method to 2D platformers: proposing a graph-based representation of Super Mario Bros levels using graph grammars [11],[12],[13]. Joris Dormans and Sander Bakkes have focused on using generative graph grammars to procedurally generate missions for action-adventure games, which due to their nature of nonlinear structures creation make them a suitable solution for games that involve exploration [14],[15]; in this research they demonstrated that graph grammars could be a powerful tool to use the advantages of procedural generation and human design in the creation of mission and level design.

Using a dungeon crawler game case of study, researchers concluded that graph grammars can be expressive enough to construct design levels in this type of games, leading to the creation of a variety of possibilities with rich and interesting results for players to explore and enjoy [16].

Besides Graph Grammars, we focused this particular study on perceived difficulty, defined as relative difficulty minus the players experience at meeting specific challenges [17].

A previous study that involves Bayesian optimization shows that there is a significant relevance between perceived difficulty and engagement [18]. Researchers found that players attributed changes in their performance to their own capacity, which was in reality affected by the covert manipulation of difficulty done by researchers.

We consider that our contribution adds up to previous research on the importance of examining difficulty from the player's point of view when analysing games [19] and the evaluation of different factors (including difficulty) on the player's performance [20].

3 Graph Grammars

The concept of graph grammars involves the idea of having a grammar that handles graphs instead of strings. One of the reasons for replacing strings by graphs when automatically creating levels is that due to their nature, graphs are a good fit to partition and represent the 2D space. Among the advantages of using graph grammars to build levels instead of other kinds of grammars is that they provide much more flexibility when creating variation and randomness. In addition, they allow creating complex levels in a more natural way [8].

The following, shows a formal definition of a directed graph: G := (V,E,lV, lE, s, t), where:

- V(G) := V is a finite set of vertices
- E(G) := E is a finite set of edges
- lV(G) : V LV is a labelling function for vertices
- lE(G) : V LE is a labelling function for edges
- s(G) : E V assigns each edge to its source
- t(G) : E V assigns each edge to its target

A graph grammar is formally defined as a tuple (A,P) where A is a non-empty initial graph and P a set of graph grammar productions. Most approaches of graph production definition concur that each production consists of two parts: left-hand side and right-hand side. Being the left side of the production the one that defines how to replace and transform graphs in the grammar [8]. The set of productions defined in the method we propose are shown in section 3.1.

The method we designed to implement Graph Grammars in 2D platformers consists of three different systems that we called: topological map system; area designation system and a graphical map system. In a very brief way, the topological map defines how many nodes and how are the nodes connected in the graph; then, the area designation system assigns a type to each node in the graph; finally the graphical map system sets a representative game object for each node in the graph. Figure 1 shows the general approach of our method.

3.1 Topological Map

This map is the logical representation of the level. The following rules are the set of productions designed to define the grammar used to create the topological map.

1. Starting Rule: This production ensures that there is always a start and there is always an end. It includes a start node S connected directly to the ending node E.
2. Adding Rule: The adding rule enables the algorithm to include more nodes between nodes. We have two different

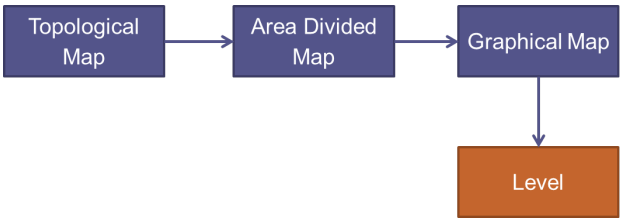

Figure 1 Graph Grammars Creation Approach:. The topological map creates a graph. The area divided map system assigns a type to each node of the graph. Finally the output area graph is used by the graphical map system to create a level.

types of adding rules:

(a) Linear adding rule: the node is created in one of the extremes of the path, increasing the length of this path.

(b) Non-linear adding rule: The node is created in a different path to the one that is being added. This usually introduces a new level in the system.

3. Linking Rule: The linking rule takes a couple of nodes and links them. This link could be directed or not directed. Two main cases can be considered when linking nodes:
   (a) Case A: Two nodes of the same path are connected
   (b) Case B: Two nodes of different paths are connected
4. Changing Rule: The changing rule enables the algorithm to decide a direction change after connecting all nodes.
5. Deleting Rule: This production says that we can eliminate nodes from a graph when needed.
6. Ending Rule: This production ensures that the result level can be cleared. In addition, it will ensure that there is always a complete path from the start of the level to the goal.

In this part of the process there are three main steps to ensure that the map is well created and will output an expected result, these are: main path, secondary path and arcs direction.

1. Main Path: This is the path that connects the start of the level to the end. This ensures that the level can always be finished by the player. This path is mandatory in every level.
2. Secondary Path: All paths that are connected to the main path and are not the main path, are called secondary paths.
3. Direction: The last segment of map generation enables the algorithm to change the direction of each arc to build a unique level. This adds variety and diversity and reduces the probabilities of having two similar levels.

All the previously explained steps guarantee the construction of a final map and sets the basis to create an output that can be used as a playable level in the end of the process. An example of a topological map can be seen in figure 2.

3.2 Area Designation Map

The result graph created by the topological system passes through a system that takes every node in the graph and assigns an area type, areas can be ”safe areas” or ”dangerous areas”.

A safe area is defined as a platform (of any length) where players can stand safely, it means, not harming game objects will be found in this type of area. Examples of this type of areas are: empty platforms (no enemies) and platforms that contain elements such as obstacles to stand on them, etc.

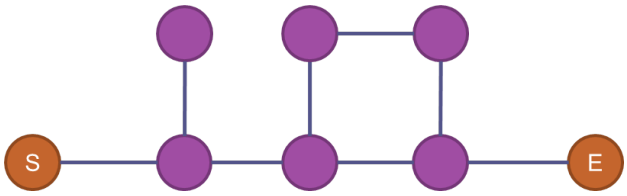

Figure 2 Example of a topological map. As we can see, there is a main path from S (start) to E (end) and, in addition, a second level (secondary path) with interconnected nodes.

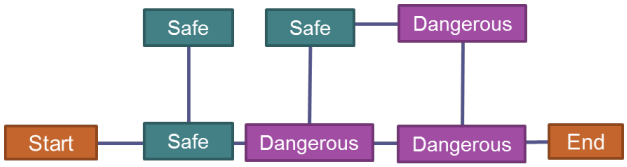

Figure 3 Example of an area divided map. Using the topological map from 2, this is an example of a possible area divided map, after assigning safe and dangerous areas.

A dangerous area is defined as a platform (of any length) where players can die, usually represented by areas that contain enemies that can harm the player in any way, examples of this type of areas are: movable platforms, platforms with enemies, platforms that fall, etc.

The starting and ending node are both set to safe areas, however it is possible to modify these rules and adapt them depending on the designer’s needs. An example of an area divided map that matches the topological map in figure 2 can be seen in figure 3.

The number of dangerous areas is decided using the expected difficulty for the level and multiply that by the total number of areas in the level. The calculation can be seen in equation 1.

$$a = l_d n \tag{1}$$

Where:

- a: Number of dangerous areas
- ld: Expected difficulty for the level
- n: Number of nodes in the graph

3.3 Graphical Map

The graphical map is the physical creation of the elements in the level. The main function of this system is to decide where to put each game object on the screen and how. This process translates a topological map to a graphical positions on the screen. Designers can decide and adapt this system according to the type of game

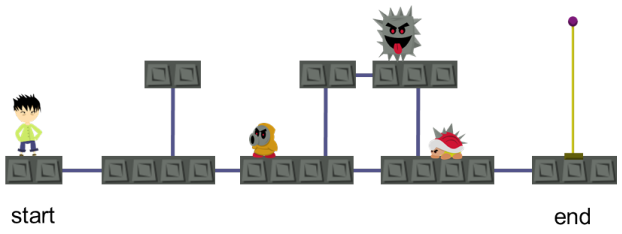

Figure 4 Example of graphical map. Using the area divided map of 3, this is an example of a possible graphical map.

elements that exist in their games. An example of a graphical map can be seen in figure 4.

4 Difficulty

The approach we took to calculate difficulty in a level is: assign difficulty values to each area in the graph and use those difficulties to calculate a general difficulty for the level. The general calculation of difficulty for a level can be seen in equation 2.

$$l_d = \frac{n}{m} V_1 + \sum_{i=1}^{n} d_i V_2 \tag{2}$$

Where:

- ld: Difficulty calculated for one level
- n: Number of dangerous areas in the level
- m: Number of total areas in the level
- di: Difficulty of each area (see equation 5)
- Vi: Weights that represent how much influence the component has

As preliminary parameters for this research we decided to start with V1 to 0.5 and V2 to 0.5 as well, due to the influence that both components have in the overall difficulty.

The calculation for each area is carried out using the game objects that are on it, difficulty values are assigned (by the designer, depending on the game) to each game object, for calculation difficulty only dangerous objects count.The difficulty calculation for each area is shown in equation 5.

$$a_d = \frac{n}{m} W_1 + \sum_{i=1}^{n} d_i W_2 \tag{3}$$

Where:

- ad: Difficulty calculated for one area
- n: Number of enemies in the area
- m: Maximum number of enemies in that area
- di: Difficulty of each enemy (see table 1)
- Wi: Weights that represent how much influence the component has

For this research, we set W1 to 0.9 and W2 to 0.1. As preliminary values we chose these two because the number of elements in a platform affects directly the performance, the more enemies, the higher the chance to get hit by them which means the performance would be lower.

To calculate the difficulty for each enemy, we interpret each skill shown in table 1 as one point of difficulty. Calculated difficulties are explained as follows:

1. Moves X Axis: Enemies that can walk or run. Moving makes an enemy more difficult than not moving.
2. Moves Y Axis: Enemies that can fall, jump, etc. Moving makes an enemy more difficult than not moving.
3. Shoots: An enemy that can shoot is more difficult than one that cannot shoot.
4. Beatable: Enemies that can be beaten by the player (jumping on its head), an enemy that cannot be beaten is more difficult than an enemy that can.
5. Visible: Enemies can hide, this makes it more difficult than an enemy that cannot hide.
6. Timing attack: The player has to be on alert when the enemy attacks, this adds a point of difficulty.
7. Aim: Aims at the player before attacking, this makes the enemy smarter, one more point of difficulty.
8. Player can stand: If the player can stand on this enemy sometimes and not be harmed by it, the enemy becomes easier. We add 1 point to all those enemies that the player cannot stand on them.

Depending on the kind of game that is being designed by the developers, the values in the difficulties table could vary to adapt to a more accurate result. For simplicity we decided to give one point of difficulty to each skill.

5 Player Performance

The player's performance calculation involves how many times is the player hit by an enemy, level clear time and number of collected coins. Equation 4 shows the calculation of the performance for this method

$$p = \frac{1}{(d+1)} X_1 + \frac{eT}{gT} X_2 + \frac{c}{m} X_3 \tag{4}$$

Table 1 Enemies features: Binary representation for the skills: 1=true,0=false

	Enemies Difficulties				
	Enemy 1	Enemy 2	Enemy 3	Enemy 4	Enemy 5
Moves X Axis	0	0	0	1	1
Moves Y Axis	0	1	1	0	0
Shoots	1	0	0	0	0
Beatable	1	0	0	0	1
Visible	1	1	0	1	1
Timing Attack	1	1	1	0	0
Aim	1	0	0	0	0
Player Can Stand	0	0	1	0	0
Total	4	4	4	3	2

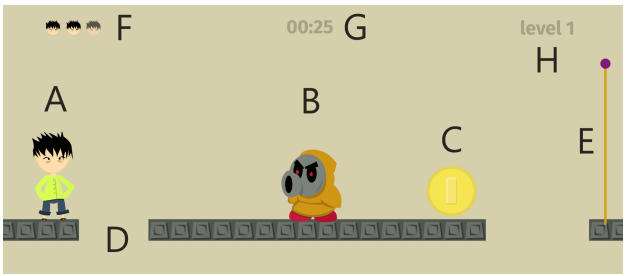

Figure 5 Game Elements. A: Player; B: Enemies, beatable and unbeatable; C: Coins, motivation to explore; D: Gap; E: Goal; F: Player's health and coins (3 per level); G: Game Time; H: Current Level.

Where:

- p: Calculated performance
- d: Number of deaths
- eT: Level estimated time (15 seconds)
- gT: Level cleared time (cannot be less than eT)
- c: Number of collected coins
- m: Maximum number of coins per level (3 coins)
- Xi: Weights that represent how much influence the component has

For this study we started testing with a set of parameters that we consider is the most suitable combination of values: X1: 0.5, X2: 0.15 and X3: 0.35, decided by the importance of each parameter in the equation.

6 Implementation and Experiments

The game is a side-scrolling 2D platformer in which the player has to reach a goal placed on the right-most part of the level. The player is also required to overcome very simple challenges: gaps between platforms, beatable enemies and unbeatable enemies, both types of enemies static. We also included 3 collectable coins to increase motivation for the player to explore different paths. Figure 5 shows the elements of the implemented game.

Players were required to play and clear 12 different levels generated automatically by our system. The first three levels were part of a tutorial to give players a sense of difficulty and rating. After finishing one level, players were taken to a results screen that showed a set of 10 different options to rate the level: [1-10]; being 1 the easiest and 10 the hardest values. Numbers were presented as integers to the player but they were normalized (0,1) for our internal calculations. The whole flow of the experiment can be seen in Figure 6.

We collected data from 16 different players. All participants were between 21 40 years old; almost all of them with previous experience playing *Super Mario Bros* (which we asked to have background information about their skills), one of the players (6.25%) has never played Super Mario Bros before, only the 7 players (43.75%) have cleared the game at least once.

For our implementation, we consider the following parameters:

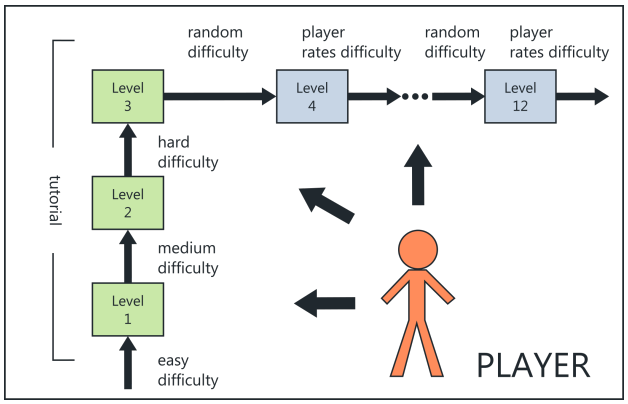

Figure 6 Experiments. Players played three tutorial levels and nine levels for which difficulty was randomly calculated. After playing a level players rated its difficulty.

minimum number of nodes in the main path, maximum number of nodes, maximum number of secondary paths, minimum number of branch nodes and maximum number of branch nodes.

1. Minimum number of nodes in the main path
2. Maximum number of nodes in the graph
3. Minimum number of secondary paths
4. Maximum number of secondary paths
5. Minimum number of branch nodes
6. Maximum number of branch nodes

In our experiments, the parameters of this system we set to the minimum possible values for simplicity. The whole creation of a topological graph is done randomly and automatically. Since this step of the process does not involve any graphical elements, we do not consider difficulty when creating this first part of the graph.

Graph Grammars System

In our implementation, the Graph Grammars method involves three different sub-systems: topological map system, area divided map system and graphical map system, each one with a clearly different purpose.

Topological Map: Graph Creation

A set of parameters are important to take into consideration to construct a graph, these are: minimum number of nodes in the main path, maximum number of nodes, maximum number of secondary paths, minimum number of branch nodes and maximum number of branch nodes. For our research we used the following values:

1. Minimum number of nodes in the main path: number of starting nodes between the node S (start node) and node E (end node). We set this value to 3.
2. Maximum number of nodes: Including all paths, main and secondary, the maximum number of nodes was set to 15.
3. Minimum number of secondary paths: number of paths (besides the main path) to start. The algorithm can create

graphs with no secondary paths.

4. Maximum number of secondary paths: For simplicity, this parameter was set to 1, it means, in total, a level created by this algorithm could have 2 paths.
5. Minimum number of branch nodes: its related to the minimum number of nodes that can be created as a branch of an existing node. We set this parameter to 1.
6. Maximum number of branch nodes: its related to the maximum number of nodes that can be created as a branch of an existing node. We set this parameter to 3.

We set these parameters to the minimum possible values for simplicity and to demonstrate the capabilities of our method.

Figure 7 and 8 show are examples of levels created for a low performance player and a high performance player respectively. Both results were generated using the Graph Grammars implementation explained in this section.

Area Divided Map

In addition to assigning a type to each node (area) of the graph, this system calculates the difficulty of each dangerous are depending on the expected difficulty of the whole level. The difficulty calculation for each area is shown in equation 5.

$$a_d = \frac{n}{m} W_1 + \sum_{i=1}^{n} d_i W_2 \tag{5}$$

Where:

- ad: Difficulty calculated for one area
- n: Number of enemies in the area
- m: Maximum number of enemies in that area
- di: Difficulty of each enemy (see table 1)

For this research, we set W1 to 0.9 and W2 to 0.1. As preliminary values we chose these two because the number of elements in a platform affects directly the performance, the more enemies, the higher the chance to get hit by them which means the performance would be lower.

To calculate the difficulty for each enemy, we interpret each skill shown in table 1 as one point of difficulty. Calculated difficulties are explained as follows:

1. Moves X Axis: It means the enemy can move in the X axis, enemies that can walk or run. Moving makes an enemy more difficult than not moving, if the enemy moves it gets one point o difficulty.
2. Moves Y Axis: It means the enemy can move in the Y axis, enemies that can fall, jump, etc. Moving makes an enemy more difficult than not moving, if the enemy moves it gets one point o difficulty.
3. Shoots: An enemy that can shoot is more difficult than one that cannot shoot, shooting gives one point of difficulty to an enemy.
4. Beatable: The enemy can be beaten by the player (jumping on its head), an enemy that cannot be beaten is more difficult than an enemy that can, an unbeatable enemy gets one point of difficulty.
5. Visible: Enemies can hide (like enemy 3), this makes it more difficult than an enemy that cannot hide. Hiding enemies get one point of difficulty.
6. Timing attack: since the player has to be on alert when the enemy attacks, this adds a point of difficulty to this kind of enemy.
7. Aim: Aims at the player before attacking, this makes the enemy smarter, one more point of difficulty.
8. Player can stand: If the player can stand on this enemy sometimes and not be harmed by it, the enemy becomes easier. We add 1 point to all those enemies that the player cannot stand on them.

Depending on the kind of game that is being designed by the developers, the values in the difficulties table could vary to adapt to a more accurate result. For simplicity we decided to give one point of difficulty to each skill.

Number of dangerous areas

To decide the number of dangerous areas, we use the expected difficulty for the level and multiply that by the total number of areas in the level. The calculation can be seen in equation 1.

Graphical Map: Level Creation

An explanation on how to create game objects for each area is as follows.

1. Coins: Coins can be placed in any type of area, safe or dangerous and there is a fixed amount of coins per level (3 coins) so we randomly choose 3 areas (could be the same area) and decide where the coins will be.
2. Safe Areas: When geometrically construction safe areas, we can decide to add a coin, a harmless obstacle or nothing. Coins have priority because it is mandatory that each area with a coin shows that coin somewhere. After deciding where the coin should be placed, the rest of the available spaces in the are set to having an obstacle or being empty. We do this randomly.
3. Dangerous Areas: To decide what enemies will be placed in a dangerous area, we use the same approach that we used to calculate how difficult an area should be but instead of using the whole levels difficulty, we use the areas difficulty and distribute the enemies on the area in a random way.

7 Results and Analysis

In order to analyse the results, we calculated the linear regression of the computed difficulty, perceived difficulty and performance in two different plots. Figure 9 shows the results of the calculated difficulty and perceived difficulty, figure 10 shows the results of the calculated difficulty and performance calculated by the algorithm.

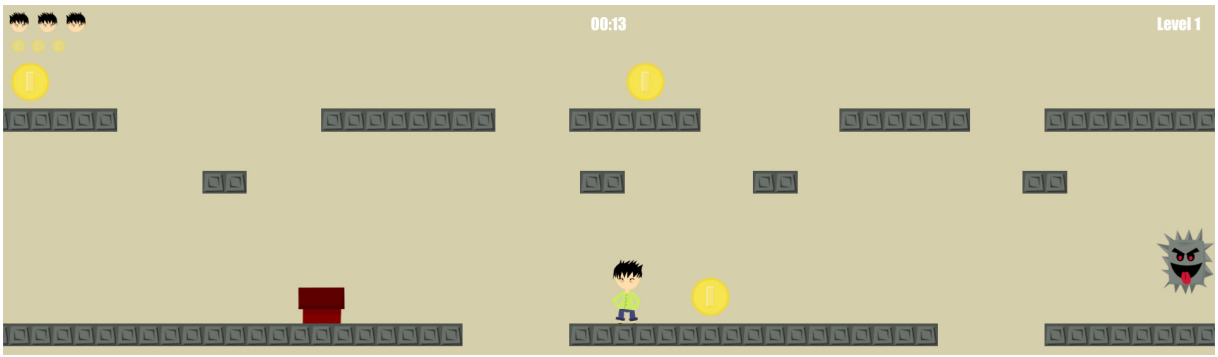

Figure 7 Low performance player. A level created for a low performance player, few challenges, easier to clear

Figure 8 High performance player. A level created for a high performance player, more challenges, more difficult to clear.

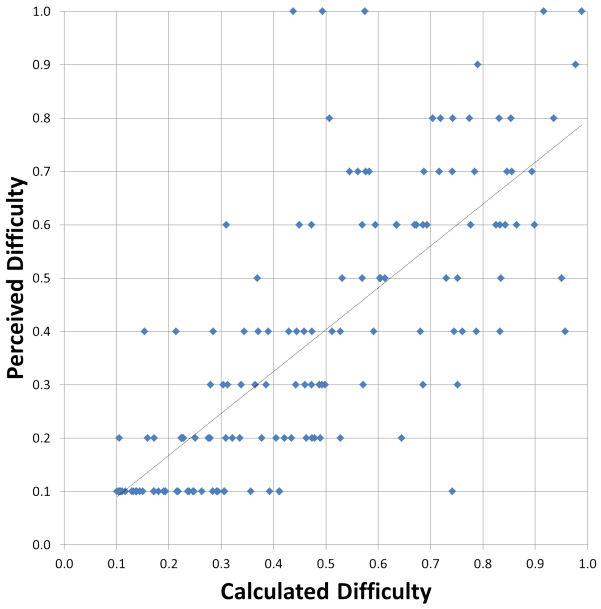

Figure 9 Perceived Difficulty vs Performance. The horizontal axis shows the calculated difficulty, vertical axis the perceived difficulty. We can clearly see a strong linear relationship between the variables.

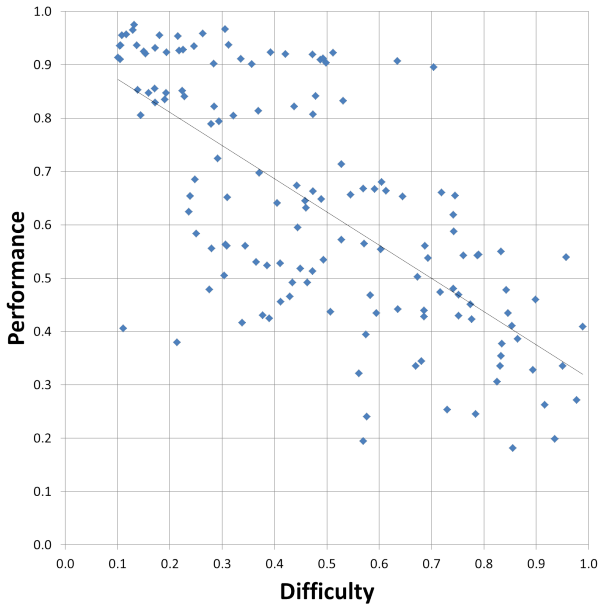

Figure 10 Performance vs Calculated Difficulty. The horizontal axis shows the calculated difficulty, vertical axis the performance. We can see a linear inversely proportional relationship between the variables.

7.1 Difficulty Calculation

The correlation coefficient for calculated vs perceived difficulty variables was 0.75, which is considered a strong correlation (more than 0.7) using statistical basics. This means that both results are strongly related, which demonstrates that the approach of the calculations of difficulty in our algorithm are heading in the right direction.

One of the reasons that these results might have been affected is the way the experiment was designed, each level's difficulty

was decided randomly during the experiment, with no particular order in way the difficulty was presented to players. We consider that a way to improve this experiment is to control exactly which difficulty and when to present it to players.

Despite the fact that the calculations and the experiment should be modified and improved to avoid abrupt changes of difficulty while playing, having a 0.75 of correlation between the perceived difficulty and the calculated one is a positive result. This means we can keep enhancing this method to achieve better results.

7.2 Performance and Difficulty

In figure 10 we can see how the data has a strong linear inverse tendency, with a correlation coefficient of -0.69, despite the fact that is not as high as 0.7 to qualify as strong correlation, it's a high value that represents a close relationship between the difficulty and performance calculated by the algorithm. This result shows that difficulty and performance are inversely proportional in this case, the higher the difficulty, the lower the performance, which in a way would be an expected result, however it depends on the kind of player that is playing.

Since the conducted experiments did not involve any adaptation functionality, we consider this a positive result, nonetheless for the ultimate goal of our research, adapting the player experience according to the player's skills, this correlation should be the opposite.

8 Conclusions

We successfully implemented a method that involves Graph Grammars to create multipath levels in 2D platform games, which increases expressiveness and variety of automatically created levels in procedural construction of levels.

This is only a first approach of a more general and bigger purpose method that we are currently designing to adapt the player experience through difficulty balance and performance.

Our experiment showed a 0.75 correlation coefficient between the difficulty calculated by our algorithm and the difficulty perceived by players. This strong correlation demonstrates that the approach is heading to the right direction, however we need to keep improving the current results to design a more robust method that accurately defines difficulty for players.

Results also showed that performance and difficulty have a correlation of -0.69, which is near to be considered a strong correlation and reinforces an expected consequence: the level's difficulty and player performance are inversely proportional. This result in the general approach of experience adaptation should be the opposite at some point where the performance of players increases along with the difficulty of the levels they play.

We need to keep improving the method by changing the parameters of each formula and test accordingly with players to find the best values and get positive results. In addition, new experiments should be designed and conducted to evaluate the accuracy of this method. Finally we will include these calculations in a more general method designed to adapt player experience and compare results.

References

[1] Jared E. Cechanowicz, Carl Gutwin, and Scott Bateman. Improving player balancing in racing games. In *Proceedings of CHI PLAY '14*, pages 47–56. ACM New York, NY, USA 2014, October 2014.

[2] Henry Fernández, Koji Mikami, and Kunio Kondo. Adaptable game experience through procedural content generation and brain computer interface. In *Proceedings of SIGGRAPH '16 Posters*. ACM New York, NY, USA 2016, July 2016.

[3] Eva Kraaijenbrink, Frank Gils, Quan Cheng, Robert Herk, and Elise Hoven. Balancing skills to optimize fun in interactive board games. In *Proceedings of INTERACT '09*, pages 301–313. Springer-Verlag Berlin, Heidelberg 2009, August 2009.

[4] Alexander Baldwin, Daniel Johnson, and Peta A. Wyeth. The effect of multiplayer dynamic difficulty adjustment on the player experience of video games. In *Proceedings of CHI EA '14*, pages 1489–1494. ACM New York, NY, USA 2014, April 2014.

[5] Robin Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of ACE '05*, pages 429–433. ACM New York, NY, USA 2005, June 2005.

[6] Martin Jennings-Teats, Gillian Smith, and Noah Wardrip-Fruin. Polymorph: Dynamic difficulty adjustment through level generation. In *Proceedings of PCGames '10*. ACM New York, NY, USA 2010, June 2010.

[7] Noor Shaker, Georgios Yannakakis, and Julian Togelius. Towards automatic personalized content generation for platform games. In *Proceedings of AIIDE '10*, 2010.

[8] David Adams. *Automatic Generation of Dungeons for Computer Games*. University of Sheffield, UK, 2002.

[9] Roland Van Der Linden, Ricardo Lopes, and Rafael Bidarra. Procedural generation of dungeons. In *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES 2014*, pages 78–79, 2014.

[10] Josep Valls-Vargas, Jichen Zhu, and Santiago Ontanon. Graph grammar-based controllable generation of puzzles for a learning game about parallel programming. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*. ACM New York, NY, USA 2017, August 2017.

[11] Kate Compton and Mateas Michael. Procedural level design for platform games. In *AIIDE 2006*, pages 109–111. American Association for Artificial Intelligence 2006, June 2006.

[12] Wang Hanqing, Koji Mikami, and Kunio Kondo. *A Research on the method of Automatic Map Generation of Platform Game*. Tokyo University of Technology, Japan, 2010.

[13] Santiago Londono and Olana Missura. Graph grammars for super mario bros * levels. In *Sixth FDG Workshop on Procedural Content Generation, At Asilomar, Pacific Grove, CA, USA*, June 2015.

[14] Joris Dormans. Adventures in level design: generating missions and spaces for action adventure games. In *PCGames '10 Proceedings of the 2010 Workshop on Procedural Content Gen-*

eration in Games, June 2010.
[15] Joris Dormans and Sander Bakkes. Generating missions and spaces for adaptable play experiences. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):216–228, 2011.
[16] Roland Van Der Linden, Ricardo Lopes, and Rafael Bidarra. Designing procedurally generated levels. In *2013 AIIDE Workshop*, pages 78–79, 2014.
[17] Ernestn Adams. *Fundamentals of Game Design, Second Edition*. Pearson Education Inc, New Riders 1249 Eighth Street Berkeley, CA 94710, 2010.
[18] Mohammad M. Khajah, Brett D. Roads, and Robert V. Lindsey. Designing engaging games using bayesian optimization. In *CHI '16 Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5571–5582, 2016.
[19] María V. Aponte, Guillaume Levieux, and Stephane Natkin. Measuring the level of difficulty in single player video games. *Entertainment Computing, 2011*, 2(4):205–213, 2011.
[20] James Fraser, Michael Katchabaw, and Robert E. Mercer. A methodological approach to identifying and quantifying video game difficulty factors. *Entertainment Computing, 2014*, 2014.