Hochschule Kempten
University of Applied Sciences

BACHELORARBEIT

Fakultät Informatik
Studiengang Game Engineering

Programmatic Calculations for Adjusting Dynamic Gameplay Difficulty

Mohamed Bouchemlal

Verfasser:	Mohamed Bouchemlal
Prüfer:	Prof. Dr. René Bühling
Arbeit vorgelegt am:	20.06.2023
Anschrift des Verfassers:	Mohamed Bouchemlal Reichlinstraße 12, 87439 Kempten

Acknowledgment

I have always been fascinated by video games and considered them as pieces of art not only on the artistic and creative side, but also on the technical and structural side. For this reason, I have been interested in the coding and programming part of the development of video games. And I think that this thesis was a good opportunity for me to improve my logical and mathematical skills in this field.

I would like to thank my supervisor Prof. Dr. René Bühling for firstly having me as a student, and secondly for all the back-and-forth discussions I had with him that were not only full of useful comments, advice and encouragement, but also fun and educational.

I would like to thank my family for all the kind of support they gave me throughout my life.

I would like to appreciate all my friends for the support they provided me throughout the development of my projects and the writing of this thesis.

Table of Contents

1. Introduction

1.1 Background

Video games are a big part of nowadays' entertainment. To keep a consumer entertained is very crucial for having a successful product. When it comes to video games, keeping a player entertained means keeping him engaged and maintain a flow state. Flow is defined by Csikszentmihalyi as "*a state in which people are so involved in an activity that nothing else seems to matter; the experience is so enjoyable that people will continue to do it even at great cost, for the sake of doing it*" [Cziksentmihaly90]. Reaching a state of flow in video games is dependent on many factors, but mainly on one major factor, which is difficulty. A game that is too easy gives the player a feeling of boredom while a game that is too hard gives the player a feeling of frustration. So, finding a balanced spot in between for the player is as Vince Palban stated, "*a complex and relatively ambiguous undertaking*" [Palban21]. Besides that, players enjoy games differently; some people want to be tested and encounter hard challenges, some want to explore and experience the game's mechanics, while others with no gaming skills want to just enjoy the story [Bycer20]. This let us draw the conclusion that, given the different motivations players might have for playing a game and the level of their skills, leads the developers to take in consideration different approaches to design and implement difficulty in their games, with the goal of increasing challenges progressively without increasing frustration" [Bycer20].

Before talking about the different types of difficulty design applications in games, we should first define the term of difficulty in videogames. According to Ricardo Valério: "Difficulty is the amount of skill required by the player to accomplish a goal or progress through the game experience. It can be as simple as jumping from one platform to another, killing a character, defeat a boss fight, etc— which could be designed to respectively be easy, medium or hard to accomplish" [Valério17].

The most common and popular methods to design difficulty in video games are static game difficulty and dynamic difficulty adjustment [Palban21].

- **Static game difficulty (SGD) [Palban21]:** is designed around the idea of having fixed pre-defined difficulty levels (mostly easy, medium and hard), from which the player can select one to play the game on and/or change it later.
- **Dynamic difficulty adjustment (DDA) [Palban21]:** is designed around the idea of adjusting a game's level of difficulty correspondingly to the player's in-game performance.

In both methods, some parameters of the game are changed and tuned to achieve the desired difficulty.

The dynamic difficulty adjustment (DDA) approach is considered on one hand the better approach for maintaining the player's engagement. In static game difficulty (SGD), the difficulty is fixed, which might not match a player's skill level. So, there is always the risk that the player may not find an appropriate challenge suiting his skills, which can negatively affect the player's experience. Whereas in DDA, the game keeps adjusting itself to match the player's performance, which results in giving him an appropriate challenge to overcome, thus having longer engagement. On the other hand, DDA is considered more challenging to design and implement in games. Because it is connected to the player's performance, which is in itself a challenging thing to properly measure.

In this paper I focus on the DDA method and try to come up with a generalized mathematical approach to adjust difficulty based on gameplay challenges that most of action and/or adventure games share. I talk about this more in details in the section "1.3 Approach".

1.2 Related work

The implementation of DDA can vary from genre to genre, from even a game to game in the same genre. In this section we will look at some of the approaches of DDA that some developers used in their games.

In order to implement DDA we need first a way to measure the difficulty in games and a way to measure the player's performance. Having a generalized method to measure difficulty in games is in itself a very difficult if not impossible task to accomplish. That is because of the many variations of video game genres and the different design rules that they can be based on. Many factors can affect a game's difficulty in different ways, such as enemies' artificial intelligence, enemies' number, obstacles, health points of both player and enemies, consumables, game mechanics and the player's skill level, etc. However, Ian Schreiber attempted to generalize – what he named – "Perceived Difficulty" in video games in a formula as follows:

"PerceivedDifficulty = (SkillChallenge + PowerChallenge) – (PlayerSkill + PlayerPower)" [Schreiber10].

This formula is a good starting point on how to apprehend difficulty in games, but unfortunately not precise enough. Some of its elements are related to the game's design which can vary from game to game, such as "SkillChallenge" and "PowerChallenge". Other elements like "PlayerSkill" are even out of the reach of the designers to know. In this thesis, I came up with a mathematical approach to evaluate the difficulty in a game based on formulas that take in consideration the most common gameplay challenges presented in most action and / or adventure games.

The game I chose to apply this mathematical approach on is based on the levels design pattern. It is one of the most used design patterns, in which the player progresses from one level to the next one until he reaches the end. In most cases, the difficulty increases from one level to the next one [Wang21].

After creating a way for evaluating the difficulty in a game's level, we need a way to measure the player's performance in that level to be able to implement DDA. This will be done by comparing both results of the level's difficulty and the player's performance and figure out - based on the difference between them and with the help of a simple algorithm - the next level's difficulty.

The measurement of the player's performance formula I use in this thesis was inspired by the paper "Perception of Difficulty in 2D Platformers Using Graph Grammars" by Henry Fernandez, Koji Mikami and Kunio Kondo, the students of the "Tokyo University of Technology" [Fernandez, Mikami, Kondo18]. In their paper they implemented a method based on Graph Grammars to create procedural levels in 2D platform games. They came up with two formulas for calculating a level's difficulty and the player's performance in that level. After running a survey and comparing the results of both formulas, they got a 0.75 correlation coefficient and concluded that their approach is down the right path, but still needs improvement [Fernandez, Mikami, Kondo18].

One of the interesting implementations of DDA in the horror genre, is in the paper "Keep Calm and Aim for the Head: Biofeedback-Controlled Dynamic Difficulty Adjustment in a Horror Game" by Alena Denisova and Paraschos Moschovitis [Alena, Paraschos22]. Their low-cost DDA system implemented in a horror game uses indirect feedback input from the player through their heart rate to adjust the game's difficulty accordingly. The heart rate of the player is captured in real time using an Arduino micro-controller and a heart sensor that clips to the player's lobe. The parameters that the game alters to adjust the difficulty in real time are the chasing enemy's max speed, max acceleration, and the player's noise radius. Their conclusion was that this approach using the feedback techniques was effective and has a potential to be adopted into everyday gaming activities [Alena, Paraschos22].

Another interesting implementation of DDA can be read in the paper “EEG-Triggered Dynamic Difficulty Adjustment for Multiplayer Games” by Rami Puzis, Guy Shani and Meirav Taieb-Maimon [Rami, Guy, Meirav17]. Their research aimed to increase the excitement of multiplayer games in a non-competitive scenario, like playing against real life friends who can have higher or lower skill levels than the player. They implemented – as they described – “*a passive feedback/affective state regulation method by using the Emotiv EPOC headset to read electroencephalography (EEG) signals”* [Rami, Guy, Meirav17]. Then, based on that, the game adjusts the difficulty. The difficulty gets adjusted by applying game modifications / modes to the game. These modes are designed to either reduce the difficulty for the player, who is not performing well, like boosting his running speed or making him invisible for a while, or increase the difficulty for the player, who is performing very well, by making him – for example - face automated AI turret that shoots him until it gets destroyed. After running a survey with this approach applied in a shooter game, the results showed -according to the author - that players preferred the game with the EEG-triggered DDA and found it more enjoyable and interesting than without EEG-triggered DDA [Rami, Guy, Meirav17].

A different approach based on player modeling through self-reported data was made by Guillermo Romera Rodriguez in his paper “Player Modeling for Dynamic Difficulty Adjustment in Top Down Action Games” [Rodriguez19]. In this approach he run two surveys; the first survey was for the players to evaluate their gaming skills before playing the survey’s game. Based on the self-reported skills data the game’s difficulty got adjusted. Then, the players played the game and had to take the second survey. The second survey evaluated the player’s experience of the game. This new approach of DDA - according to the author – provided an extra layer of information through the self-reported data, but the sample size of the survey has been a drawback, so the results could not be deemed as conclusive [Rodriguez19].

An approach for implementing DDA based on the probability of the player’s death has been done by Robin Hunicke in her paper “The case for dynamic difficulty adjustment in games” [Hunicke05]. In the research, the death’s probability of the player gets calculated with the help of the Gaussian error integral by taking a series of measurements over time of the received damage every tick of the game. Then, based on the calculated death probability, an adjusting intervention event may get triggered that changes the economy of the game. The game’s economy system is based on the supply and demand principle. On the supply side, interventions can change the player’s inventory like health packs, ammunition, weapons, etc, or other variables like player’s damage and weapons’ accuracy. On the demand side, interventions change the impact of enemies on the game, by modifying their number, hit points, AI, spawn location, etc. The results of the study showed – according to the author – that DDA increased the enjoyment of the players who are familiar with FPS games but was unclear if it had an impact on the players who are unfamiliar with FPS games [Hunicke05].

Reading about the papers above, we can conclude that DDA is a vast field to research. It aims to improve the player’s experience and increase his retention through different methods. Different researchers tried different DDA approaches, from focusing on one game’s genre like horror games to trying to generalize it for all games. Some researchers used even external tools outside of the gaming world, like brain and / or heart sensors to get a precise insight into the player’s experience in real time to adjust the difficulty perfectly.

In this thesis, as I mentioned before, I focus on exploring a way to generalize DDA in games with the help of collective mathematical formulas that separately address different gameplay elements most games share, especially action and adventure games.

1.3 Approach

The approach of DDA explored in this thesis is based on the artificial difficulty that focuses on changing game stats, is easier to control and present in almost all game genres [Valério17]. Unlike the designed difficulty, which focuses on introducing or mixing different game mechanics that can be unique to a game [Valério17].

In this thesis, I came up with mathematical formulas that take in consideration the aspect of:

- Terrain's difficulty.
- Obstacles.
- Enemies' Number.
- Enemies' HP (Health Points).
- Enemies' positions.
- Enemies' AI (Artificial Intelligence).

All these game elements are present in most of games and will be used to evaluate a level's difficulty to create a dynamic next level based on the player's performance in the previous level.

Each game element has a so called "Sub-Formula" that only calculates the difficulty of a level in terms of only the aspect of that game element. By combining these Sub-Formulas, we will get a general formula that calculates the whole difficulty of a level. The result of every formula, whether it is a Sub-Formula or the general formula will always be between 0 and 1; with 0 indicating the easiest version and 1 the hardest version.

After calculating a level's difficulty, the player's performance gets be calculated at the end of the level. Based on a comparison between the two, the next level's difficulty will be decided by a simple algorithm.

So, the goal of this thesis is to experiment and explore this approach of having individual mathematical formulas that collectively measure a level's difficulty and test their efficiency and flexibility. A game may lack one of these elements or have additional elements that are unique to it. But for the sake of simplicity, this thesis considers only the elements mentioned above, since they are the most common ones.

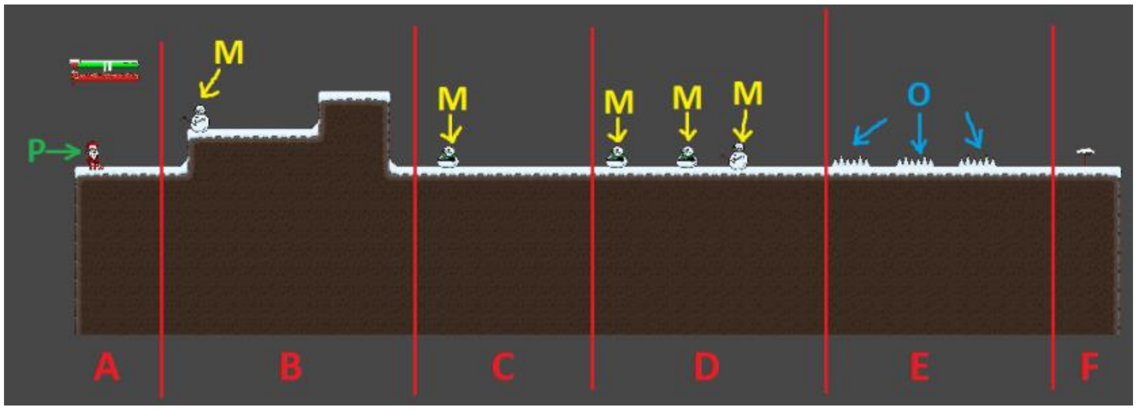
2. The Testing Game

The game used in this study is a 2D action platformer based on the levels' design system. The player must play through different levels to reach the end level and thus finishing the game.

The game's levels are randomly generated using different tiles that get placed near each other. The tiles used are:

• **Start tile:** is the start of the level and will always be placed first.
• **End tile:** is the end of the level and will always be placed last.
• **Enemies' tiles:** are filled with all possible enemy positions. When the game starts, random enemy types get randomly spawned in the pre-determined enemy positions that are set by the designer. There are ten different enemies' tiles that the randomly generating algorithm can randomly choose from to put in the level.
• **Obstacles tile:** can be filled only with obstacles and is always present in a level.

A level can have six to eight tiles in total including the start, end and obstacles tiles.



Screenshot 1: *The components of a randomly generated level.*

Every randomly generated level consists of the following components:

- **P**: is the player (Santa Claus).
- **M**: are the monsters aka enemies.
- **O**: are obstacles.
- **A**: is the start tile.
- **B, C & D**: are the enemies tiles.
- **E**: is the obstacles' tile.
- **F**: is the end tile

3. Evaluation Formulas

3.1 Evaluating a Level's Difficulty

All the formulas $\boldsymbol{Fi}$, $\boldsymbol{i} \in \mathbb{N}$ represent the difficulty of one of the gameplay challenges mentioned in the section "*1.3 Approach*". All the formulas $\boldsymbol{Fi} \in [\boldsymbol{0}, \boldsymbol{1}]$ have a result between **0** and **1**, with **0** being the difficulty of a specific gameplay challenge at its easiest version while **1** at its hardest version. A result in between can be interpreted as relatively easy or hard.

The formulas $\boldsymbol{F1}, \boldsymbol{F2}, \boldsymbol{F3}, \boldsymbol{F4}$ and $\boldsymbol{F5}$ are based on an old work.

3.1.1 Tiles/Terrain

The tiles' difficulty takes in consideration the shape of the tiles or a terrain. A very flat tile is considered to be easier to navigate than a tile with different shapes. The difficulty of a tile is to be set by the designer according to his vision for the game. In this game, the more edges and slopes a tile has, the higher its difficulty is. Not only it is challenging to traverse tiles with more edges, but they can also contain hard enemies' positions, which makes the game more challenging.

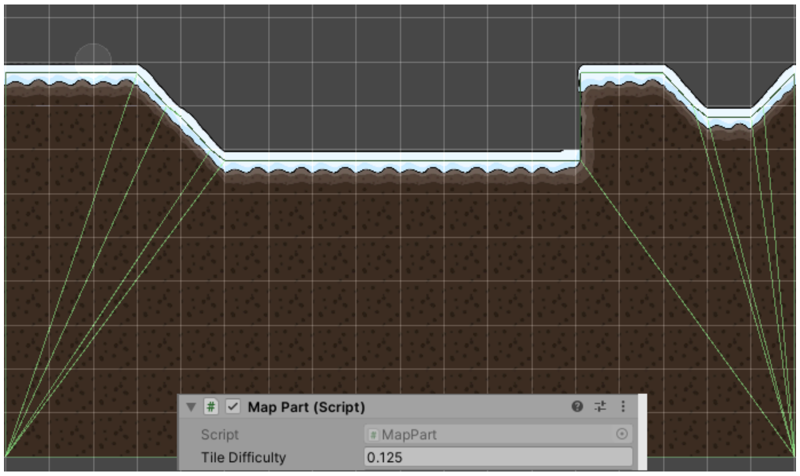
Formula

$$\boldsymbol{F0} = \sum_{i=1}^{m} \frac{t_i}{m} \text{; where } \boldsymbol{m} \in \mathbb{N}_{\boldsymbol{1}} \;\&\; \boldsymbol{t_i} \in [0,1].$$

- $\boldsymbol{t_i}$ is the difficulty of a tile $\mathbf{i}$ present in the level.
- $\boldsymbol{m}$ is the maximum number of tiles a level can have. The designer sets $\boldsymbol{m}$.

Implementation

- My game's levels can have a maximum of five tiles excluding the obstacles, start and end tiles. So $\boldsymbol{m}$ in my situation is 5. Only the enemies' tiles are taken in consideration.
- Every enemies' tile has a script "Map Part" attached to it that has a variable called "Tile Difficulty", as can be seen in Screenshot 2.
- Whenever an enemies' tile gets generated into the level, its "*Tile Difficulty*" gets added to a "*Map Difficulty*" variable in another script. When the level gets fully generated, the "*Map Difficulty*" variable gets divided by five to give in the end the tiles' difficulty of that level.

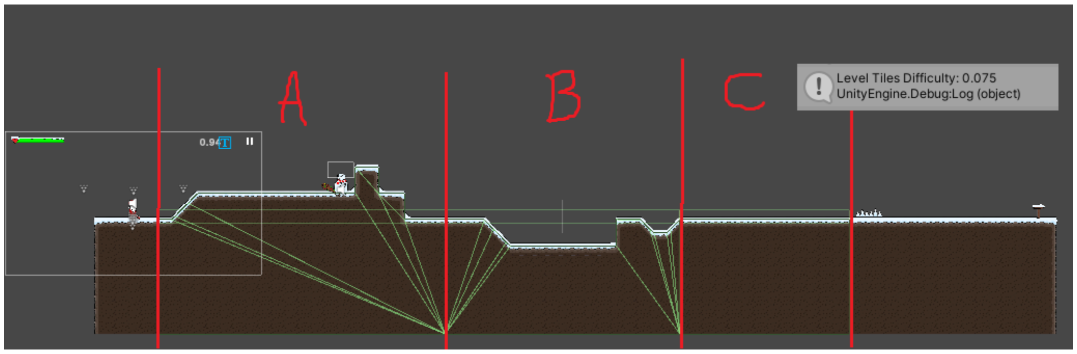

Map Part (Script)

Script: MapPart

Tile Difficulty: 0.125

Screenshot 2: *Example of an enemies' tile and its script "Map Part" with a Tile Difficulty of 0.125.*

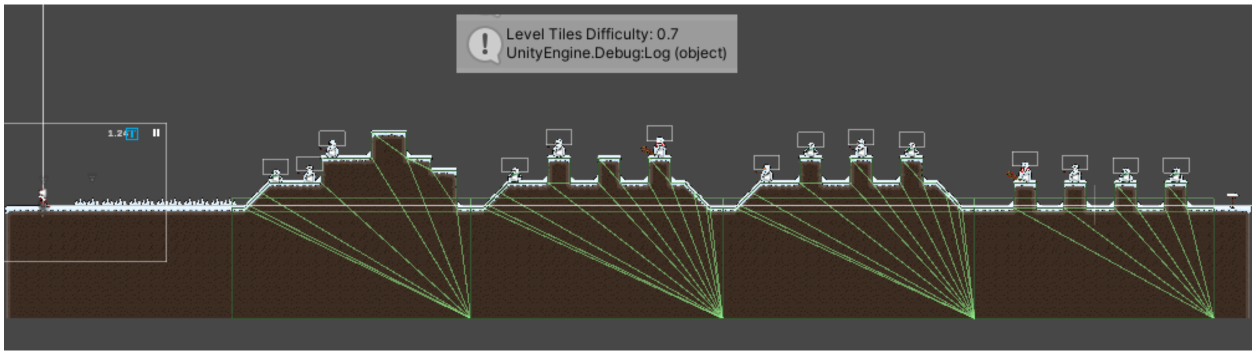
Example



Level Tiles Difficulty: 0.075
UnityEngine.Debug:Log (object)

Screenshot 3: *A randomly generated level with 3 enemies' tiles and a tiles' difficulty of 0.075.*

This randomly generated level in Screenshot 3 has three enemies' tiles (**A**, **B** and **C**) that have the following "*Tile Difficulty*" respectively from left to right: **0.25**, **0.125** and **0**. Adding those numbers and dividing them by 5 (max tiles number in the level) we get the value **0.075**, which is the tiles difficulty of this level. The third enemies' tile is flat, which does not provide any sort of navigation challenges, so it has a value of 0. The value 0.075 corresponds well with the interpretation of the level's difficulty in terms of the tiles. A level with high tiles' difficulty can be seen in Screenshot 4 to draw a comparison.



Level Tiles Difficulty: 0.7
UnityEngine.Debug:Log (object)

Screenshot 4: *A randomly generated level with 4 enemies' tiles and a tiles' difficulty of 0.7.*

In this game, I have eleven enemies' tiles, from which only two are identical. Due to their variety, their "*Tile Difficulty*" value varies from 0 to 1, which makes it impossible to have a combination that reaches the value 0.8 as a result of the tiles' difficulty formula (**F0**).

3.1.2 Number of Enemies

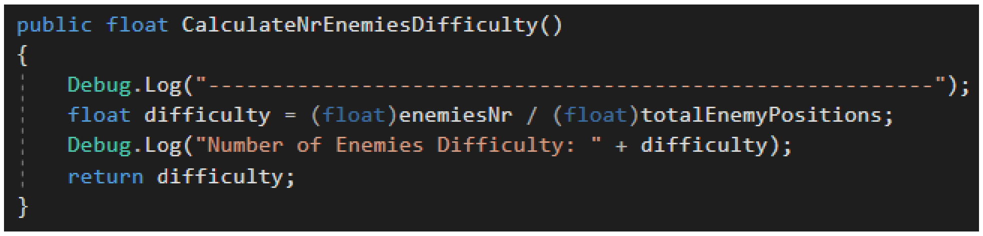
The enemies' number difficulty is about how many enemies are in the level relative to the maximum number of enemies that can be set in that level.

Formula

$$\boldsymbol{F1} = \frac{n}{m}\text{; where } \boldsymbol{n} \in \mathbb{N}_0, \boldsymbol{m} \in \mathbb{N}_1\ \boldsymbol{n} \leq \boldsymbol{m}.$$

- $\boldsymbol{n}$ is the number of enemies in a level.
- $\boldsymbol{m}$ is the max number of enemies that can be spawned in that level. The designer sets $\boldsymbol{m}$.

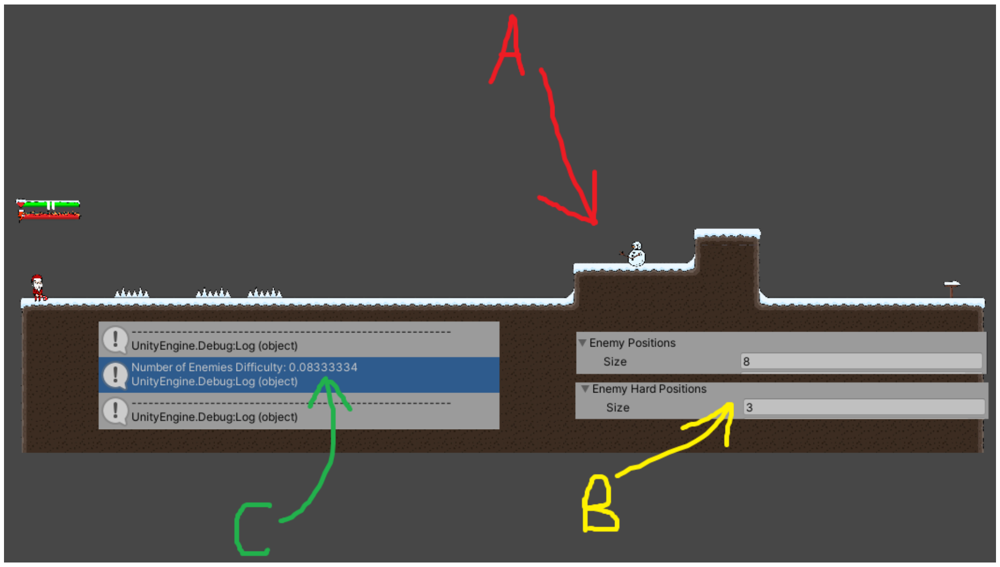
Code



```
public float CalculateNrEnemiesDifficulty()
{
    Debug.Log("----------------------------------------------------------");
    float difficulty = (float)enemiesNr / (float)totalEnemyPositions;
    Debug.Log("Number of Enemies Difficulty: " + difficulty);
    return difficulty;
}
```

Screenshot 5: *Implementation of* $\boldsymbol{F1}$ *in the function "CalulateNrEnemiesDifficulty".*

Example



```
------------------------------------------------------------
UnityEngine.Debug:Log (object)
Number of Enemies Difficulty: 0.08333334
UnityEngine.Debug:Log (object)
------------------------------------------------------------
UnityEngine.Debug:Log (object)
```

```
▼ Enemy Positions
    Size                8
▼ Enemy Hard Positions
    Size                3
```

Screenshot 6: *Example of the number of enemies' difficulty with one enemy in the level.*

In Screenshot 6, the randomly generated level has one enemy as pointed at in **A**. Since the level can have maximumly 11 (8+3) enemies as pointed at in **B**, using the HP formula, we got a result of 0.0833 as pointed at in **C**. Since 0.0833 is quite close to 0, this means that this level is very easy in terms of the number of enemies.

3.1.3 HP (Health / Hits Points)

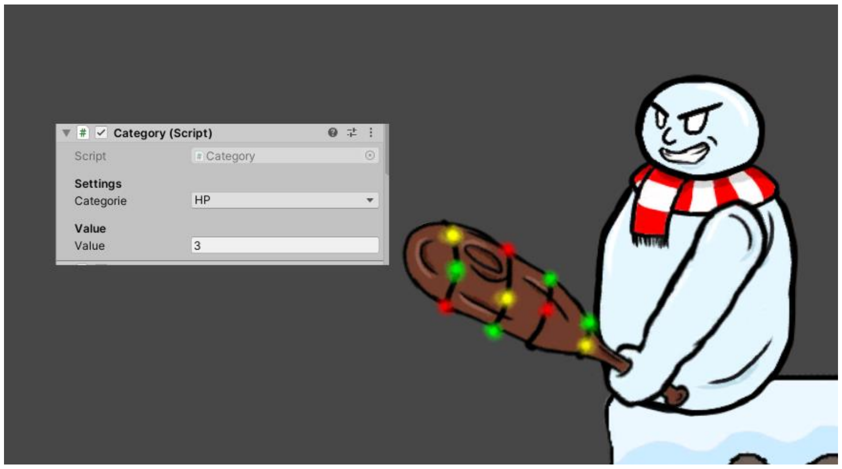
The HP difficulty is about the enemies' HP. The more hits an enemy needs to be hit to die, the higher the difficulty regarding that enemy is. An enemy needs at least one hit to die.

Formula

$$\boldsymbol{F2} = \sum\nolimits_{i \in I} \frac{i}{m} \frac{e_i}{e_t} F1 \text{ ; where } \boldsymbol{n}, \boldsymbol{m} \in \mathbb{N}_1, \boldsymbol{i} \in \boldsymbol{I}, \boldsymbol{n} \leq \boldsymbol{m} \; \& \; \boldsymbol{I} \subset [\boldsymbol{n}, \boldsymbol{m}].$$

- Numbers from $\boldsymbol{n}$ to $\boldsymbol{m}$ are HP variations; with $\boldsymbol{n}$ being the min (1 in this case) number of hits required to kill an enemy and $\boldsymbol{m}$ being the max. The designer sets $\boldsymbol{m}$.
- $\boldsymbol{I}$ is the group of implemented HP variations from $\boldsymbol{n}$ to $\boldsymbol{m}$ in the level.
- $\boldsymbol{i}$ is an HP variation.
- $\boldsymbol{e_i}$ is the number of enemies that has the HP variation $\boldsymbol{i}$.
- $\boldsymbol{e_t}$ is the total number of enemies in the level. If $\boldsymbol{e_t}$ is $\boldsymbol{0}$ the formula gets disregarded **($\boldsymbol{e_t} = \boldsymbol{0} \rightarrow \boldsymbol{F2} = \boldsymbol{0}$)**.
- $\boldsymbol{F1}$ is the result of the number of enemies' formula.

Implementation

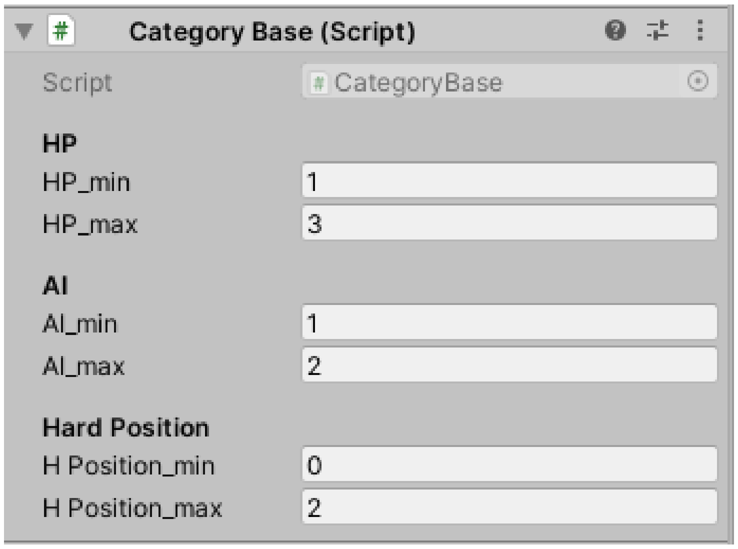

Category (Script)

Script	Category
Settings	
Categorie	HP
Value	
Value	3

Screenshot 7: *The enemy "Heavy SnowMonster" and the to it attached script "Category".*

Since many formulas share the same structure of using categories, I wrote a "*Category*" script that can be attached to any Gameobject, so I can program the formulas easily. This script automatically reads the min and max values from another script I wrote called "*CategoryBase*" that is shown in Screenshot 8. The value in "*Category*" gets clamped between the min and max values in "*CategoryBase*" corresponding to its category.

In Screenshot 7 the "*Heavy SnowMonster*" has the value 3 in the category HP, which gets assigned to his health at the start of the game.

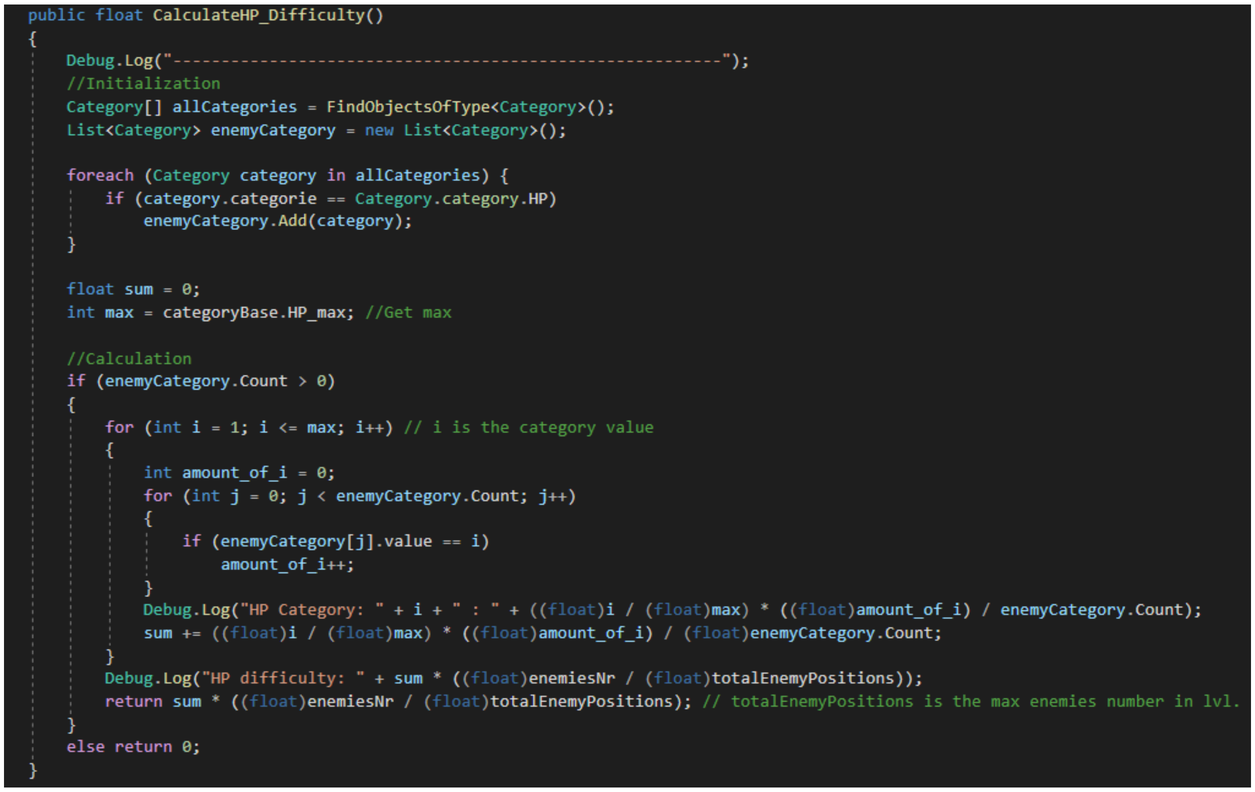


Category Base (Script)

Script	CategoryBase
HP	
HP_min	1
HP_max	3
AI	
AI_min	1
AI_max	2
Hard Position	
H Position_min	0
H Position_max	2

Screenshot 8: *The "CategoryBase" script in the unity editor as a GameObject's component.*

Code

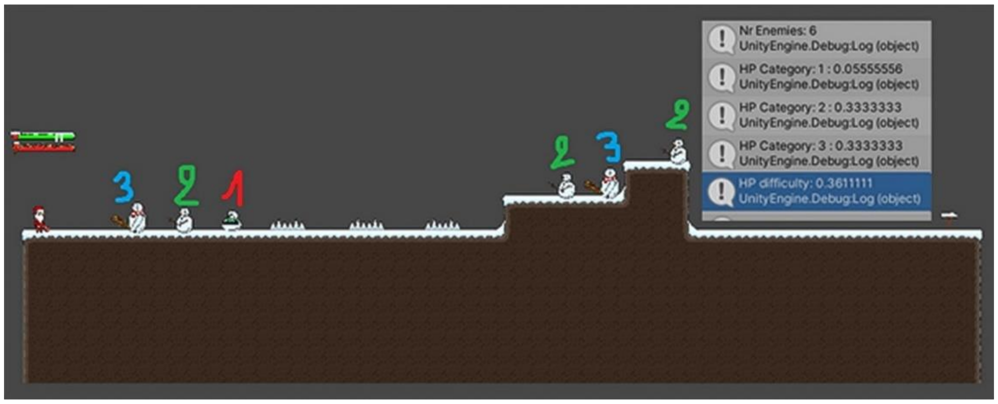


```
public float CalculateHP_Difficulty()
{
    Debug.Log("---------------------------------------------------------");
    //Initialization
    Category[] allCategories = FindObjectsOfType<Category>();
    List<Category> enemyCategory = new List<Category>();

    foreach (Category category in allCategories) {
        if (category.categorie == Category.category.HP)
            enemyCategory.Add(category);
    }

    float sum = 0;
    int max = categoryBase.HP_max; //Get max

    //Calculation
    if (enemyCategory.Count > 0)
    {
        for (int i = 1; i <= max; i++) // i is the category value
        {
            int amount_of_i = 0;
            for (int j = 0; j < enemyCategory.Count; j++)
            {
                if (enemyCategory[j].value == i)
                    amount_of_i++;
            }
            Debug.Log("HP Category: " + i + " : " + ((float)i / (float)max) * ((float)amount_of_i) / enemyCategory.Count);
            sum += ((float)i / (float)max) * ((float)amount_of_i) / (float)enemyCategory.Count;
        }
        Debug.Log("HP difficulty: " + sum * ((float)enemiesNr / (float)totalEnemyPositions));
        return sum * ((float)enemiesNr / (float)totalEnemyPositions); // totalEnemyPositions is the max enemies number in lvl.
    }
    else return 0;
}
```

Screenshot 9: *Implementation of* $F2$ *in the function "CalculateHP_Difficulty".*

Note: *In the code and the console, with the word "category", I basically mean "variation".*

Example



```
Nr Enemies: 6
UnityEngine.Debug:Log (object)
HP Category: 1 : 0.05555556
UnityEngine.Debug:Log (object)
HP Category: 2 : 0.3333333
UnityEngine.Debug:Log (object)
HP Category: 3 : 0.3333333
UnityEngine.Debug:Log (object)
HP difficulty: 0.3611111
UnityEngine.Debug:Log (object)
```

Screenshot 10: *A randomly generated level with its HP difficulty results.*

In this level (Screenshot 10), we have six enemies with three different HP values (**1**,**2** and **3**), and in the console on the right, it is shown, what is the HP difficulty for every HP variation. At last (highlighted with blue) we have their sum multiplied by $\boldsymbol{F1}$, which is the overall HP difficulty of the level.

Looking at the results in the console, we see that the results of the HP variation **2** and **3** are the same (0.333), even though we have three enemies of variation **2** and only two of variation **3**, which makes sense. The higher the HP of a variation is, the more impact has that variation on the result of the overall HP difficulty.

The level's HP difficulty before multiplying it by $\boldsymbol{F1}$ in this case is 0.722 (not shown in image), which means that the level is quite hard in terms of enemies' HP, which makes sense, because 33% (2/6) of the enemies have the max HP value (**3**), 50% (3/6) have the middle HP value (**2**) and only 16% (1/6) of the enemies have the lowest HP value (**1**). But the final result is 0.36, which is lower, because the HP difficulty is related to the number of enemies. Having one enemy with the highest HP value is comparably easier than having six enemies with the highest HP value, therefore it is very important to multiply by $\boldsymbol{F1}$**.**

3.1.4 Number of Enemies' Hard Positions

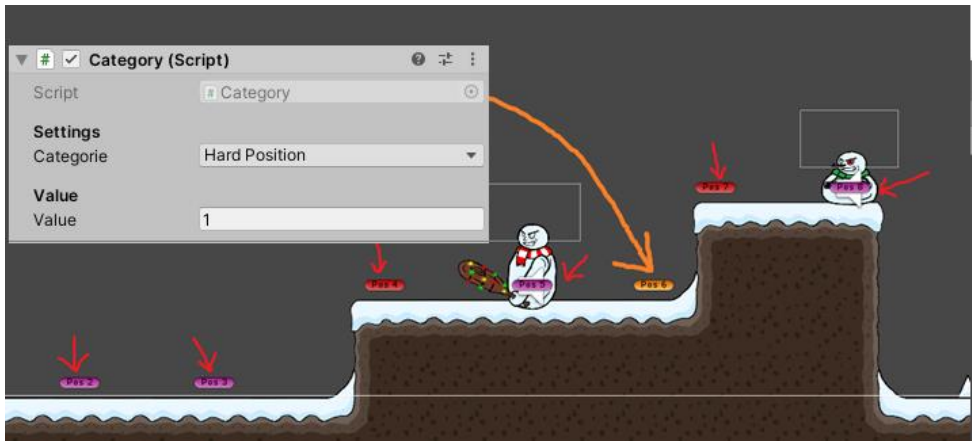
In a level, enemies get placed in different positions. Enemies in some positions are more difficult or dangerous for the player to deal with than in other positions. So, we categorize these positions from normal to hard (from 0 to a max value). It is up to the designer to assign these positions to their corresponding categories as he sees fit.

Formula

$$\boldsymbol{F3} = \sum_{i \in I} \frac{i}{m} \frac{p_i}{p_t}; \text{ where } \boldsymbol{n} \in \mathbb{N}_{\boldsymbol{0}}, \boldsymbol{m} \in \mathbb{N}_{\boldsymbol{1}}, \boldsymbol{i} \in \boldsymbol{I}, \boldsymbol{n} \leq \boldsymbol{m} \;\&\; \boldsymbol{I} \subset [\boldsymbol{n}, \boldsymbol{m}].$$

- Numbers from $\boldsymbol{n}$ to $\boldsymbol{m}$ are hard position's variations; $\boldsymbol{n}$ is the least dangerous position (1 in this case) and $\boldsymbol{m}$ is the most dangerous. The designer sets $\boldsymbol{m}$.
- $\boldsymbol{I}$ is the group of dangerous positions' categories/variations occupied by enemies.
- $\boldsymbol{i}$ is a hard position's variation.
- $\boldsymbol{p_i}$ is the number of occupied dangerous positions of variation $\boldsymbol{i}$.
- $\boldsymbol{p_t}$ is the total number of dangerous positions that can be occupied by enemies in the level. If $\boldsymbol{p_t}$ is **0**, which means that there are no positions to place enemies in, the formula gets disregarded ($\boldsymbol{p_t = 0 \ \rightarrow F3 = 0}$).

Implementation



Settings
Categorie: Hard Position
Value
Value: 1

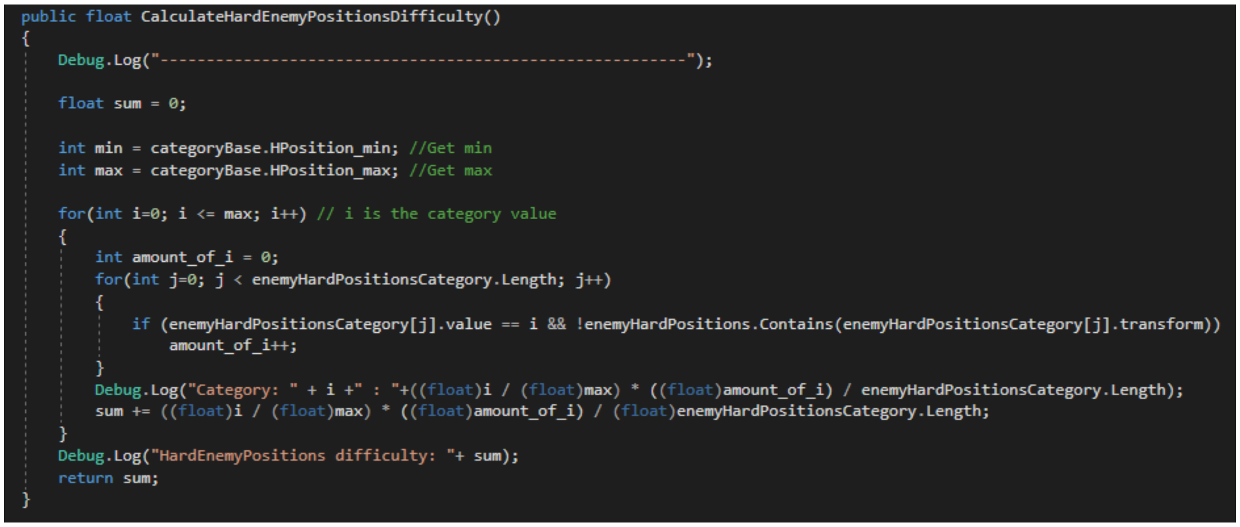
Screenshot 11: *Some of the enemy positions in the level and the script "Category" in the unity editor as a component of the GameObject "Pos 6".*

In Screenshot 11 we can see a cutout of a randomly generated level. In it, some with arrows pointed at enemy positions on the level are labeled with different colors and the name "*Pos i*". These positions have the script "*Category*" attached to them. Each position's color refers to the value in the script "*Category*", which refers to how hard is a position relative to the other ones:

- Violet: has a value of 0 and means that the position is normal and does not get considered as a dangerous position at all (The level has nine in total, but only four are shown in Screenshot 11).
- Orange: has a value of 1 and means that the position is medium difficult (The level has one).
- **Red**: has a value of 2 and means that the position is the hardest (The level has two).

In Screenshot 11, the values seen in the script "*Category*" belong to the "*Pos 6*" in orange, which has the value 1.

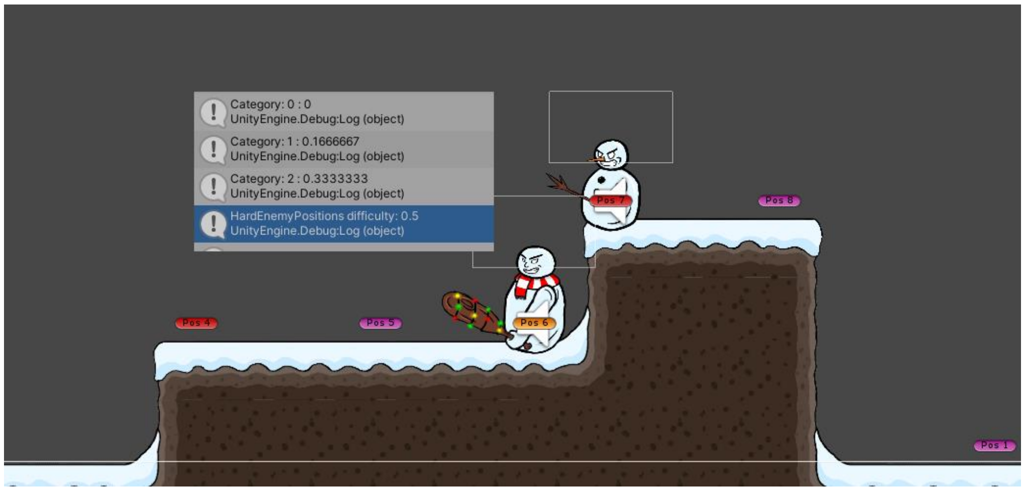
Code



```
public float CalculateHardEnemyPositionsDifficulty()
{
    Debug.Log("---------------------------------------------------------");

    float sum = 0;

    int min = categoryBase.HPosition_min; //Get min
    int max = categoryBase.HPosition_max; //Get max

    for(int i=0; i <= max; i++) // i is the category value
    {
        int amount_of_i = 0;
        for(int j=0; j < enemyHardPositionsCategory.Length; j++)
        {
            if (enemyHardPositionsCategory[j].value == i && !enemyHardPositions.Contains(enemyHardPositionsCategory[j].transform))
                amount_of_i++;
        }
        Debug.Log("Category: " + i +" : "+((float)i / (float)max) * ((float)amount_of_i) / enemyHardPositionsCategory.Length);
        sum += ((float)i / (float)max) * ((float)amount_of_i) / (float)enemyHardPositionsCategory.Length;
    }
    Debug.Log("HardEnemyPositions difficulty: "+ sum);
    return sum;
}
```

Screenshot 12: *Implementation of* $F3$ *in the function "CalculateHardEnemyPositionsDifficulty".*

Example



```
Category: 0 : 0
UnityEngine.Debug:Log (object)
Category: 1 : 0.1666667
UnityEngine.Debug:Log (object)
Category: 2 : 0.3333333
UnityEngine.Debug:Log (object)
HardEnemyPositions difficulty: 0.5
UnityEngine.Debug:Log (object)
```

Screenshot 13: *A cut-out of a randomly generated level and its* $F3$*'s results.*

Since normal positions (**Violet** / easy) have the value 0, their variation's result will always be 0. For the variation 1 (**Orange** / medium), we have the only position occupied, so we got 0.166. As for the variation 2 (**Red** / Hard), we have one occupied position of two, so we got 0.333. The result is the sum of all of them, so we got 0.5. This means that, in terms of positioning the enemies in hard positions in the level, the difficulty is medium.

3.1.5 Number of Smart Enemies (AI)

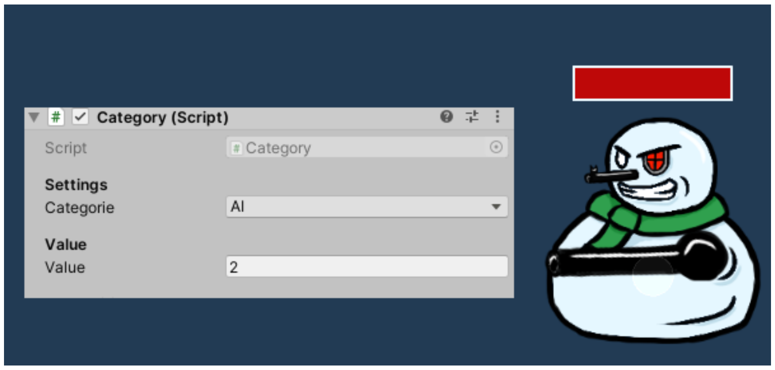
This difficulty is about enemies' artificial intelligence or traits, the more traits an enemy has, the smarter he is considered than other enemies with less traits, and harder for the player to deal with.

Formula

$$\boldsymbol{F4} = \sum\nolimits_{i \in I} \frac{i}{m} \frac{e_i}{e_t} F1 \text{ ; where } \boldsymbol{n}, \boldsymbol{m} \in \mathbb{N}_1, \boldsymbol{i} \in \boldsymbol{I}, \boldsymbol{n} \leq \boldsymbol{m} \text{ \& } \boldsymbol{I} \subset [\boldsymbol{n}, \boldsymbol{m}].$$

- Numbers from $\boldsymbol{n}$ to $\boldsymbol{m}$ are an AI variation; with $\boldsymbol{n}$ being the category of the enemies with the least traits (1 in this case) and $\boldsymbol{m}$ being the category of enemies with the most traits (smartest).
- $\boldsymbol{I}$ is the group of implemented enemy's AI categories/variations from $\boldsymbol{n}$ to $\boldsymbol{m}$.
- $\boldsymbol{i}$ is an AI variation.
- $\boldsymbol{e_i}$ is the number of enemies with an AI category $\boldsymbol{i}$.
- $\boldsymbol{e_t}$ is the total number of enemies in the level. If $\boldsymbol{e_t}$ is **0** the formula gets disregarded **(**$\boldsymbol{e_t} = \boldsymbol{0} \rightarrow \boldsymbol{F4} = \boldsymbol{0}$**)**.
- $\boldsymbol{F1}$ is the result of the number of enemies' formula.

Implementation


Category (Script)

Script: Category

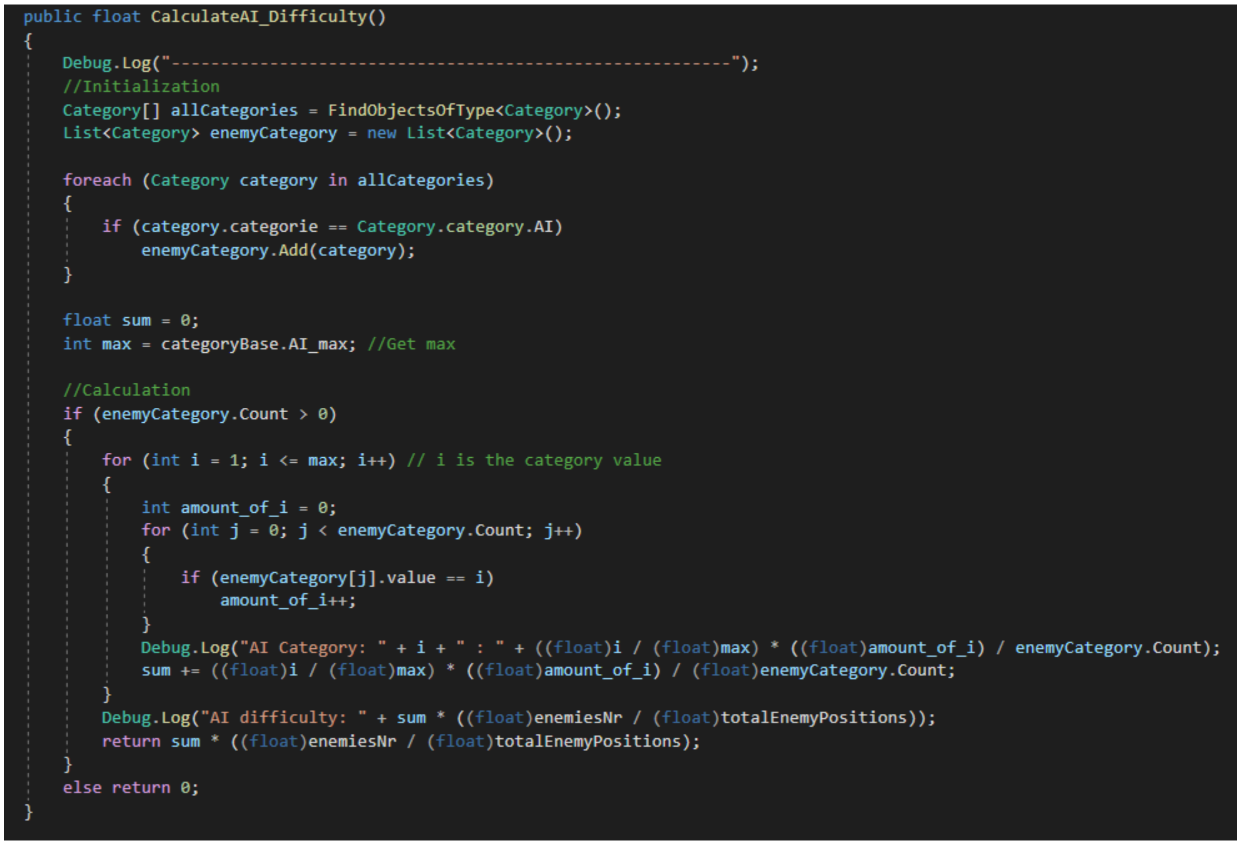
Settings
Categorie: AI

Value
Value: 2

Screenshot 14: *"Range SnowMonster" and the to it attached script "Category".*

Screenshot 14 displays the "*Range SnowMonster*" and its associated "*Category*" script, which has an AI category value of 2. The AI category value ranges from 1 to 2, where 1 represents the min and 2 represents the max level of AI. The "*Range SnowMonster*" has an AI category value of 2 as it can shoot from a distance and rotate its arm to track the player's position. In contrast, the remaining enemies have an AI category value of 1.

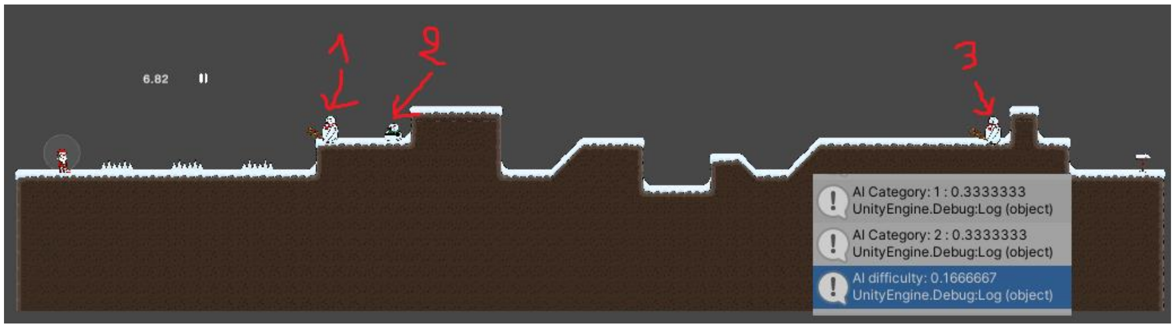
Code



```
public float CalculateAI_Difficulty()
{
    Debug.Log("------------------------------------------------------------");
    //Initialization
    Category[] allCategories = FindObjectsOfType<Category>();
    List<Category> enemyCategory = new List<Category>();

    foreach (Category category in allCategories)
    {
        if (category.categorie == Category.category.AI)
            enemyCategory.Add(category);
    }

    float sum = 0;
    int max = categoryBase.AI_max; //Get max

    //Calculation
    if (enemyCategory.Count > 0)
    {
        for (int i = 1; i <= max; i++) // i is the category value
        {
            int amount_of_i = 0;
            for (int j = 0; j < enemyCategory.Count; j++)
            {
                if (enemyCategory[j].value == i)
                    amount_of_i++;
            }
            Debug.Log("AI Category: " + i + " : " + ((float)i / (float)max) * ((float)amount_of_i) / enemyCategory.Count);
            sum += ((float)i / (float)max) * ((float)amount_of_i) / (float)enemyCategory.Count;
        }
        Debug.Log("AI difficulty: " + sum * ((float)enemiesNr / (float)totalEnemyPositions));
        return sum * ((float)enemiesNr / (float)totalEnemyPositions);
    }
    else return 0;
}
```

Screenshot 15: *Implementation of* $\boldsymbol{F4}$ *in the function "CalculateAI_Difficulty".*

Example



```
AI Category: 1 : 0.3333333
UnityEngine.Debug:Log (object)
AI Category: 2 : 0.3333333
UnityEngine.Debug:Log (object)
AI difficulty: 0.1666667
UnityEngine.Debug:Log (object)
```

Screenshot 16: *An example of* $\boldsymbol{F4}$*'s results of a randomly generated level with two enemies with an AI value of 1 and one enemy with an AI value of 2.*

In Screenshot 16 we see a randomly generated level with three enemies (**1**, **2** and **3**). Looking at the formula $\boldsymbol{F4}$, we can see that the results showed on the Unity's console match the level's content in terms of AI. We have one "*Range SnowMonster*" (**2**) with an AI value of 2, which gives us the result 0.33 in his AI category ("*AI Category 2*"), and two "*Heavy SnowMonsters*" with an AI value of 1, which gives us a result of 0.33 in their AI Category ("*AI Category 1*"). The final result of $\boldsymbol{F4}$ is **0.16**, which strongly matches the level in terms of AI. We have three enemies in the level out of twelve (max possible number of enemies in this generated level) and only one of those three has the highest AI value 2. That indicates that the level is easy in terms of AI visually (through looking at the level) and theoretically (through the results of $\boldsymbol{F4}$).

3.1.6 Obstacles

This difficulty variation is about the obstacles. There are two ways the obstacles can affect a level's difficulty, first is their number and second is the distances between them. Dealing with a group of obstacles that are close to each other is harder than dealing with each obstacle alone at a time.

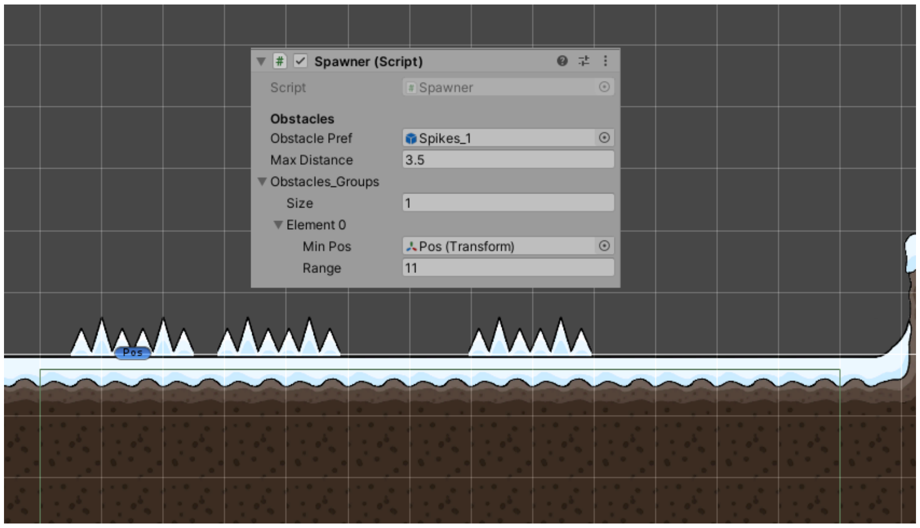
Formula

$$\boldsymbol{F5} = \frac{n}{m} w_1 + \left(\sum_{j=1}^{K} \left(\sum_{i=1}^{n_j - 1} \left(D_m - \left| \vec{a}_{ji} - \vec{a}_{ji_{-1}} \right| \right) \frac{1}{D_m (n_j - 1)} \right) \frac{1}{k} \right) \frac{n}{m} w_2$$

; where $\boldsymbol{n}, \boldsymbol{k} \in \mathbb{N}_{\boldsymbol{0}}$, $\boldsymbol{m}, \boldsymbol{i}, \boldsymbol{j} \in \mathbb{N}_{\boldsymbol{1}}$ $\boldsymbol{n} \leq \boldsymbol{m}$ & $\boldsymbol{w_1}, \boldsymbol{w_2} \in [0,1]$.

- $\boldsymbol{n}$ is the number of obstacles in the level and $\boldsymbol{m}$ is the max number of obstacles that can be put in that level. The designer sets $\boldsymbol{m}$.
- $\boldsymbol{w_1}$& $\boldsymbol{w_2}$ are weights that represent the importance of each element of the formula. The designer sets them according to what he sees fit.
- $\boldsymbol{K}$ is the number of obstacle groups inside a level, a group of obstacles has at least one obstacle. If $\boldsymbol{K}$ equals 0 (we have no group of obstacles in the level) then $\boldsymbol{F5}$ gets disregarded $(\boldsymbol{K = 0} \rightarrow \boldsymbol{F5 = 0})$.
- $\boldsymbol{n_j}$ is the number of obstacles in a group of obstacles $\boldsymbol{j}$. If $\boldsymbol{n_j}$ is **1**, the whole summations part (distances part) gets disregarded (becomes 0). Because we need at least two obstacles for a distance to exist. If there is no distance, then the level is at its easiest form in terms of distances between obstacles.
- $\vec{\boldsymbol{a}}_{\boldsymbol{ij}}$ (Vector) is the position of an obstacle $\boldsymbol{i}$ in the group of obstacles **j**.
- $\boldsymbol{D_m}$ is the max distance between two obstacles, if a distance between two obstacles is greater than $\boldsymbol{D_m}$, then they belong to different group of obstacles.
- The first element/part of the formula handles the difficulty relatively to **the number of the obstacles** in the level. The second element/part handles the difficulty relatively to **the distances between the obstacles** in the level.

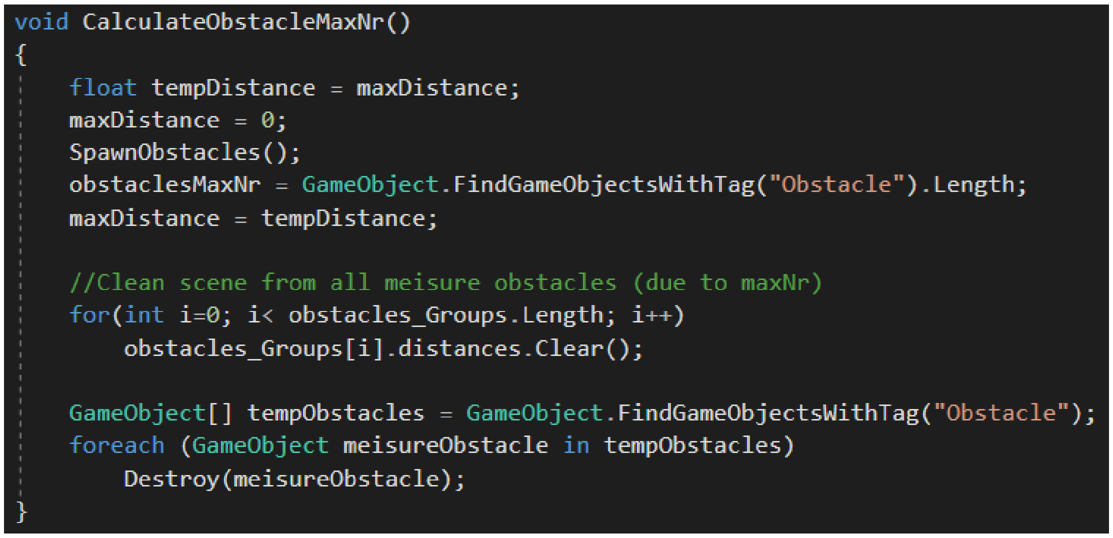
Implementation


Spawner (Script)
Script: Spawner
Obstacles
Obstacle Pref: Spikes_1
Max Distance: 3.5
Obstacles_Groups
Size: 1
Element 0
Min Pos: Pos (Transform)
Range: 11

Screenshot 17: *Obstacles tile and Obstacles' section in the script "Spawner".*

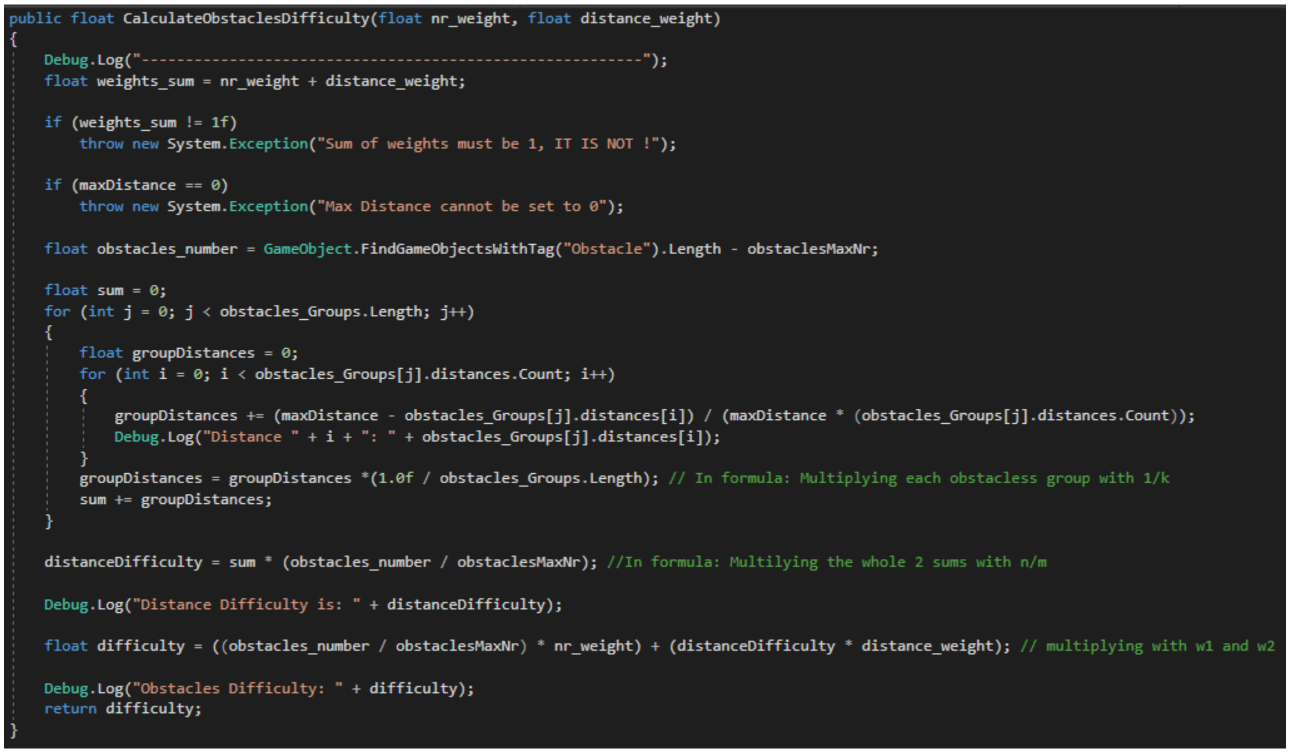
In the testing level, I have a “*Spawner*” script that spawns the enemies and obstacles randomly. In the obstacles’ section there’s a prefab of the obstacle, the max distance (set to 3.5) and an array of the obstacles’ groups in the level. Every group has a start position of the first obstacle (Min Pos), a range that limits the spawning of the obstacles of that group and a list of the distances between those obstacles, which are hidden in the inspector, because they get randomly determined in code. The latter list gets used in the calculation of the formula $F5$.

Code



```
void CalculateObstacleMaxNr()
{
    float tempDistance = maxDistance;
    maxDistance = 0;
    SpawnObstacles();
    obstaclesMaxNr = GameObject.FindGameObjectsWithTag("Obstacle").Length;
    maxDistance = tempDistance;

    //Clean scene from all meisure obstacles (due to maxNr)
    for(int i=0; i< obstacles_Groups.Length; i++)
        obstacles_Groups[i].distances.Clear();

    GameObject[] tempObstacles = GameObject.FindGameObjectsWithTag("Obstacle");
    foreach (GameObject meisureObstacle in tempObstacles)
        Destroy(meisureObstacle);
}
```

Screenshot 18: *Calculating the max number of obstacles **m** of $F5$ in the function “CalculateObstacleMaxNr”.*

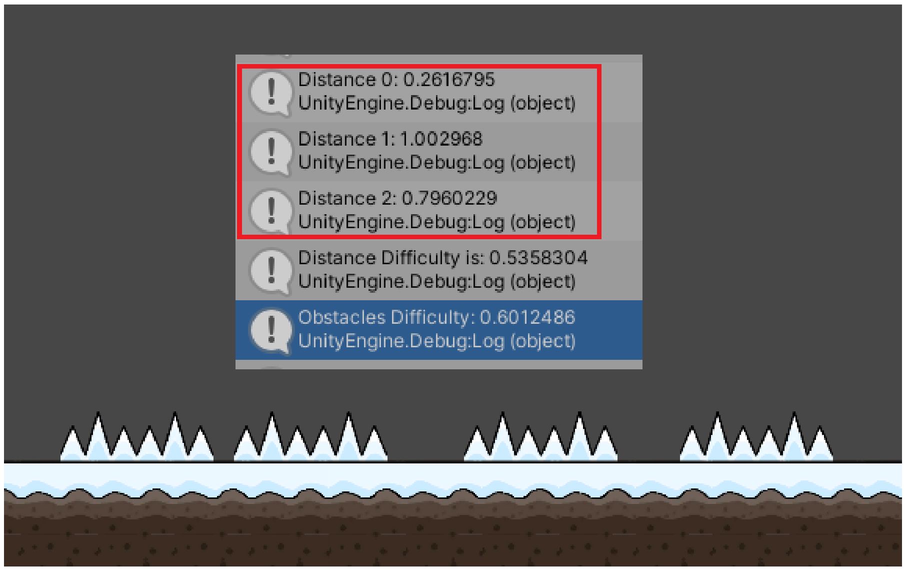


```
public float CalculateObstaclesDifficulty(float nr_weight, float distance_weight)
{
    Debug.Log("------------------------------------------------------");
    float weights_sum = nr_weight + distance_weight;

    if (weights_sum != 1f)
        throw new System.Exception("Sum of weights must be 1, IT IS NOT !");

    if (maxDistance == 0)
        throw new System.Exception("Max Distance cannot be set to 0");

    float obstacles_number = GameObject.FindGameObjectsWithTag("Obstacle").Length - obstaclesMaxNr;

    float sum = 0;
    for (int j = 0; j < obstacles_Groups.Length; j++)
    {
        float groupDistances = 0;
        for (int i = 0; i < obstacles_Groups[j].distances.Count; i++)
        {
            groupDistances += (maxDistance - obstacles_Groups[j].distances[i]) / (maxDistance * (obstacles_Groups[j].distances.Count));
            Debug.Log("Distance " + i + ": " + obstacles_Groups[j].distances[i]);
        }
        groupDistances = groupDistances *(1.0f / obstacles_Groups.Length); // In formula: Multiplying each obstacless group with 1/k
        sum += groupDistances;
    }

    distanceDifficulty = sum * (obstacles_number / obstaclesMaxNr); //In formula: Multilying the whole 2 sums with n/m

    Debug.Log("Distance Difficulty is: " + distanceDifficulty);

    float difficulty = ((obstacles_number / obstaclesMaxNr) * nr_weight) + (distanceDifficulty * distance_weight); // multiplying with w1 and w2

    Debug.Log("Obstacles Difficulty: " + difficulty);
    return difficulty;
}
```

Screenshot 19: *Implementation of $F5$ in the function “CalculateObstaclesDifficulty”.*

For the sake of demonstration, we give an equal weight value of 0.5 to both elements (number of obstacles and distance weights) of the formula $F5$.

Example



```
Distance 0: 0.2616795
UnityEngine.Debug:Log (object)
Distance 1: 1.002968
UnityEngine.Debug:Log (object)
Distance 2: 0.7960229
UnityEngine.Debug:Log (object)
Distance Difficulty is: 0.5358304
UnityEngine.Debug:Log (object)
Obstacles Difficulty: 0.6012486
UnityEngine.Debug:Log (object)
```

Screenshot 20: *$\boldsymbol{F}5$'s results of a randomly generated level.*

As seen in Screenshot 20, we have four obstacles, which have three distances between them. The distances shown in the console match the visual distances in Screenshot 20. The first distance is 0.26 which makes crossing these two obstacles for the player hard, but as for the second and last distances, they have respectively a value of 1.00 and 0.79, which makes it quite possible for the player to cross the last two obstacles. Combining all the three distances, we can say that this obstacles' group has a slightly above medium difficulty in terms of distances between obstacles, thus giving us a distance difficulty of 0.53.

In terms of the obstacles number, we have in the level 4 obstacles out of 6 (max obstacles number), which gives a value of 0.66. Combining the latter value with the distances' one, we get a final obstacles difficulty of 0.60, which translates well the output of the randomly generated obstacles in the level.

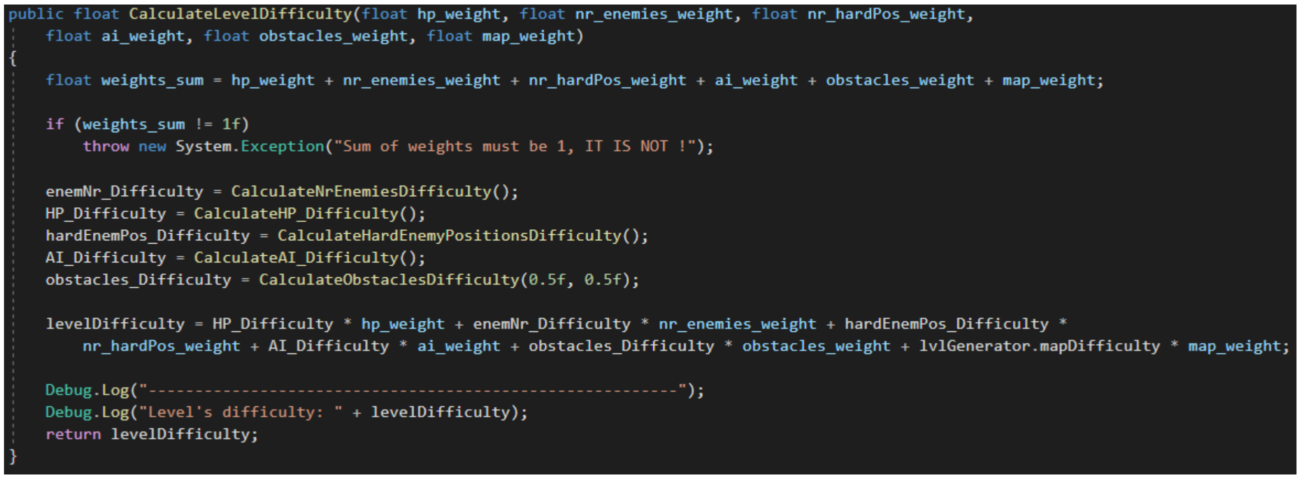
3.1.7 The Level's Difficulty Formula

$$\boldsymbol{F} = \sum_{i=1}^{6} F_i w_i$$

- $\boldsymbol{F_i} \in [\boldsymbol{0}, \boldsymbol{1}]$ are the results of each difficulty variation.
- $\boldsymbol{w_i} \in [\boldsymbol{0}, \boldsymbol{1}]$ are the weight of each difficulty variation. Where $\sum_i \boldsymbol{w_i} = \boldsymbol{1}$.

This formula is a combination of all the six previous gameplay difficulty variations mentioned above and it concludes a whole level's difficulty.

Implementation



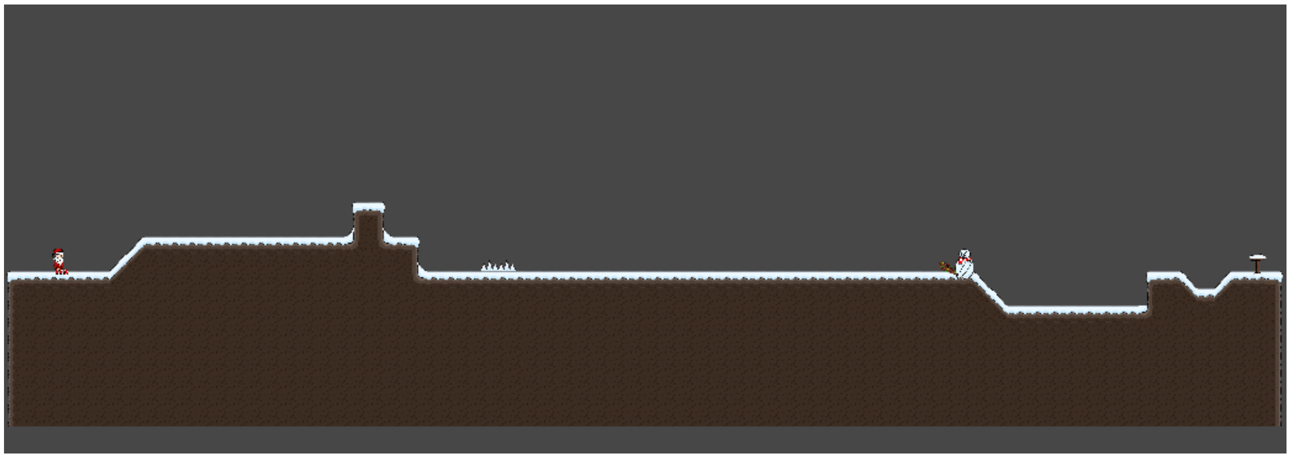
```
public float CalculateLevelDifficulty(float hp_weight, float nr_enemies_weight, float nr_hardPos_weight,
    float ai_weight, float obstacles_weight, float map_weight)
{
    float weights_sum = hp_weight + nr_enemies_weight + nr_hardPos_weight + ai_weight + obstacles_weight + map_weight;

    if (weights_sum != 1f)
        throw new System.Exception("Sum of weights must be 1, IT IS NOT !");

    enemNr_Difficulty = CalculateNrEnemiesDifficulty();
    HP_Difficulty = CalculateHP_Difficulty();
    hardEnemPos_Difficulty = CalculateHardEnemyPositionsDifficulty();
    AI_Difficulty = CalculateAI_Difficulty();
    obstacles_Difficulty = CalculateObstaclesDifficulty(0.5f, 0.5f);

    levelDifficulty = HP_Difficulty * hp_weight + enemNr_Difficulty * nr_enemies_weight + hardEnemPos_Difficulty *
        nr_hardPos_weight + AI_Difficulty * ai_weight + obstacles_Difficulty * obstacles_weight + lvlGenerator.mapDifficulty * map_weight;

    Debug.Log("---------------------------------------------------------");
    Debug.Log("Level's difficulty: " + levelDifficulty);
    return levelDifficulty;
}
```

Screenshot 21: *Implementation of* $\boldsymbol{F}$ *in the function "CalculateLevelDifficulty".*

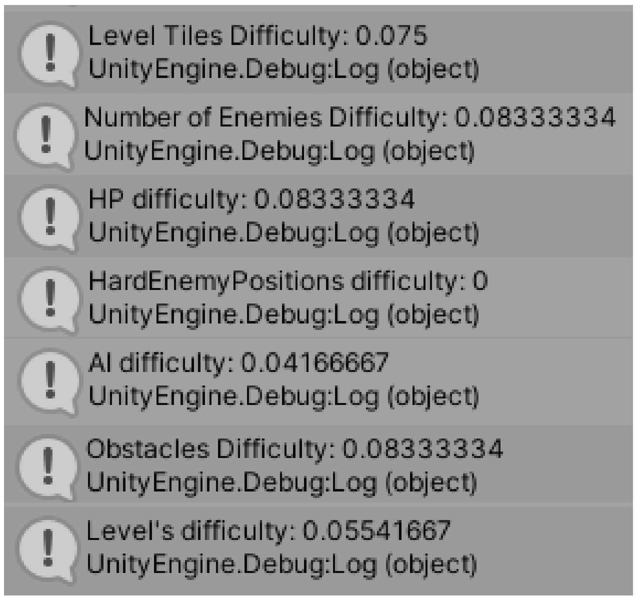
Listed below are the weights I chose to assign to each formula $\boldsymbol{F_i}$ in my game and why:

- $\boldsymbol{w_0}$ **(Tiles) = 0.1**: As a designer for this game, I felt that the tiles do not have a huge impact on the difficulty in my game.
- $\boldsymbol{w_1}$ **(Number Of Enemies) = 0.3**: I think, having more enemies in my game's levels, make them harder for sure.
- $\boldsymbol{w_2}$ **(HP) = 0.05**: The value I assigned here and in AI $\boldsymbol{w_4}$ is very low because the enemies in my game have either high HP and very low AI or vice versa. So, when the AI difficulty is higher, the HP difficulty is lower and vice versa. Which impacts the overall difficulty of the level, making it not able to reach a high difficulty value of 0.9 for example. If I had enemies who would have high AI and HP value at the same time, I may have changed $\boldsymbol{w_2}$ and $\boldsymbol{w_3}$.
- $\boldsymbol{w_3}$ **(Hard Positions) = 0.3**: The enemies' positions in this game are very important. Having an enemy in a position where the player has room to move and dodge to deal with him is easier than having the same enemy in a tight position or an edge for example.
- $\boldsymbol{w_4}$ **(AI) = 0.05**: Read the third point about HP ($\boldsymbol{w_4}$).
- $\boldsymbol{w_5}$ **(Obstacles) = 0.2**: I think 0.2 is an average value for obstacles.

Example 1 (Easy Level)



Screenshot 22: *An easy randomly generated level.*



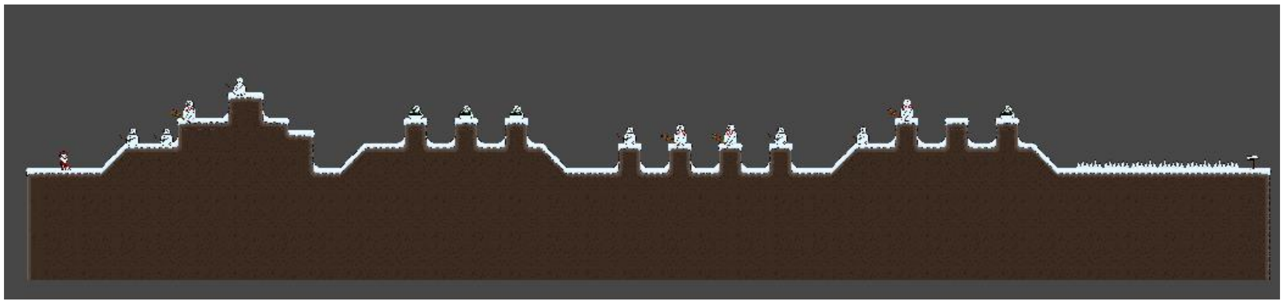
```
Level Tiles Difficulty: 0.075
UnityEngine.Debug:Log (object)
Number of Enemies Difficulty: 0.08333334
UnityEngine.Debug:Log (object)
HP difficulty: 0.08333334
UnityEngine.Debug:Log (object)
HardEnemyPositions difficulty: 0
UnityEngine.Debug:Log (object)
AI difficulty: 0.04166667
UnityEngine.Debug:Log (object)
Obstacles Difficulty: 0.08333334
UnityEngine.Debug:Log (object)
Level's difficulty: 0.05541667
UnityEngine.Debug:Log (object)
```

Screenshot 23: *Results of $\boldsymbol{Fi}$ and $\boldsymbol{F}$ of the randomly generated level in the "Screenshot 22".*

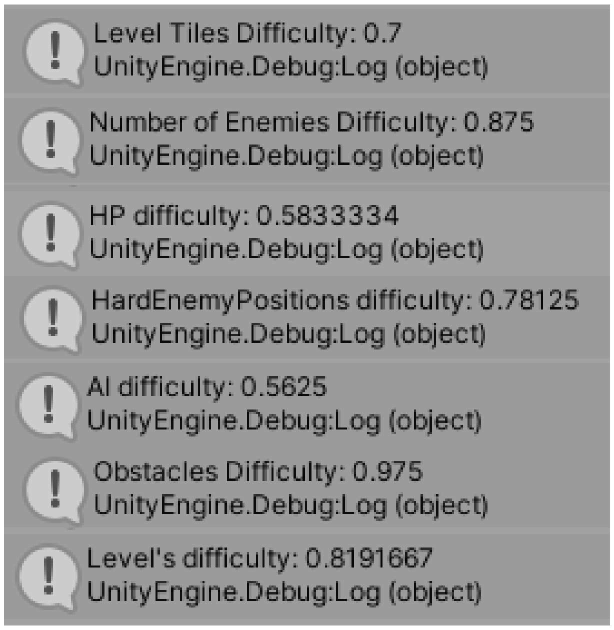
As we can see in Screenshots 22 and 23, the results of $\boldsymbol{Fi}$ and $\boldsymbol{F}$ match the content of the level in Screenshot 22. All the tiles are almost flat so $\boldsymbol{F0}$ **= 0.075**. We have 1 enemy in the level out of 12 possible enemies, so we got $\boldsymbol{F1}$ **= 0.083**. Additionally, the only enemy in the level is in a flat position, so no hard position is occupied, thus $\boldsymbol{F3}$ **= 0**. As for the HP and AI, we have one enemy with an HP of 3 out 3 and lower AI value of 1 out of 2, and since that enemy is the only enemy in the level, we get $\boldsymbol{F2}$ **= 0.083** and $\boldsymbol{F4}$ **= 0.041** respectively. As for the obstacles, we have only one. So only the number of obstacles gets regarded, therefor we get $\boldsymbol{F5}$ **= 0.083**.

Summing up $\boldsymbol{Fi}$ after multiplying them by their weights we get $\boldsymbol{F}$ **= 0.05**, which indicates that the level is very easy. Seeing the level's content in Screenshot 22, we can confirm that the level is actually very easy.

Example 2 (Hard Level)



Screenshot 24: *A hard randomly generated level.*



```
Level Tiles Difficulty: 0.7
UnityEngine.Debug:Log (object)
Number of Enemies Difficulty: 0.875
UnityEngine.Debug:Log (object)
HP difficulty: 0.5833334
UnityEngine.Debug:Log (object)
HardEnemyPositions difficulty: 0.78125
UnityEngine.Debug:Log (object)
AI difficulty: 0.5625
UnityEngine.Debug:Log (object)
Obstacles Difficulty: 0.975
UnityEngine.Debug:Log (object)
Level's difficulty: 0.8191667
UnityEngine.Debug:Log (object)
```

Screenshot 25: *Results of $\boldsymbol{Fi}$ and $\boldsymbol{F}$ of the randomly generated level in the "Screenshot 24".*

In Screenshot 24 we see a very hard level with four tiles that are mostly filled with enemies. Screenshot 25 displays each difficulty of the level's components and its overall difficulty. Looking at the results of $\boldsymbol{Fi}$ and $\boldsymbol{F}$, and the level while following the same analysis approach used in "*Example 1 (Easy Level)*" of this section, we can see that they match.

Note: *It is impossible in my game to have a level with a difficulty above 0.85 due to the design of the game; In the game I have multiple variations in every gameplay challenge. These variations have difficulty values from 0 to 1. So, having a combination of them will never result into having a very high value such as 0.9 or 1.*

3.2 Evaluating the Player's Performance

The player's performance formula (*PP Formula*) used in this thesis - as mentioned in the "*1.2 Related Work*" section - is inspired by the player's performance calculation in the paper "*Perception of Difficulty in 2D Platformers Using Graph Grammars*" by Henry Fernandez, Koji Mikami and Kunio Kondo, the students of the "Tokyo University of Technology" [Fernandez, Mikami, Kondo18].

The PP Formula takes in consideration three elements to calculate the player's performance. Those elements are the damage taken by the player, the number of killed enemies and the time took the player to finish the level.

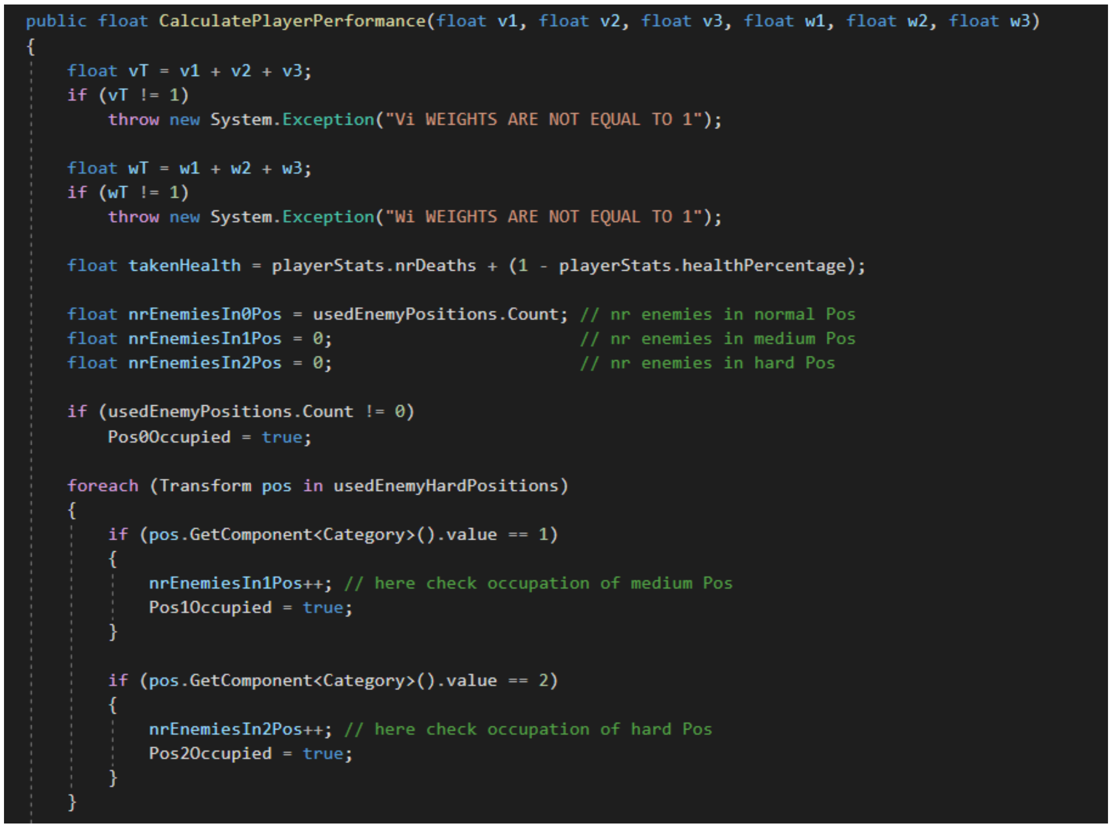
3.2.1 Player's Performance Formula

$$\boldsymbol{F_p = \frac{1}{(1+D)} w_1 + \left(\frac{E_N}{E_{tN}} v_1 + \frac{E_M}{E_{tM}} v_2 + \frac{E_H}{E_{tH}} v_3\right) w_2 + \frac{T_e}{T_f} w_3}$$

- $\boldsymbol{F_p} \in [\mathbf{0}, \mathbf{1}]$ **0** (Worst performance). **1** (Best performance).
- $\boldsymbol{D}$ is the damage taken by the player (a value of 1 means 1 death etc).
- $\boldsymbol{E_N}$ is the number of killed enemies in a normal position in the level.
- $\boldsymbol{E_M}$ is the number of killed enemies in a medium position in the level.
- $\boldsymbol{E_H}$ is the number of killed enemies in a hard position in the level.
- $\boldsymbol{E_{tN}}$ is the total number of enemies in a normal position (if $\boldsymbol{E_{tN} = 0 \rightarrow \frac{E_N}{E_{tN}} = 1}$) $\Leftrightarrow \boldsymbol{Note}$.
- $\boldsymbol{E_{tM}}$ is the total number of enemies in a medium position (if $\boldsymbol{E_{tM} = 0 \rightarrow \frac{E_M}{E_{tM}} = 1}$) $\Leftrightarrow \boldsymbol{Note}$.
- $\boldsymbol{E_{tH}}$ is the total number of enemies in a hard position if $\boldsymbol{E_{tH} = 0 \rightarrow \frac{E_H}{E_{tH}} = 1}$) $\Leftrightarrow \boldsymbol{Note}$.
- $\boldsymbol{T_e}$ is the time estimated to complete the level.
- $\boldsymbol{T_f}$ is the time that took the player to finish the level. $\boldsymbol{T_e \leq T_f}$
- $\boldsymbol{v_i}$ are the weights of each position variation. It depends on the game -> The designer sets them.
  - $\boldsymbol{v_1 + v_2 + v_3 = 1}$
- $\boldsymbol{w_i}$ are the weights of each element of the formula. It depends on the game -> The designer sets them.
  - $\boldsymbol{w_1 + w_2 + w_3 = 1}$

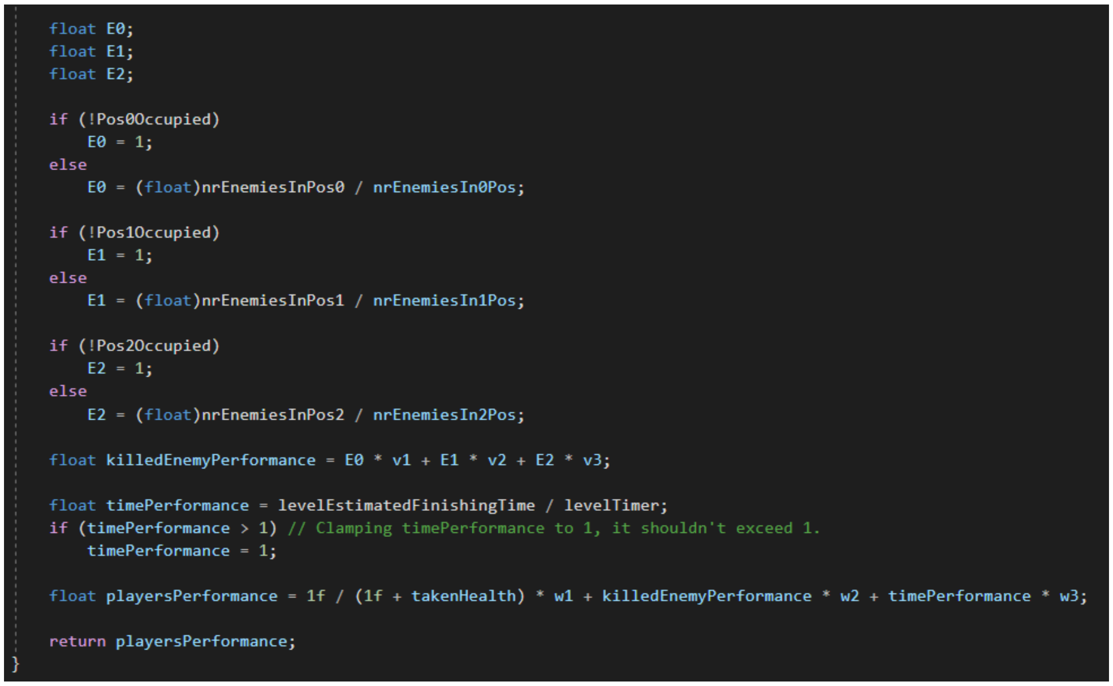
$\boldsymbol{Note}$: *The player gets evaluated only on the things present in the level. If that thing does not exist in the level, he gets for it a performance value of 1, so that it does not affect the overall PP formula.*

Implementation



```
public float CalculatePlayerPerformance(float v1, float v2, float v3, float w1, float w2, float w3)
{
    float vT = v1 + v2 + v3;
    if (vT != 1)
        throw new System.Exception("Vi WEIGHTS ARE NOT EQUAL TO 1");

    float wT = w1 + w2 + w3;
    if (wT != 1)
        throw new System.Exception("Wi WEIGHTS ARE NOT EQUAL TO 1");

    float takenHealth = playerStats.nrDeaths + (1 - playerStats.healthPercentage);

    float nrEnemiesIn0Pos = usedEnemyPositions.Count; // nr enemies in normal Pos
    float nrEnemiesIn1Pos = 0;                         // nr enemies in medium Pos
    float nrEnemiesIn2Pos = 0;                         // nr enemies in hard Pos

    if (usedEnemyPositions.Count != 0)
        Pos0Occupied = true;

    foreach (Transform pos in usedEnemyHardPositions)
    {
        if (pos.GetComponent<Category>().value == 1)
        {
            nrEnemiesIn1Pos++; // here check occupation of medium Pos
            Pos1Occupied = true;
        }

        if (pos.GetComponent<Category>().value == 2)
        {
            nrEnemiesIn2Pos++; // here check occupation of hard Pos
            Pos2Occupied = true;
        }
    }
```

Screenshot 26: *Implementation of player's performance formula in the function "CalculatePlayerPerformance" – Part 1.*



```
    float E0;
    float E1;
    float E2;

    if (!Pos0Occupied)
        E0 = 1;
    else
        E0 = (float)nrEnemiesInPos0 / nrEnemiesIn0Pos;

    if (!Pos1Occupied)
        E1 = 1;
    else
        E1 = (float)nrEnemiesInPos1 / nrEnemiesIn1Pos;

    if (!Pos2Occupied)
        E2 = 1;
    else
        E2 = (float)nrEnemiesInPos2 / nrEnemiesIn2Pos;

    float killedEnemyPerformance = E0 * v1 + E1 * v2 + E2 * v3;

    float timePerformance = levelEstimatedFinishingTime / levelTimer;
    if (timePerformance > 1) // Clamping timePerformance to 1, it shouldn't exceed 1.
        timePerformance = 1;

    float playersPerformance = 1f / (1f + takenHealth) * w1 + killedEnemyPerformance * w2 + timePerformance * w3;

    return playersPerformance;
}
```

Screenshot 27: *Implementation of player's performance formula in the function "CalculatePlayerPerformance" – Part 2.*

The variables "*nrEnemiesInPos0*", "*nrEnemiesInPos1*" *and* "*nrEnemiesInPos2*" are initialized with 0. Each time an enemy gets killed, the corresponding variable to his position gets incremented in a different script attached to him. Thus, we keep on track how many enemies are dead and in what positions' category.

Listed below are the weights of each element of the player's performance formula:

- **w_1 (Damage taken) = 0.35**.
- **w_2 (Killed enemies) = 0.35**.
- **w_3 (Time) = 0.3**.
- **v_1 (Killed enemies in normal positions) = 0.2**:
- **v_2 (Killed enemies in medium positions) = 0.3**:
- **v_3 (Killed enemies in hard positions) = 0.5**: Enemies in hard positions are hard to deal with. That should impact the performance more than killing enemies in medium or normal positions.

3.2.2 Measurement of The Estimated Completion Time

The way I measured the best optimised time to finish a level was by measuring the completion time of all its components separately including tiles, enemies and obstacles. Then, I summed what the level includes together to get the level's estimated time to complete.

For the tiles, I recorded for each one of them the time to traverse from start to end with no enemies included. I did this twice to make sure that I get the most optimal route.

As for the enemies I recorded the time to kill each one of them in an optimal way without taking any damage. Since I developed the game and have a lot of experience with it, I know the most efficient way to kill them as fast as possible without taking any damage.

As for the obstacles, if the player has space between them to land, they mostly have no effect. But in hard levels, the obstacles have mostly no space in between for the player to Jump, so when the player gets hit by them, he slightly gets delayed (approximately 0.2s).

In the tables below are a level's components and their corresponding optimised completion time.

Best Optimised Time to Traverse a Tile:

Tile	Time in seconds	Tile	Time in seconds	Tile	Time in seconds
1	3.9	**5**	3.9	**9**	5.4
2	3.1	**6**	5.7	**10**	5.6
3	4	**7**	5	**11**	5.4
4	4.6	**8**	4.7	**Obstacles**	3.1

Best Optimised Time to Kill Enemies and Overcome Obstacles:

Enemy / Obstacle	Time in seconds
Monster Light	1.5
Monster Range	1
Monster Heavy	2.75
Obstacle	0.2

4. First Survey – Evaluating the Formulas

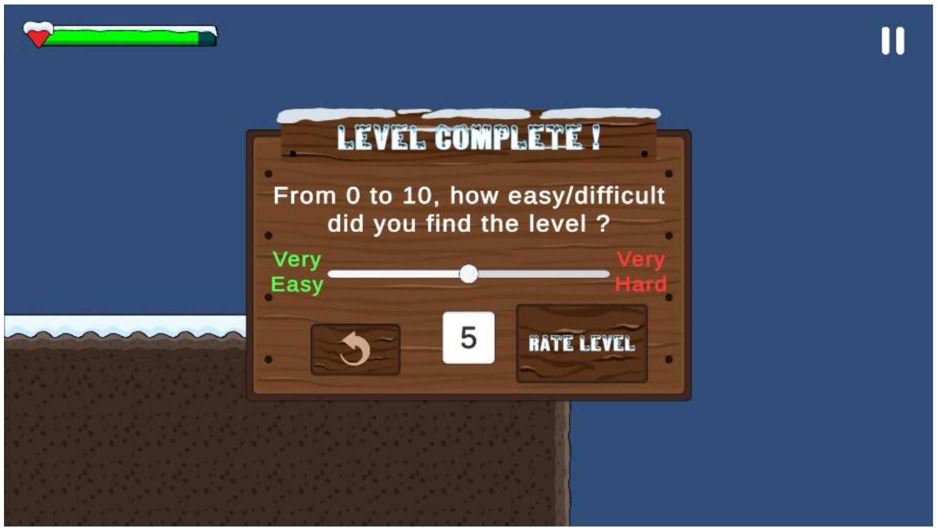
4.1 Goal

The aim of this survey is to find out how accurate the calculated difficulty by the formulas is to the perceived difficulty determined by the play-testers. The perceived difficulty is of course something that cannot be determined precisely and absolutely since it is connected to each player individually. Different players can perceive a level's difficulty differently. But having a lot of people giving their feedback on how they perceived a level's difficulty, can give us an intuitive insight into how precise and helpful the formulas are for determining a level's difficulty.

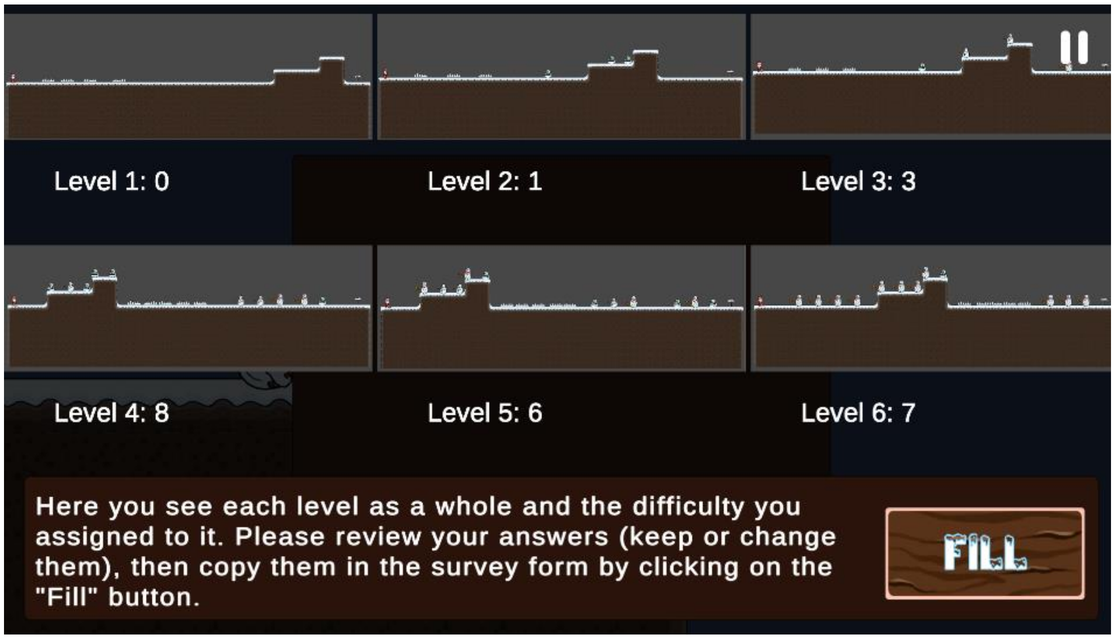
4.2 Set Up

To conduct this survey, as shown in the "Testing Level" section, I have created in Unity six levels with different arrangements of tile-parts for the sake of variety. The six levels had different numbers and positions of enemies and obstacles, which resulted in having different levels with different unique difficulties.

Before the player starts playing the levels, a tutorial on the basics of the game had to be played first, then the player starts with the levels. At the end of each level the player is asked to rate it from 0 to 10 with 0 being "very easy" and 10 being "very hard" as demonstrated in Screenshot 28. The levels' order was random because I wanted the player to rate each level independently having no influence from the other levels; if the order of the levels was linearly based on their difficulties, the player would rate each next level higher than the previous one considering the fact that the difficulty is linear. After finishing all six levels, the player gets shown all the levels as a whole with their corresponding assigned difficulties, then asked to review his answers by keeping them or changing them as shown in Screenshot 29, then at last copy his answers in a google survey form by clicking the "Fill" button.



Screenshot 28: *The "Level Complete" panel with the rating slide from 0 "Very Easy" to 10 "Very Hard", that gets shown to the player after completing a level.*



Here you see each level as a whole and the difficulty you assigned to it. Please review your answers (keep or change them), then copy them in the survey form by clicking on the "Fill" button.

Screenshot 29: *All the six levels shown to the player as a whole with their assigned difficulty and the "Fill" Button.*

The project was uploaded on the website "itch.io" in a zip file as a WebGL project, then the search for participants aka play-testers begun.

This survey took about 10 minutes to finish.

4.3 Data Collection

Since this survey is focused on comparing the perceived difficulty and the calculated difficulty, I targeted people who have prior gaming experience to participate in.

The process of collecting participants was done by various ways, such as word of mouth by asking classmates and friends, posting the survey at the university's forum with the help of my teacher and posting the survey in Facebook and discord groups that are related to game development and/or gaming.

To avoid bias in the results of the survey, I did not explain to the participants that their answers aka perceived difficulty will be compared with a pre-defined difficulty aka calculated difficulty. The participants were given the "itchi.io" link I mentioned above and asked to follow the instructions in the game.

In a window of time of four days, I managed to have 28 participants in total. The participants' age ranged from 20 to 30 years old.

With the help of google forms, I had the 28 responses organized in charts, which I used to analyse the data as I will discuss in the next section.

4.4 Results and Analysis

4.4.1 Collected Data (Perceived Difficulty)

Before starting to analyse the findings of the survey I want to show the survey's results of each level presented in the bar graphs from google forms. Each graph has an X axis that represents the difficulty selected by the play-testers aka perceived difficulty, and a Y axis that represents the number of responses for each selected difficulty.

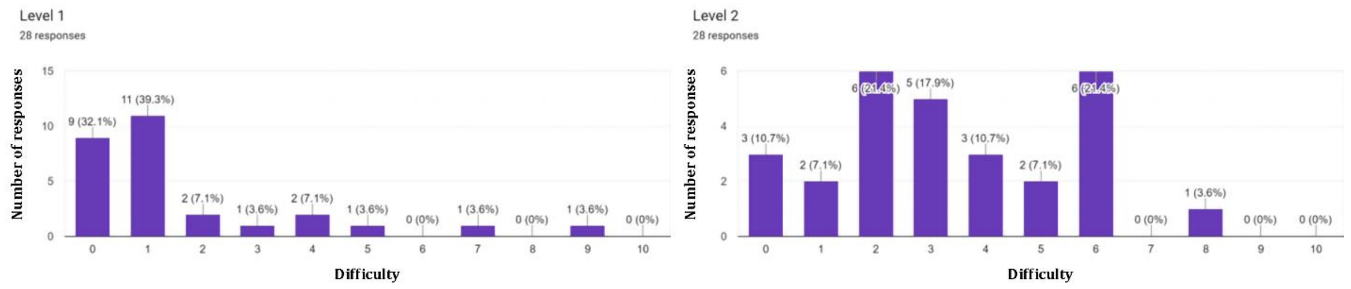


***Figure 1**: First survey's results of the level 1 and 2 as graph.*

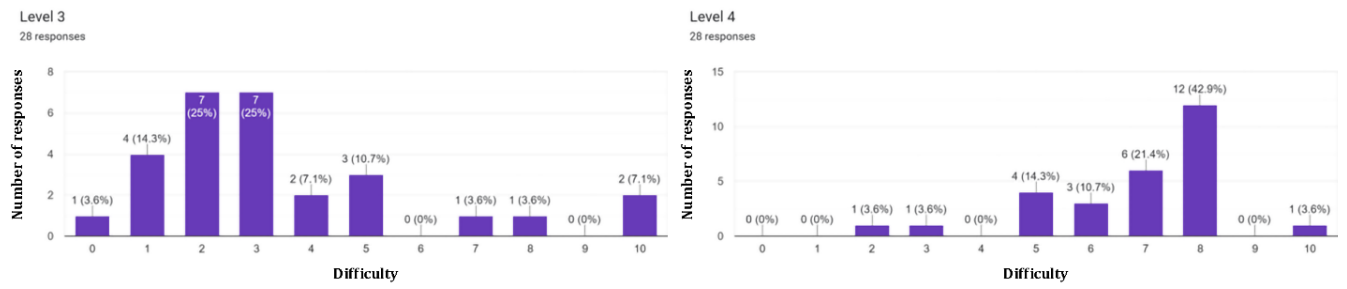


***Figure 2:** First survey's results of the level 3 and 4 as graphs.*

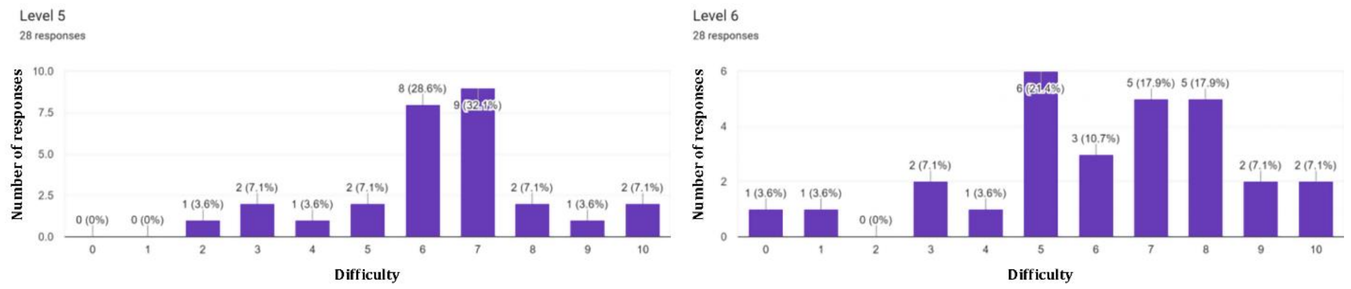


***Figure 3**: First survey's results of the level 5 and 6 as graphs.*

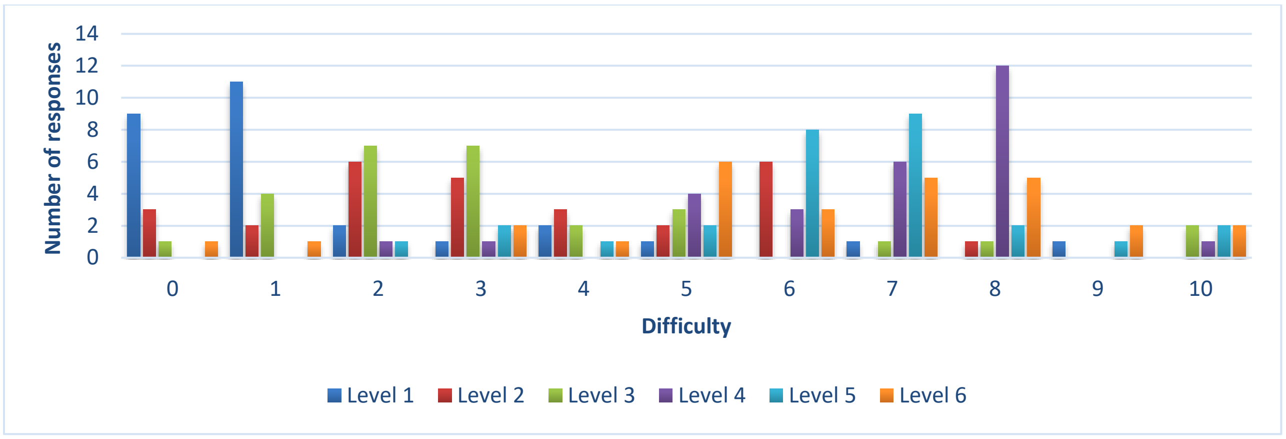


Figure 4: *A summary of the difficulty responses for each level, in which we see lighter colours more on the left of the diagram representing the first levels, and more darker colours on the right representing the last levels.*

Looking at the survey's results in Figure 1, 2 and 3, we can clearly see that in almost each graph, the number of responses per difficulty are - respectively to the difficulty axis - cognate, which means that the play-testers had similar but slightly different perceived difficulty to the levels. The only two exceptions are the level 2 and 6 due to reasons are discussed in the next sections.

As I mentioned before, the levels were in a fixed random order for all the players that goes as follow: Tutorial -> Level 2 -> Level 4 -> Level 1 -> Level 6 -> Level 5 -> Level 3. So, after playing the tutorial, the player starts with the level 2, in which he encounters enemies and obstacles for the first time and rate it first. Since the level 2 is the player's first experience of the game, it is hard for him to rate its difficulty without prior knowledge of the game (has not adapted to the game yet). Thus, it is normal to get different responses about the level's difficulty that are way different from each other, because each player's first impression of the level 2 (first level) is unique.

4.4.2 Perceived Difficulty Vs Calculated Difficulty

So far, I have just discussed the perceived difficulty's data given by the play-testers only. Now I compare it to the calculated difficulty data, which was determined by the mathematical formulas.

To compare the two data, I calculated the average perceived difficulty of each level from the responses in the graphs, to get one generalized perceived difficulty value for each level. Then I compare that value to the calculated difficulty value of the same level. The results are put in a graph as shown in figure 5.

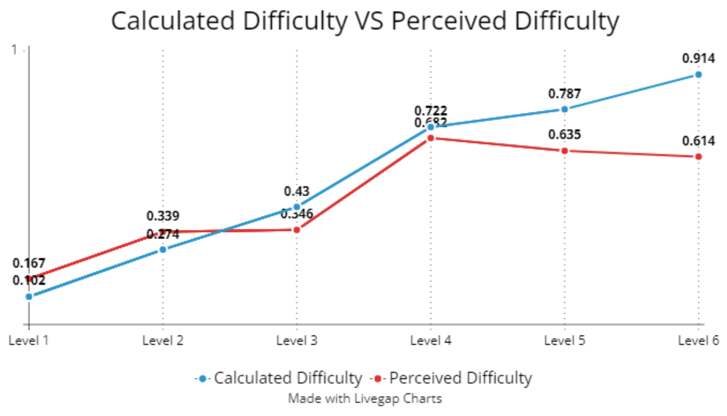
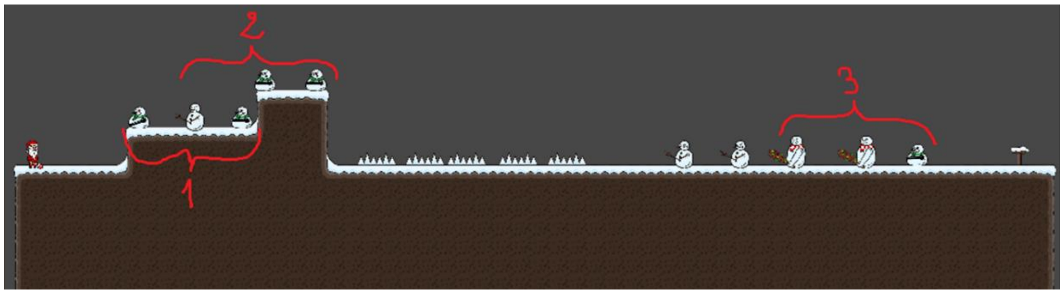


Figure 5: *The perceived difficulty diverging from the calculated difficulty starting from the level 5.*

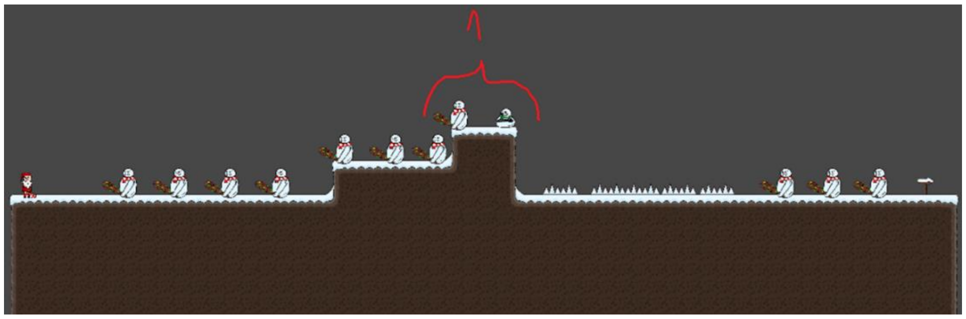
In figure 5 we have two curves; the red one represents the perceived difficulty throughout the levels, while the blue one represents the calculated difficulty by the formulas. As we can see from the graph, I intended for the difficulty in the six levels to be increasing with each level. And the survey was conducted to see if the levels' perceived difficulties from the play-testers match the calculated ones.

According to figure 5, the perceived difficulty and calculated difficulty from the level 1 to the level 4 are almost identical, which proves that the mathematical formulas could predict the perceived difficulty to some extent. Even the level 2 who had a random and scattered responses as shown in Figure 1, its average perceived difficulty was very close to the calculated one. Yet starting from the level 5, we see that the two curves start to diverge; the calculated difficulty was increasing as I intended it to be, but the perceived one kept decreasing slightly.

In order to explain the divergence in Figure 5, we have to look at what the mathematical formulas take in consideration to calculate the difficulty and analyse the difference between the fourth and sixth level. The level 4 – according to Figure 5 - was the hardest level for the participants, while the level 6 should have been the hardest. The mathematical formulas take into consideration enemies' AI, number, positions in the map, HP, the obstacles' number, and the distances between obstacles. Some of the participants who took the survey pointed out that having only big enemies with high HP and damage did not necessarily increase the difficulty of the last levels, aka level 5 and 6, but rather the combination of range and melee enemies what made a level hard. Dealing with two enemies at the same time – in my game's case dealing with the bullets of a range enemy and the attacks of a melee enemy at the same time – is of course harder than dealing with only one of them separately. Given the level design of the fourth and sixth level, as shown in Screenshots 29 and 30, we can see that not only the combination of range and melee monsters - which are used more in the fourth level - made the level harder, but also the cluster of range enemies at the beginning of the level 4. The players had to deal with multiple bullets at once.



Screenshot 30: *The level 4 with three combinations of range and melee monsters pointed at with red (1,2,3), which affects the perceived difficulty significantly.*



Screenshot 31: *The level 6 with one combination of range and melee monsters pointed at with red (1), which does not affect the perceived difficulty that much.*

Since the combination of enemies is the missing part, we should first understand what "*enemies' combination*" in the context of the latter means and where I can implement it.

In the formula $\boldsymbol{F3}$ about the enemies' positions, the positions' labelling from easy to hard is related to the level's map. For example, some positions are at the edge of a cliff, so positioning an enemy there is more challenging for the player to deal with than an enemy in a flat tile, where the player has more room to move, thus the position at the edge of the cliff is regarded as difficult compared to the flat one. The thing we understand from "enemies' combination" is basically the positions of the enemies in respect to each other, which is something I did not consider. So, the solution would be to label positions from easy to hard based not only on the level's map, but also on other enemies' positions. For instance: An enemy "A" can be placed in an "easy" labelled position, but if another enemy occupies a nearby position, the label of the enemy A's position should be changed or stay the same based on the type of the new enemy that occupies that near position.

4.4.3 Fixing the Implementation of $\boldsymbol{F3}$

The enemies' combination that increases the difficulty in my game is putting a melee monster in front of a range monster as I mentioned before. So, the way I altered my code to consider that, was through an algorithm that is simplified as follows:

- Store in an array all the range enemies in the level.
- Go through each range enemy in that array and check if the position in front of him is occupied.
- If that position is occupied, that position's label becomes "hard" by giving it a value of 2.

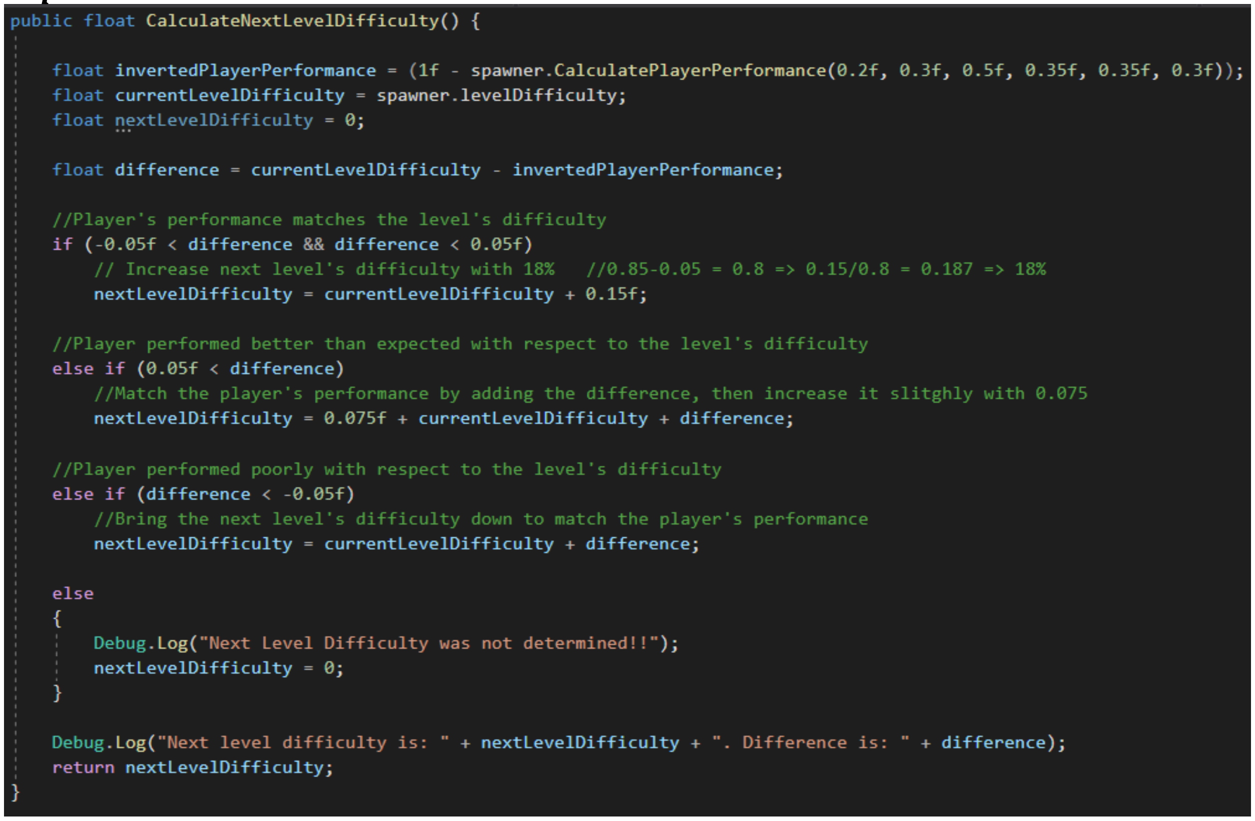
Now that the implementation of $\boldsymbol{F3}$ is fixed, the overall formula $\boldsymbol{F}$ is now more precise at determining a level's difficulty. With that being done, we can move to the implementation of dynamic difficulty adjustment using the formulas $\boldsymbol{Fi}$.

5. Implementation of Dynamic Difficulty Adjustment

Now that a level's difficulty can be evaluated through the formulas, a DDA algorithm is implemented to determine the next level's difficulty based on the calculated difficulty of the previous level and the player's performance in it. After that, the next level gets generated based on that.

The generation of the next level that matches a given difficulty is done by altering some parameters of the functions that spawn the level's components individually based on the given difficulty value before generating the level, and by additional functions that change the content of the level after its generation. This results in a level's difficulty that is approximate to the given difficulty.

Implementation



```
public float CalculateNextLevelDifficulty() {

    float invertedPlayerPerformance = (1f - spawner.CalculatePlayerPerformance(0.2f, 0.3f, 0.5f, 0.35f, 0.35f, 0.3f));
    float currentLevelDifficulty = spawner.levelDifficulty;
    float nextLevelDifficulty = 0;

    float difference = currentLevelDifficulty - invertedPlayerPerformance;

    //Player's performance matches the level's difficulty
    if (-0.05f < difference && difference < 0.05f)
        // Increase next level's difficulty with 18%   //0.85-0.05 = 0.8 => 0.15/0.8 = 0.187 => 18%
        nextLevelDifficulty = currentLevelDifficulty + 0.15f;

    //Player performed better than expected with respect to the level's difficulty
    else if (0.05f < difference)
        //Match the player's performance by adding the difference, then increase it slitghly with 0.075
        nextLevelDifficulty = 0.075f + currentLevelDifficulty + difference;

    //Player performed poorly with respect to the level's difficulty
    else if (difference < -0.05f)
        //Bring the next level's difficulty down to match the player's performance
        nextLevelDifficulty = currentLevelDifficulty + difference;

    else
    {
        Debug.Log("Next Level Difficulty was not determined!!");
        nextLevelDifficulty = 0;
    }

    Debug.Log("Next level difficulty is: " + nextLevelDifficulty + ". Difference is: " + difference);
    return nextLevelDifficulty;
}
```

Screenshot 32: *Implementation of the DDA algorithm in the function "CalculateNextLevelDifficulty".*

On one hand, a level's difficulty is a decimal number from 0 to 1; with 0 being the easiest and 1 the hardest a level can ever be. On the other hand, the player's performance is also a decimal number from 0 to 1; with 0 being the worst performance and 1 the best. **Naturally, the easier a level is, the higher the performance of the player is expected to be**; For instance, if a level has a difficulty of 0.1, the player should get a performance around 0.9. To be able to read the relationship between those two variables, I invert the player's performance so that the difference between the player's performance and the level's difficulty is very clear. Since the two variables can never be identical, I set a **tolerance variable of 0.05** to make the comparison easier. This leaves us with three cases to draw from the difference as seen in the DDA algorithm in Screenshot 32:

- **Difference is very small: Between -0.05 and 0.05:** In this case, the two variables are almost identical, which indicates that the player's performance matches the level's difficulty. So, the difficulty increases with **18% (*this variable is a design choice)*** by adding **0.15** to the current level's difficulty to have the next one. If the player keeps falling in this case, he'll have a linear difficulty in the game.
- **Difference is bigger than 0.05:** This means that the player performed very well respectively to the level's difficulty. The difficulty gets increased by adding the difference, to match the player's performance and additionally the value **0.075 (9%)** to slightly further the difficulty of the next level above his performance/skills.
- **Difference is smaller than -0.05:** This means that the player performed poorly. The difficulty gets decreased by adding the difference (*difference is a negative value in this case*) to bring it down to match his performance/skills.

6. Second Survey – Dynamic Difficulty Adjustment

6.1 Goal

The aim of this survey is to compare the experience of players playing a game with a DDA system based on the formulas $\boldsymbol{Fi}$, with an experience of playing a game with a pre-defined difficulty curve. The results should give us an insight into the usefulness of these formulas to improve the player's experience through DDA.

6.2 Set Up

To conduct this survey, I created two playthrough versions:

- **DDA version (D-Version):** With dynamic difficulty adjustment implemented. In this mode, the player starts with a level with a low random difficulty between **0.05** and **0.1**. Then based on his performance, the next level's difficulty gets determined. The end of the playthrough is determined by two conditions:
    - If the player reaches a level with a difficulty of **0.82** or more, that level is considered the last level and the game is considered finished.
    - If the player plays eight levels in total, the eighth level is considered the last level and the game is considered finished. This condition is for players who perform bad and cannot reach a level with a difficulty of 0.82 or more.
- **Normal version (N-Version):** With a normal difficulty curve. In this mode, the player plays exactly five levels with pre-determined difficulty that increases with each level. The levels' difficulties are as follows: **Level 1 (0.12) -> Level (0.31) -> Level 3 (0.53) -> Level 4 (0.71) -> Level 5 (0.84)**. The tester finishes this playthrough by completing the fifth level.

In each version, before playing the first level, the testers play a very short tutorial level that teaches the basics of the game.

In order to measure the testers' experience, they got asked different questions throughout their playing:

- **After completing a level:** They got asked about its difficulty, fairness and enjoyment.
- **After completing a playthrough (a version):** They got asked if the game's version's challenges met their skills, whether they felt frustrated or motivated and whether they felt bored or excited from a scale of five.
- **After completing both playthroughs:** They get asked to compare both experiences (versions) in terms of enjoyment, challenge, boredom and frustration.

Beside asking the testers their opinions, not only the levels' difficulty in the D-Version gets recorded for each player, but also their performance in both versions. This is very important, because it is sometimes difficult for the testers to put their true experience in words, so it is easier for me as an evaluator to look at numbers while at the same time keep their opinions in consideration.

Since the game is new to all testers, a bias in comparing the two versions N and D can emerge from the fact that a tester who plays one version first might find it hard compared to the other version, since it would be his first experience with the game. That would make the other version – which he would play second – slightly easier, because he would have already been familiar with the game and that would influence his answers. To avoid this problem, I divided the testers into two groups, a group that starts with the D-version and a group that starts with the N-version.

Each version was uploaded on the website "*itch.io*" in a zip file as a WebGL project.

Whenever a tester must answer a survey's question, he switches to a google form page, where the survey's questions are saved.

This survey took about 20 to 25 minutes to finish.

6.3 Data Collection

Since this survey is focused on the player's experience, I tried to have people to participate, who have different gaming experiences. Some had zero gaming experience while some were daily gamers. The number of weekly gaming hours of the participants ranged from 0 to 40.

The process of collecting participants was done by asking friends and students in the university's cafeteria to participate for an exchange for a small reward (candy and cookies). In a window of a week and a half, I managed to have 25 participants in total. The participants' age ranged from 17 to 32 years with the most dominant age range being 21-25.

To avoid bias in the results of the survey, the participants were not informed which playthrough version were they playing and did not know what the levels' difficulties were. Some participants knew that the bachelor thesis was about dynamic difficulty adjustment, but not in a detailed way.

With the help of google forms, I had the 25 responses organized in charts, which I used to analyse the data, as I will discuss in the next section.

In the following section, I categorized the participants into 3 groups:

- **Beginners**: people who play video games 0 to 5 hours per week.
- **Intermediates**: people who video games 7 to 10 hours per week.
- **Experienced**: people who video games 12 to 40 hours per week.

6.4 Results and Analysis

6.4.1 Completion of Extra Levels (6, 7 & 8) in D-Version

As I mentioned before, in the N-Version a player plays only 5 levels, but in the D-Version, depending on his performance he can play from 4 (if he is really skilful and very familiar with the game; only I the developer could beat the game in the 4th level) to 8 in total. So, a very good player should finish the game in the 5th level, making his experience similar to the N-Version's experience.

In the survey, it was interesting to see how many extra levels (6, 7 and 8) the players had to play to finish the D-Version. Some of them reached the 8th level and still did not reach the difficulty 0.82.

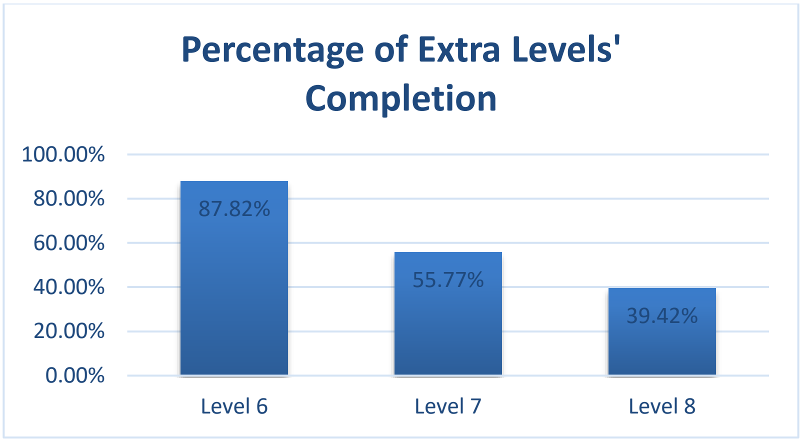


Figure 6: *Percentage of players who played the extra levels in D-Version.*

In Figure 6, we see that 87.82% of participants had to finish the level 6 to proceed, which means that only 12.18% finished the D-Version in 5th level. This makes the latter have a similar experience to the N-Version, since it also requires 5 levels to beat.

As we can see in Figure 6, the percentage of completion of the extra levels keeps decreasing per level. This means that the dynamic difficulty adjustment in the D-Version adjusts the game's difficulty for players according to their skill levels, this is why some participants had to play more levels than others. More on this can be seen in Figure 7 below in detailed with groups.

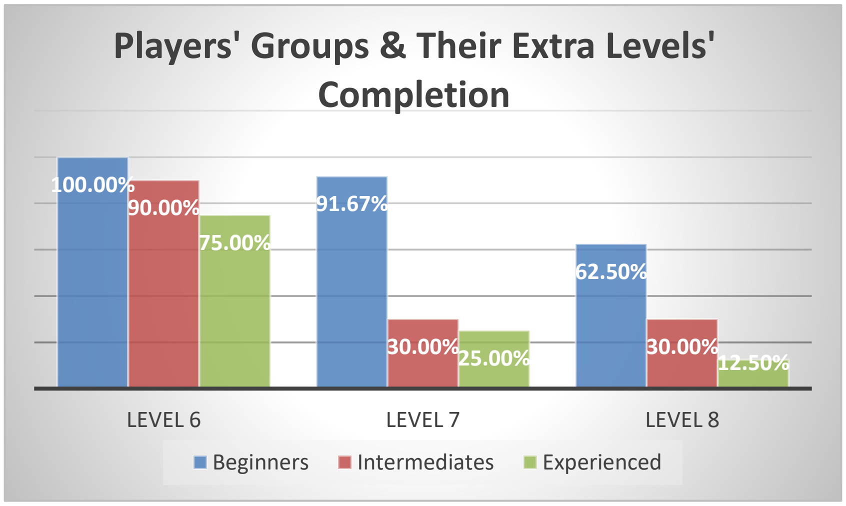


Figure 7: *Players' groups and their completion of an extra level.*

From Figure 7, we see that on one hand, all the beginners' completion percentage is higher than all other groups, especially in the level 7 and 8. On the other hand, the experienced group's completion percentage declined drastically. This indicates that the beginners needed more playtime with the game to adjust and be good at it while the experienced group got accustomed to it quickly and therefore finished the D-Version earlier. The DDA implemented in the D-Version offered each group (or even a player) a unique experience to finish the game according to their skills and performance.

6.4.2 Perceived Difficulty

In this section, I analyse and compare the perceived difficulty of the levels in both versions N and D.

In the survey, participants were asked to rate each level's difficulty from **very easy (0) to very hard (5)**.

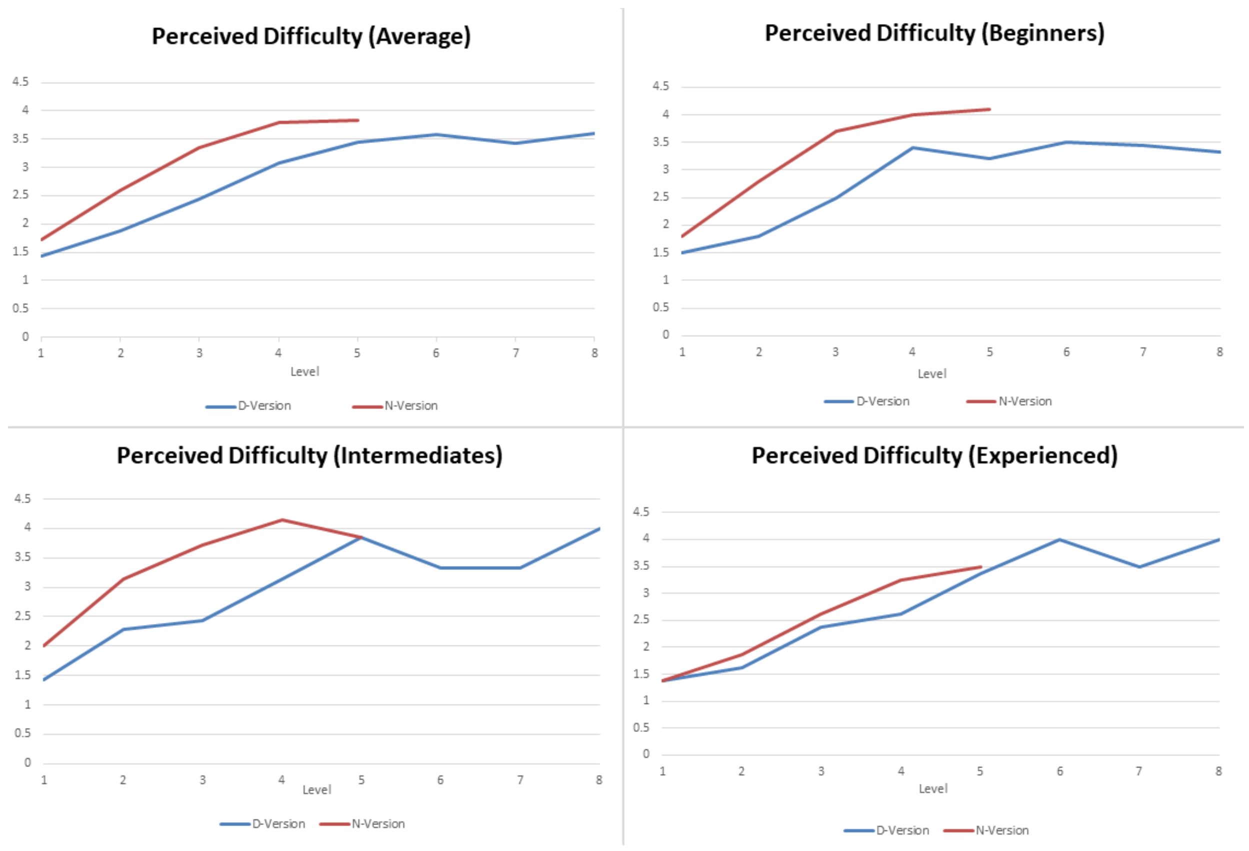


Figure 8: *The perceived difficulty in D-Version VS N-Version in average, for beginners, intermediates and experienced (from top left to bottom right).*

In Figure 8, we see the perceived difficulty (Y-Axis) of each level (X-Axis) in D-Version and N-Version in average and for each group. From the graph, we can conclude that the participants found the levels of the N-Version quite harder than the D-Version, especially the beginners. This is due to the difficulty curve of the N-Version that keeps raising the difficulty with each level no matter how bad or good the player performs. The N-Version's challenges expect from the player to be good enough to meet them, which was not the best case for the beginners. As for the experienced group, they had an almost matching perceived difficulty of both version D and N with a slight difference, except in level 4.

6.4.3 Levels' Recorded Difficulty and Players' Performance

In this section, I analyse and compare the recorded difficulty - calculated by the formulas - of the levels and the players' performance in both versions N and D.

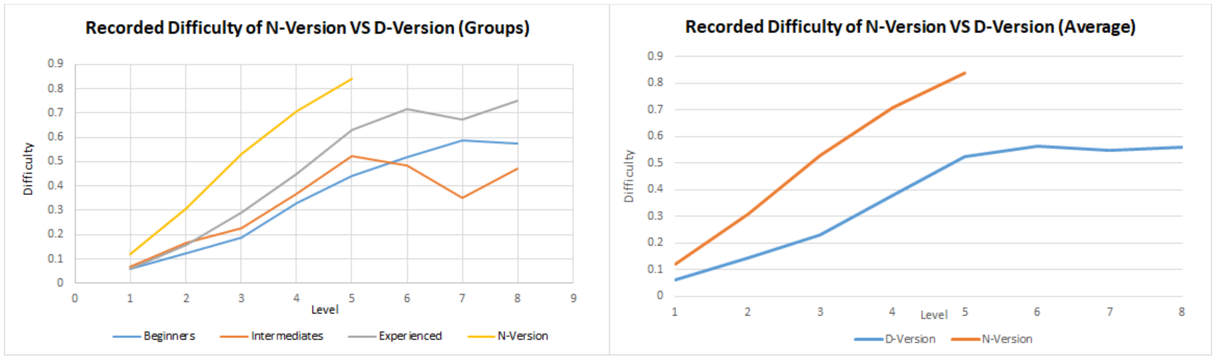


Figure 9: *Recorded difficulty of the levels calculated by the formulas of the N-Version VS D-Version in general (Right graph) and in groups (Left graph).*

Since the N-Version has a linear predefined difficulty curve, its levels' difficulties are the same in each walkthrough. So, as seen in Figure 9, the N-Version's difficulty curve is always the same. It is coloured in yellow on left graph and in orange on the right graph. As for the D-Version, each player's walkthrough is a unique experience; each level he plays has a different difficulty from the other players' levels, except for the first level; some first levels might have similar difficulty since their difficulty is not based on the player's performance of a previous level. I recorded those levels' difficulty and now we can see them in Figure 9 in average on the right graph in blue and for each participants' group in average on the left graph.

On the right graph in Figure 9, it is clear to see a big difference between the recorded difficulty of the N-Version and the D-Version. This is due to the DDA system implemented in the D-Version that determines the next level's difficulty based on the player's performance. Since the players are not familiar with the game, they did not perform well enough to match the N-Version's difficulty curve, which demands from the player to have very high skills to perform perfectly, especially in 2D platformers.

On the left graph, we see in detail the average performance of each group, which of course is below the N-Version. But what I want to focus on in this graph, is the difference between the groups. The difference in their recorded difficulty in the D-Version is at the first levels small, but it gets bigger with each level. Also, the positioning of the groups' graphs match their skill levels; we have the beginners being the lowest, the intermediates in the middle and the experienced being the highest. Realistically, this should be the case, because the higher skills a player has, the better he performs, and the higher the levels' difficulties are. This indicates that the DDA system did a good job in adjusting the difficulty for each group accordingly.

The only exception we see on the left graph is the two drops of the experienced and the intermediate groups' graphs. This could be simply explained by the fact that after the players performed well in a level (5 (*only for intermediates*) or 6 for example), they got a next level with a higher difficulty than the previous one, in which they could not keep up a similar performance to the previous one. But in both groups, in the level 8, we see that their graphs rose again, which means that the players adapted. This change of difficulty in both cases indicates that the DDA system adapted the game's difficulty to the players' performance as intended.

Note: *The average recorded difficulty and performance (coming next) of the extra levels in the graphs, consider only the played ones. So, if a level 8 is played only by 2 players, the average is of those 2 players only.*

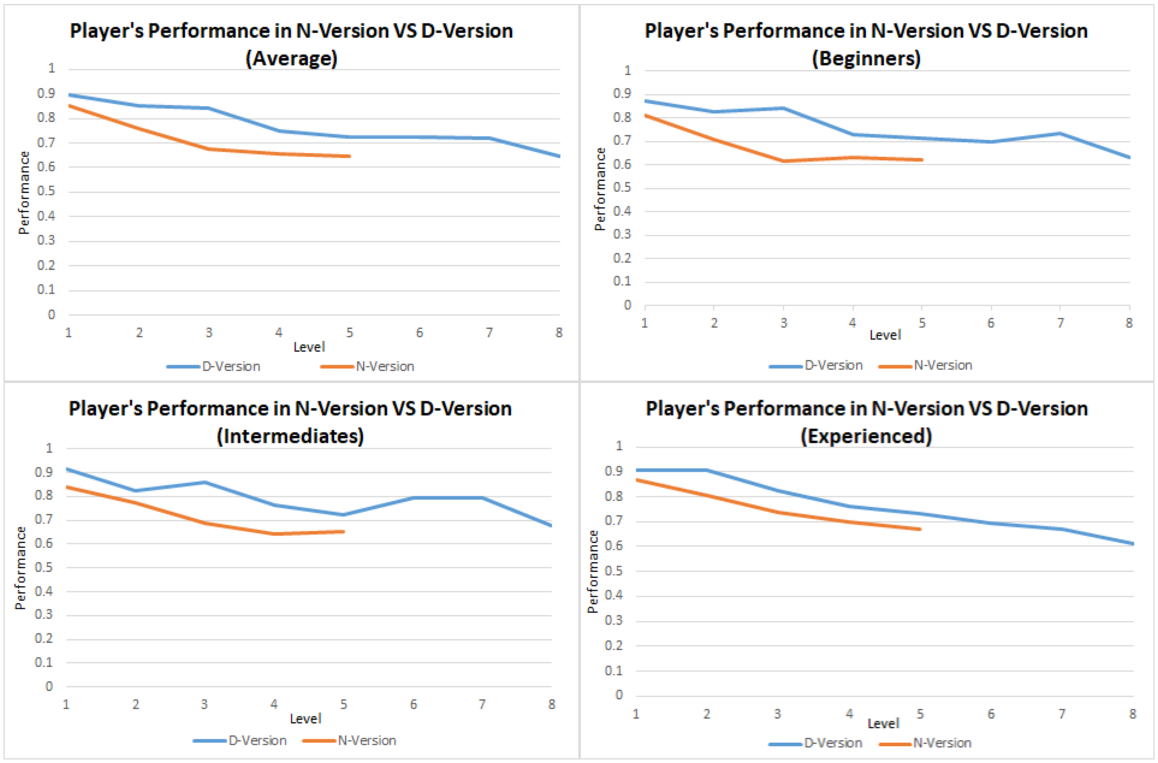


Figure 10: *The players' performance in D-Version VS N-Version in average, for beginners, intermediates and experienced (from top left to bottom right).*

In Figure 10, we clearly see a difference in the players' performance between the versions N and D. The participants performed relatively worse in the N-Version than in the D-Version. Looking at Figure 9, we can expect these performance results. Since the levels' difficulties in the N-Version are higher than in the D-Version, the performance in the N-Version would be worse than in the D-Version.

The interesting thing in these graphs is the difference between the two versions in each group and its constancy. For the beginners and intermediates, they had ups and downs in their performance in the D-Version, but in the N-Version, their performance kept decreasing until it kind of reached a plateau around the value **0.62** in level 4 and 5 for intermediates and around the value **0.64** in levels 3, 4 and 5 for beginners. The ups and downs in the D-Version could be interpreted as the adjustments of both players and the DDA system to one another. The plateau can be explained by the fact that it is quite hard to get a worse performance value below 0.6, since I asked the participants that it is a must to kill all enemies, which in itself raises the performance a lot.

As for the experienced group, their performance in the D-Version kept decreasing from level 2 and from the start in the N-Version. But interestingly, the difference between both versions in the experienced group is overall quite narrower and more consistent than the beginners and intermediate groups. Also, the worst performance they got in the N-Version is **0.67**, which makes them have the best performance compared to the other groups.

Another thing to mention is the worst performance reached in the D-Version. Interestingly, it is the experienced group who has the lowest performance at level 8, which makes sense. If we look at the left graph in Figure 9, we see that the average 8th level's difficulty that the experienced group played was way higher than the other groups. Which emphasis the idea that each group had a different experience thanks to the DDA system.

6.4.4 D-Version VS N-Version

After playing through each version, the players were asked to rate from 1 to 5 how they felt about their walkthrough using the following statements:

- I felt that the game's challenges met my skills. (1: Strongly disagree & 5: Strongly agree).
- After playing some levels, I felt _____. (1: Frustrated & 5: Motivated).
- After playing some levels, I felt _____. (1: Bored & 5: Excited).

We can see the results in Figure 11 below.

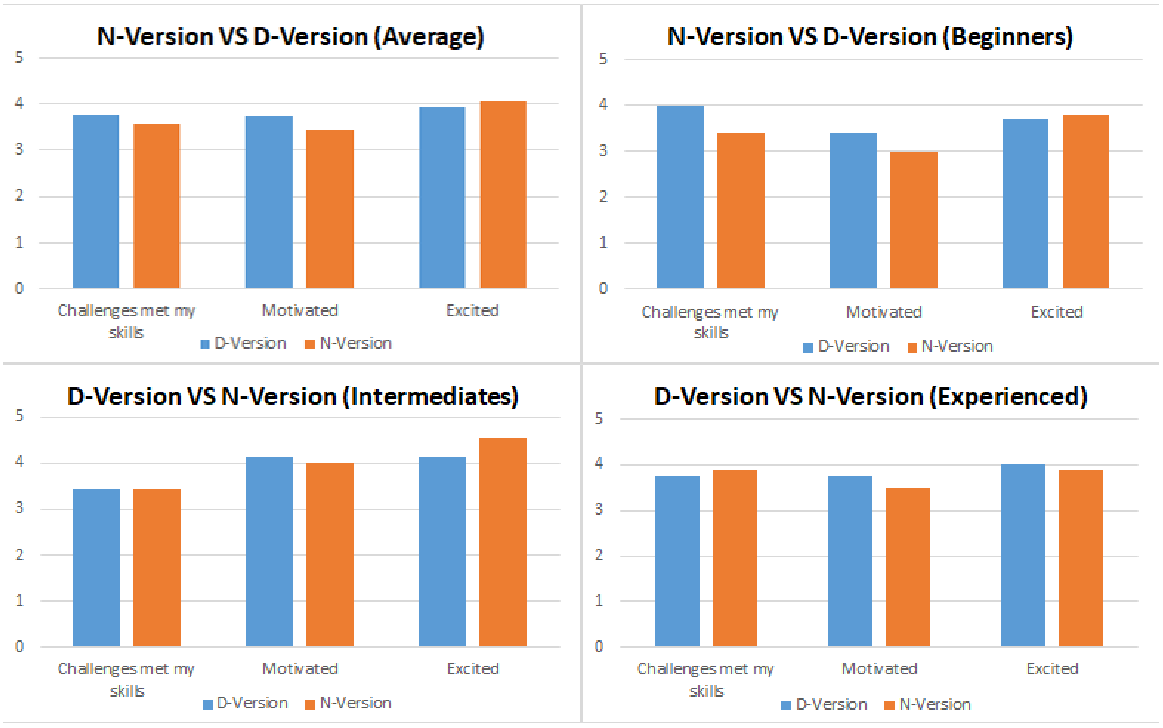


Figure 11: *The players' performance in D-Version VS N-Version in average, for beginners, intermediates and experienced (from top left to bottom right).*

On the top left graph in Figure 11, we see that there is not a big difference between both versions on average. The players found the D-Version's challenges slightly met their skills more than the N-version's. Also on one hand, they felt slightly more motivated in the D-Version, but on the other hand they felt slightly more excited in the N-Version.

Some significant differences can be seen when we look at the rating of each group separately. For the beginners, they found the D-Version met their skills more than the N-Version, which makes sense, since the N-Version requires more skills and the D-Version adapted to theirs. For the same reason, they were somewhat more motivated to continue playing in the D-Version than in the N-Version. Interestingly, they found the N-Version somewhat slightly more exciting than the D-Version. I think that is because the N-Version is more challenging. But overall, not a big difference in terms of excitement.

As for the intermediates, they had a similar experience in terms of the two versions matching their skills with the lowest rate of **3.42,** which could be interpreted to "Somewhat agree". In terms of motivation, they rated both experiences almost the same, with the D-Version being slightly above. However, they rated the N-Version higher than the D-Version in terms of excitement which goes back to the same reason of the N-Version being more challenging.

For the experienced group, it was interesting to see that there is no significant difference between the two versions in terms of all three aspects. They felt more motivated in the D-Version than in the N-Version and that the N-Version slightly met their skills more than the D-Version, which is the opposite of the ratings of the other groups. This might be because the D-Version increases the difficulty significantly only when the players get an almost perfect score, which led in giving the experienced players not very hard levels compared to the N-Version, and since they are gamers with long weekly gaming hours, they might have had expected difficult challenges that truly match their skills regardless of their performance (like the N-Version did). Unlike the D-Version that kind of take the player's hands and adjusts itself to his performance.

After finishing both versions, the players were asked to compare them in terms of challenge, enjoyment, boredom, and frustration. This task was not easy for the participants, because they did not feel any significant difference between the 2 versions. They found both experiences quite similar, but for the research's sake, I told them that they had to choose between the two versions in terms of the four aspects mentioned before. The results can be seen in Figure 12.

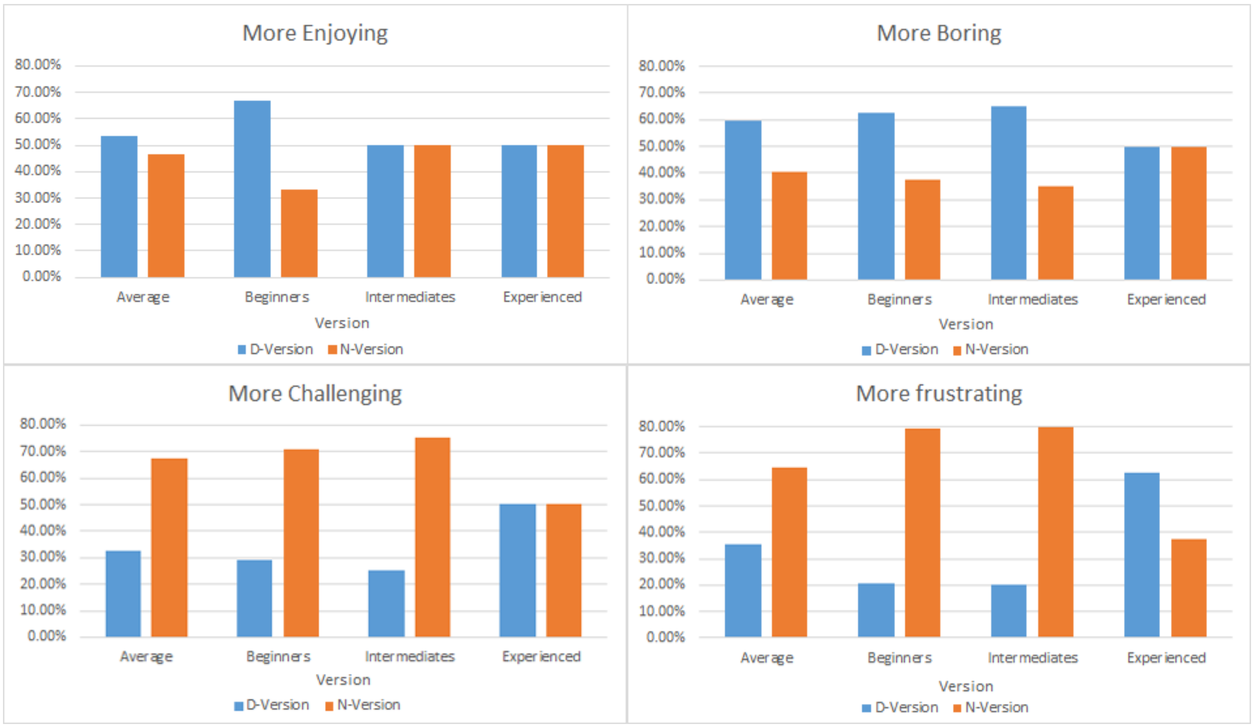


***Figure 12:** D-Version VS N-Version in terms of Enjoyment (top left), Boredom (top right), Challenge (bottom left) and Frustration (bottom right).*

On average, we can see from Figure 12 that the participants found the D-Version slightly enjoyable than the N-Version, but boring. Regarding the frustration and the challenge aspect, the N-Version was found to be more challenging and frustrating because it has a fix difficulty that requires good skills.

For the beginners, the D-Version was more enjoyable (top left graph). However, they rated it as more boring than the N-Version (top right graph). I think that is because the N-Version was more challenging and at the same time frustrating for them than the D-Version. I think this is related to the idea that the less challenging something is, the more boring it is. This leaves us however with a contradiction; how could they rate the D-Version to be the most enjoyable and the most boring at the same time. Firstly, **I want to remind that the participants found both versions almost alike, so they had to pick one**

version. So, the results in Figure 12 - I would say - are not 100% decisive on which version was better. Secondly, when the beginners rated the enjoyment, they might have considered their confidence in being good at the game, so they chose the D-Version more because it offered a decent challenge that met their skills. But when it came to rating boredom, they were - so to say - more genuine because of their human's nature of "Negativity bias" - which is: "*a tendency to give more importance to negative experiences than to positive or neutral experiences"* [Frothingham19]. Basically, the D-Version was rated more enjoyable because it helped beginners with their confidence in performing good, but rated boring, because it was not that challenging compared to the N-Version.

In order to figure out which group preferred which version; I came up with a simple mathematical formula that calculates the preference percentage of a version over the other based on the data on the graphs in Figure 12. The formula is based on knowing the difference between positive emotions (*Enjoyment*) and negative emotions (*Boredom and Frustration*). I did not include the challenge aspect because it is hard to interpret it as something good or bad in a formula. A very hard challenge can be frustrating or exciting, and a very easy challenge can be boring or comforting.

Preference Percentage (%) = Enjoyment (%) – (0.5 * Boredom (%) + 0.5 * Frustration (%)).

***Note:** A result of one version is always the negative of the other version's result. So, calculating for the D-Version is enough.*

The D-Version had a preference percentage of **25%** in the beginners' group, because the enjoyment (66.67%) percentage is higher than the boredom's (62.50%) and frustration's (20.83%).

As for the Intermediates' group, the D-Version was preferable by **7.5%** than the N-Version, which is not a high value. Regarding the enjoyment, the intermediates found both versions equally enjoyable with a rating of 50% for each. As for the boredom, they found the D-Version more boring and the N-Version way frustrating. The intermediates gave a quite similar rating compared to the beginners, except for the enjoyment. The equal rating of the enjoyment in both versions indicates that the enjoyment of the intermediate players in both versions was almost the same.

For the experienced group, the results are interesting, because they found both versions equally similar not only in terms of enjoyment but also in terms of boredom. But regarding the frustration, they found the D-Version more frustrating than the N-Version, which is the opposite of the other groups. This could be because some players had to play more levels in the D-Version than in the N-Version, which took longer and was tiring for some of them. For the experienced, the N-Version was preferable by **6.25%** than the D-Version.

On average the D-Version was **5.7%** preferable than the N-Version, which is not a significant difference.

I would like to quote one of the participants' comment on his experience with both versions that I think helps summarize the results in one paragraph: "*Game version D is definitely more into binding the user to the game. More interesting game design, not too hard levels in the beginning - pretty beginner friendly ;). Game version N is probably more fun for people who are gaming on a regular basis. So I'd say there is no better version (Although version D feels more appealing in my eyes). It is depending on what kind of user u wanna attract*" – A participant.

7 Conclusion

The aim of this study was to provide a DDA system based on mathematical formulas that calculates the difficulty of the most common gameplay challenges in adventure and action games, and test how efficient it is in giving players with different gaming skills a good gaming experience.

7.1 Analysis' Conclusion

Looking at the analysis above, we can conclude that overall, the players' experiences in both versions D and N were almost the same with minor differences. Only the beginners group showed a bit more interest in the D-Version than the N-Version. The similarity of both versions' experiences indicates that the DDA system implemented in the D-Version kept the players engaged enough not to notice a big difference between it and the N-Version. Because when we look at the levels' difficulties in Figure 9 of each version, on one hand, the data clearly shows a big difference between the two, but on the other hand, the players – according to their rated experiences in the "6.4.4 D-Version VS N-Version" section - did not notice a significant difference.

- **For the beginners**, the D-Version was better because it provided challenges that met their skills. So, we can conclude that the DDA system was a success in their case.
- **As for the intermediate**, both versions provided almost the same experience. One of the participants told me that the difficulty curve in the N-Version was well done, and it could suit any player, because the increase of difficulty from level to level in it was smooth and gradual. So, - as he expressed - for an intermediate to experienced player, the difference would not be significant. Since this group found the D-Version almost similar to the N-Version and the N-Version had a good difficulty curve, we can conclude that the DDA version was good enough to provide challenges in the D-Version that it was perceived near similar to the N-Version.
- **As for the experienced** group, they are quite like the intermediate group, it is just that they found the D-Version more frustrating due to its length in levels, as demonstrated in Figure 12. From this we can conclude that the DDA system was successful enough in providing a decent challenge for them, but not a very satisfying one. This could easily be fixed by tweaking the increasing difficulty percentages that are based on the player's performance (look Screenshot 32) to give this group a more challenging and satisfying experience in the game.

7.2 Reflections

As we know, the DDA topic is a vast topic to research. Some researchers specified their approaches for one genre, while some tried to encompass it for all genres. As for my approach which focuses on action and adventure gameplay challenges, I can say - from the results of the survey- that the mathematical formulas were not only useful on evaluating the difficulty of a level, but also on providing a DDA system that gave all participants a decent gaming experience regardless of their gaming skills. Of course, the DDA system is not perfect, but I think it was a success as a first study.

One of the most important things about the approach in this study, is that the DDA system is a collection of mathematical formulas. This makes it very flexible to adjust and expand corresponding to a game's mechanics:

- As for expandability, new mathematical formulas that tackle other or new gameplay challenges can be added to the overall formula $\boldsymbol{F}$, so that it would be included in calculating the overall difficulty.
- As for adjustability, as seen in Screenshot 32, the designer can adjust the increasing difficulty percentages and when to increase or decrease them based on what performance value. Also, almost every individual formula $\boldsymbol{Fi}$ and the overall formula $\boldsymbol{F}$ have weights as parameters for determining the importance of each formula's element. These can also be adjusted according to what the designer sees fit.

7.3 Alternative Approach

The way I implemented DDA in this game, was by giving a difficulty value to it. Then, based on that value, every gameplay challenge tries to adjust itself to match that difficulty and thus we get a collective overall difficulty. I am writing this now to mention that with the help of the formulas, one can also take a different approach in implementing a DDA system a bit different from mine that would keep on track the performance of the player in each gameplay challenge. And based on each performance in a challenge, only the difficulty regarding that gameplay challenge would change. So instead of changing the overall difficulty, only a specific part of it would be changed. For instance, the player's interactions with the obstacles can be tracked to check if the player is struggling with them differently than what the designer intended. That can be - for instance -by spending a lot of time overcoming them or taking a lot of damage from them. If so, the obstacles' formula $\boldsymbol{F5}$ can be used to only adjust the obstacles' difficulty. Thus, only the difficulty problem regarding the gameplay challenge of obstacles would get fixed.

I believe that this thesis provided an insight into the usefulness of the mathematical formulas in crafting a decent DDA system that is not perfect, but flexible enough to be adjusted and expanded for other games.

References

1. [Alena, Paraschos22] Alena Denisova and Paraschos Moschovitis. *Keep Calm and Aim for the Head: Biofeedback-Controlled Dynamic Difficulty Adjustment in a Horror Game,* 06.2022. https://www.researchgate.net/publication/361077195_Keep_Calm_and_Aim_for_the_Head_Biofeedback-Controlled_Dynamic_Difficulty_Adjustment_in_a_Horror_Game

2. [Bycer20] Josh Bycer. *How to Design Effective Difficulty in Video Games*. 05.10.2020 https://medium.com/super-jump/how-to-design-effective-difficulty-in-video-games-dc6692ba0d4f

3. [Csikszentmihaly90] Mihaly Csikszentmihaly. *Flow: The Psychology of Optimal Experience*. 1990

4. [Fernandez, Mikami, Kondo18] Henry Fernandez, Koji Mikami, Kunio Kondo. *Perception of Difficulty in 2D Platformers Using Graph Grammars.* 07.07.2018 http://adada.info/journals/Vol.22_No.2-1.pdf

5. [Hunicke05] Robin Hunicke. *The case for dynamic difficulty adjustment in games.* 06.2005. https://www.researchgate.net/publication/220982524_The_case_for_dynamic_difficulty_adjustment_in_games

6. [Rami, Guy, Meirav17] Rami Puzis, Guy Shani and Meirav Taieb-Maimon. *EEG-Triggered Dynamic Difficulty Adjustment for Multiplayer Games,* 12.2017. https://www.researchgate.net/publication/321469435_EEG-Triggered_Dynamic_Difficulty_Adjustment_for_Multiplayer_Games

7. [Rodriguez19] Guillermo Romera Rodriguez. *Player Modeling for Dynamic Difficulty Adjustment in Top Down Action Games.* 05.2019. https://repository.library.northeastern.edu/files/neu:m0455c22w/fulltext.pdf

8. [Palban21] Vince Palban. *Managing Difficulty In Games*. 2021 https://ecampusontario.pressbooks.pub/gamedesigndevelopmenttextbook/chapter/managing-difficulty-in-games/

9. [Schreiber10] Ian Schreiber. *Game Balance Concepts: A continued experiment in game design andnteaching*. 08 09.2010 https://gamebalanceconcepts.wordpress.com/

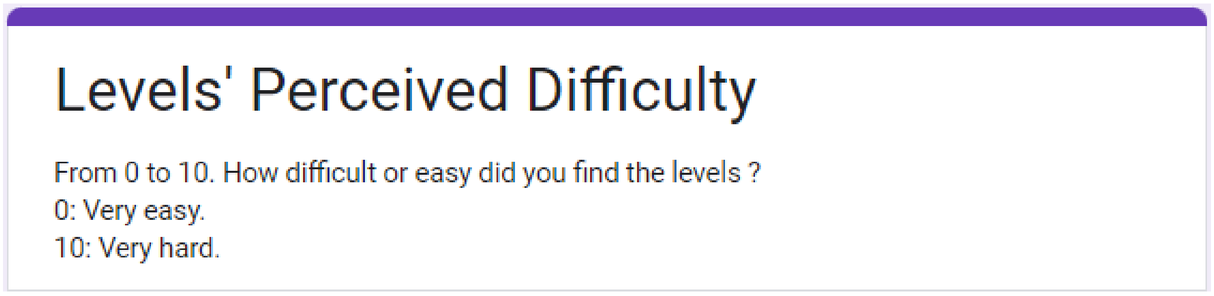
10. [Valério17] Ricardo Valério. *Difficulty in Game Design, flow motivation* and *learning curves.* 10.07.2017 https://ricardo-valerio.medium.com/make-it-difficult-not-punishing-7198334573b8

11. [Wang21] Scarlett (Huiyu) Wang. *Depth vs Complexity: Self-Efficacy's Variation and Player Performance's Fluctuation in a Game Containing Two Different Versions of Dynamic Difficulty Adjustment.* 05.2021. https://repository.library.northeastern.edu/files/neu:bz617n15s/fulltext.pdf

12. [Frothingham19] Scott Frothingham. *What Is Negativity Bias, and How Does It Affect You?* 17.12.2019
https://www.healthline.com/health/negativity-bias

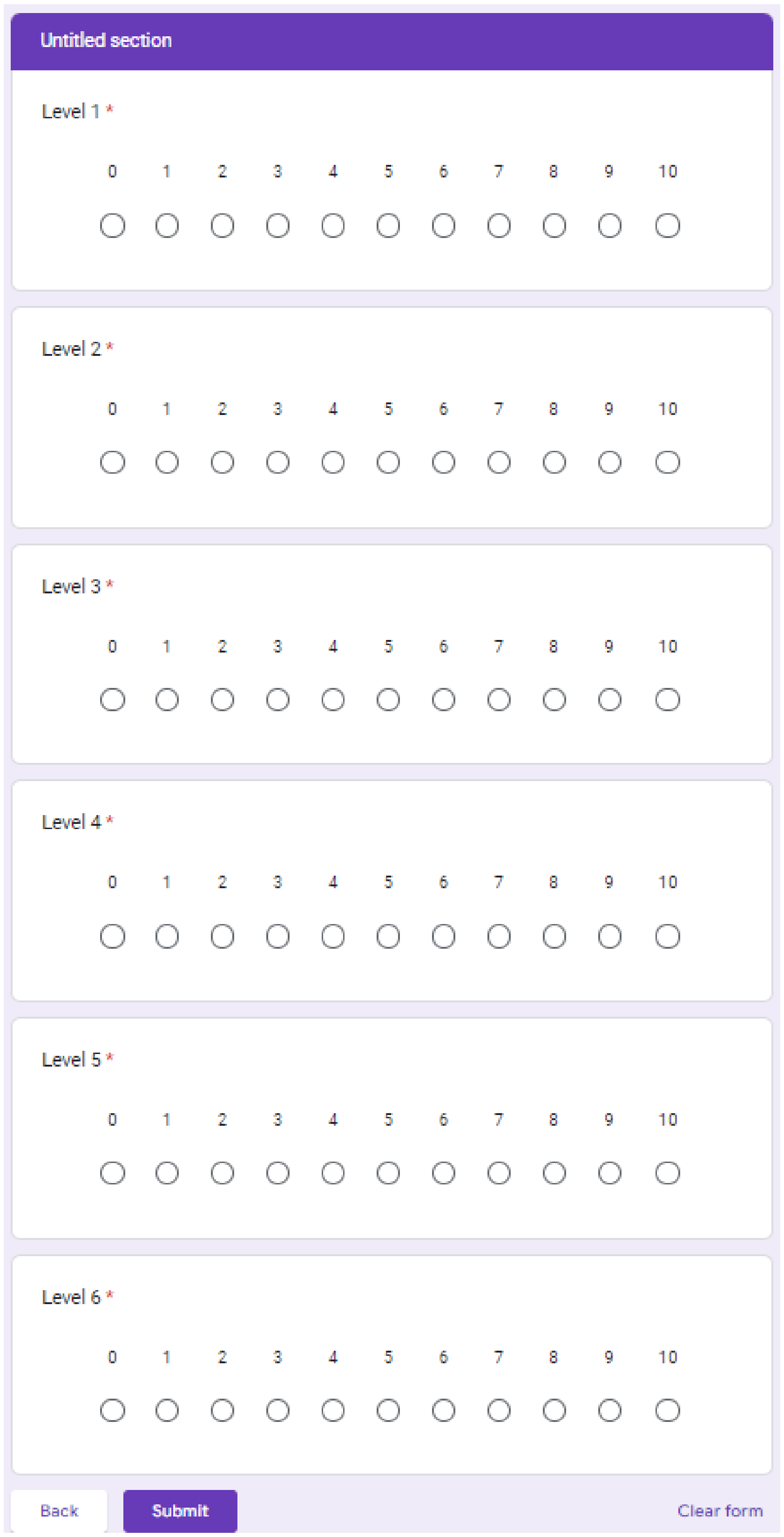
Appendix

First Survey


Levels' Perceived Difficulty

From 0 to 10. How difficult or easy did you find the levels ?
0: Very easy.
10: Very hard.

Screenshot 33: *First survey's description.*


Untitled section

Level 1 *

0	1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○	○

Level 2 *

0	1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○	○

Level 3 *

0	1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○	○

Level 4 *

0	1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○	○

Level 5 *

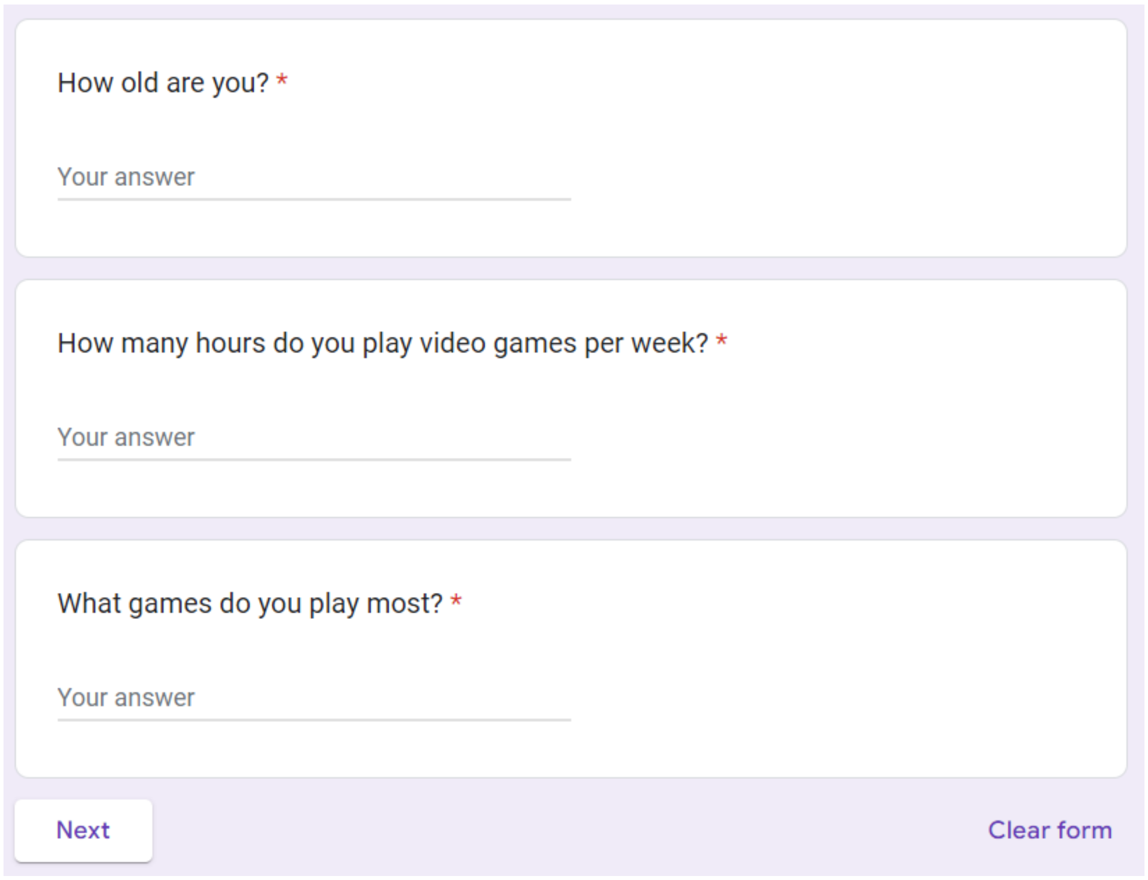
0	1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○	○

Level 6 *

0	1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○	○

Screenshot 34: *The levels from zero to six to be rated.*

Second Survey



How old are you? *

Your answer

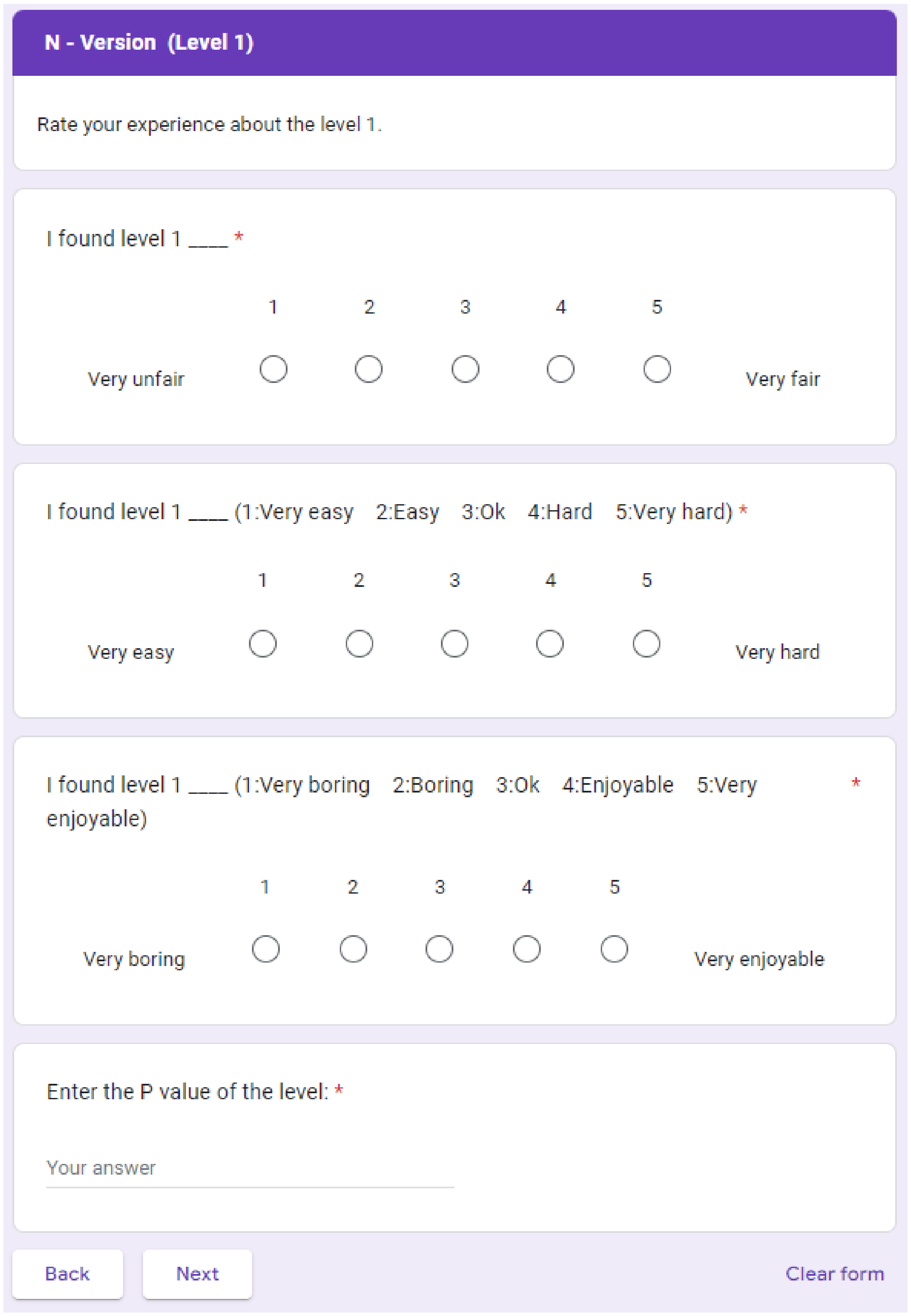
How many hours do you play video games per week? *

Your answer

What games do you play most? *

Your answer

Screenshot 35: *General questions about the participant's gaming experience.*



N - Version (Level 1)

Rate your experience about the level 1.

I found level 1 ____ *

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 1 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard) *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

I found level 1 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable) *

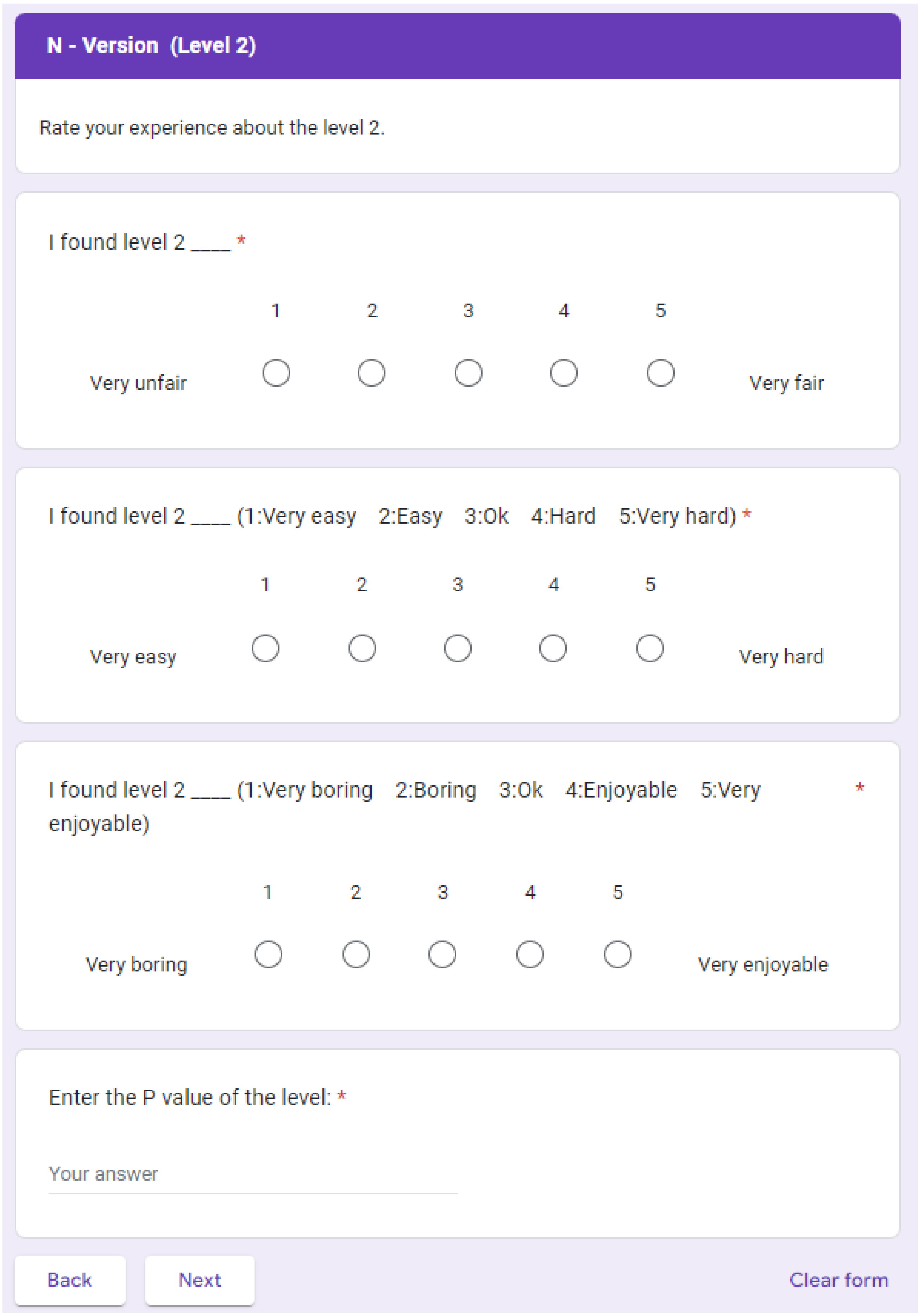
	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

Enter the P value of the level: *

Your answer

Screenshot 36: *Rating level 1 in the N-Version.*

Note: *P value is the performance.*



N - Version (Level 2)

Rate your experience about the level 2.

I found level 2 ____ *

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 2 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard) *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

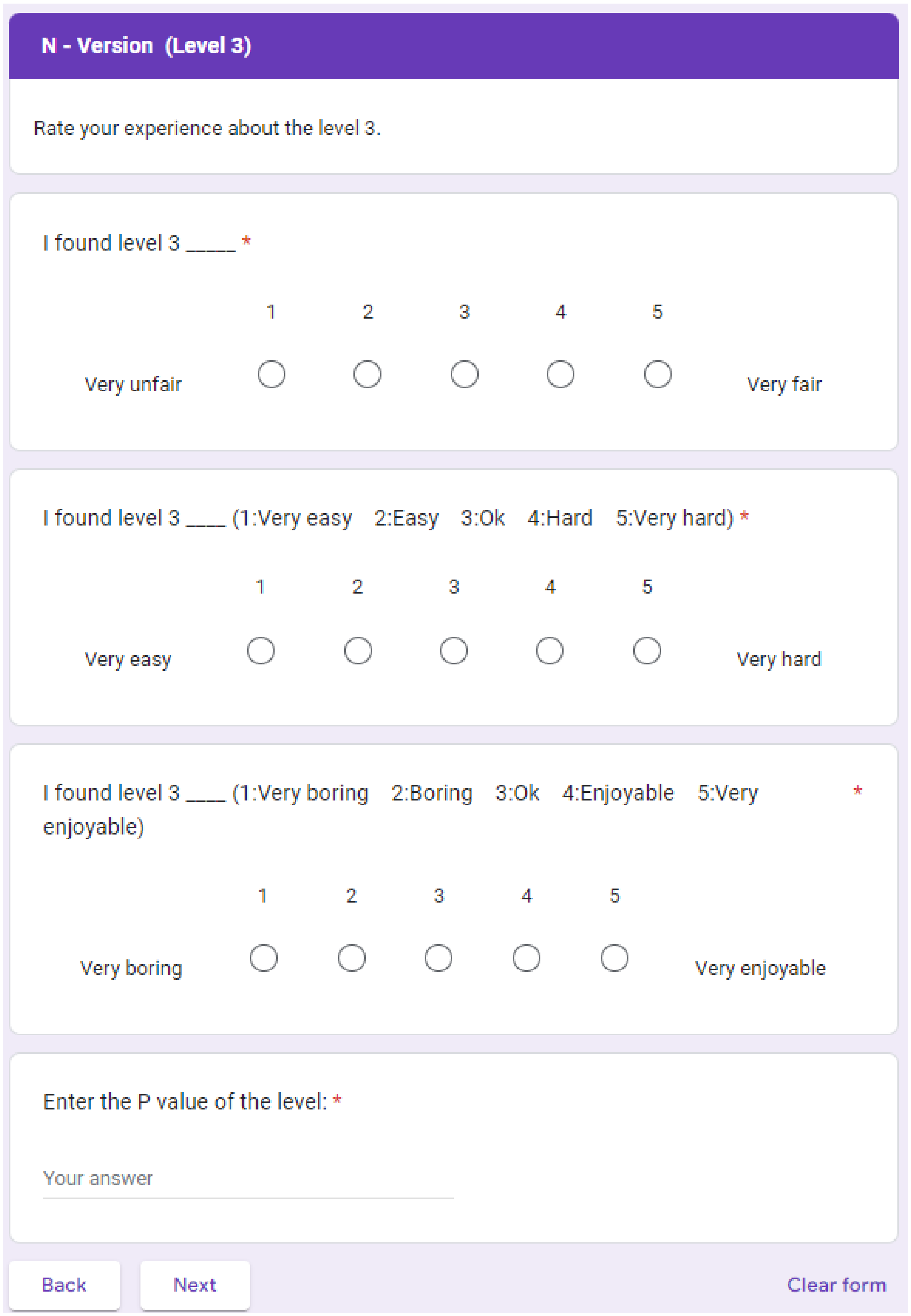
I found level 2 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable) *

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

Enter the P value of the level: *

Your answer

Screenshot 37: *Rating level 2 in the N-Version.*



N - Version (Level 3)

Rate your experience about the level 3.

I found level 3 _____ *

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 3 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard) *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

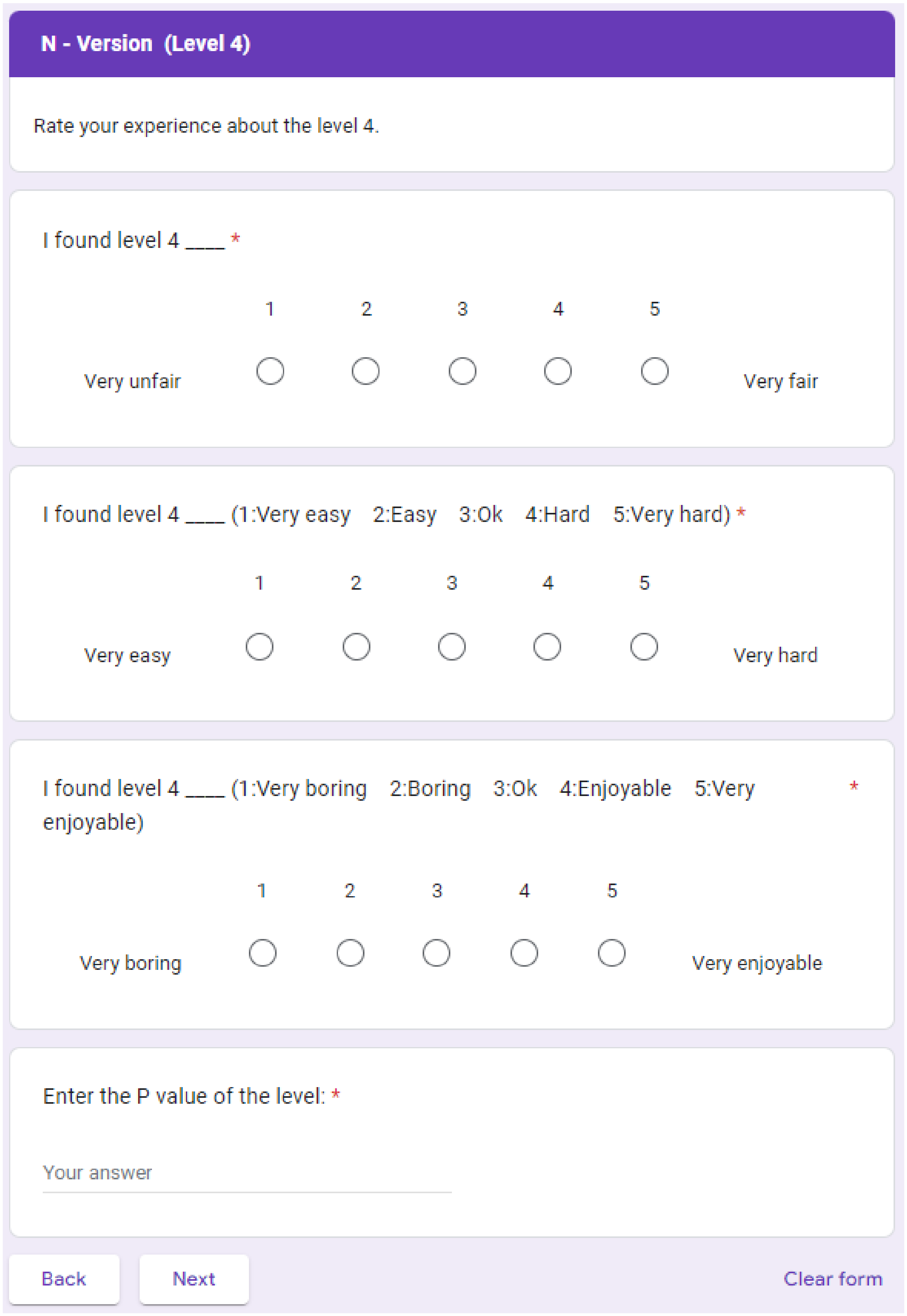
I found level 3 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable) *

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

Enter the P value of the level: *

Your answer

Screenshot 38: *Rating level 3 in the N-Version.*


N - Version (Level 4)

Rate your experience about the level 4.

I found level 4 ____ *

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 4 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard) *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

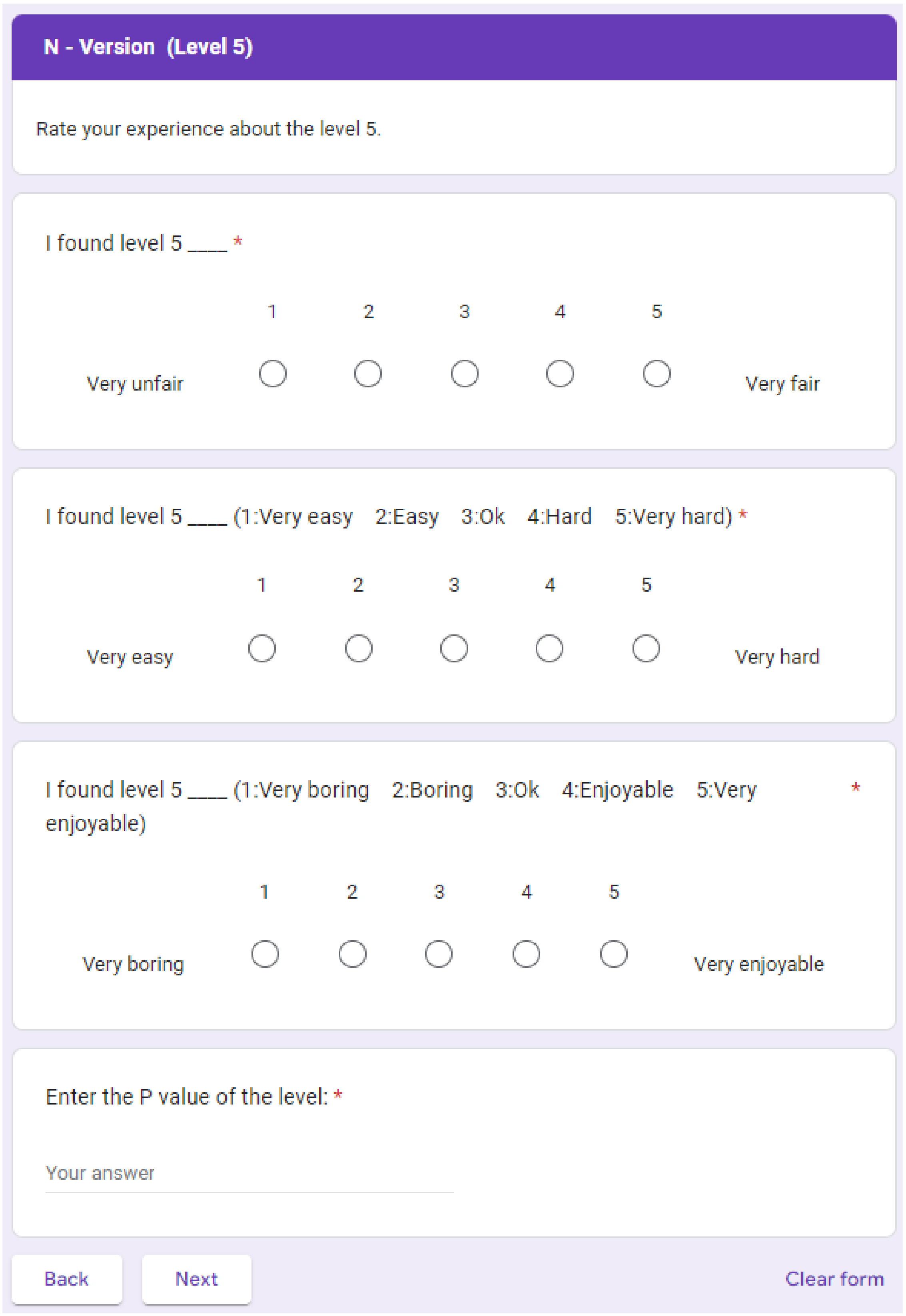
I found level 4 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable) *

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

Enter the P value of the level: *

Your answer

Screenshot 39: *Rating level 4 in the N-Version.*



N - Version (Level 5)

Rate your experience about the level 5.

I found level 5 ____ *

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 5 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard) *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

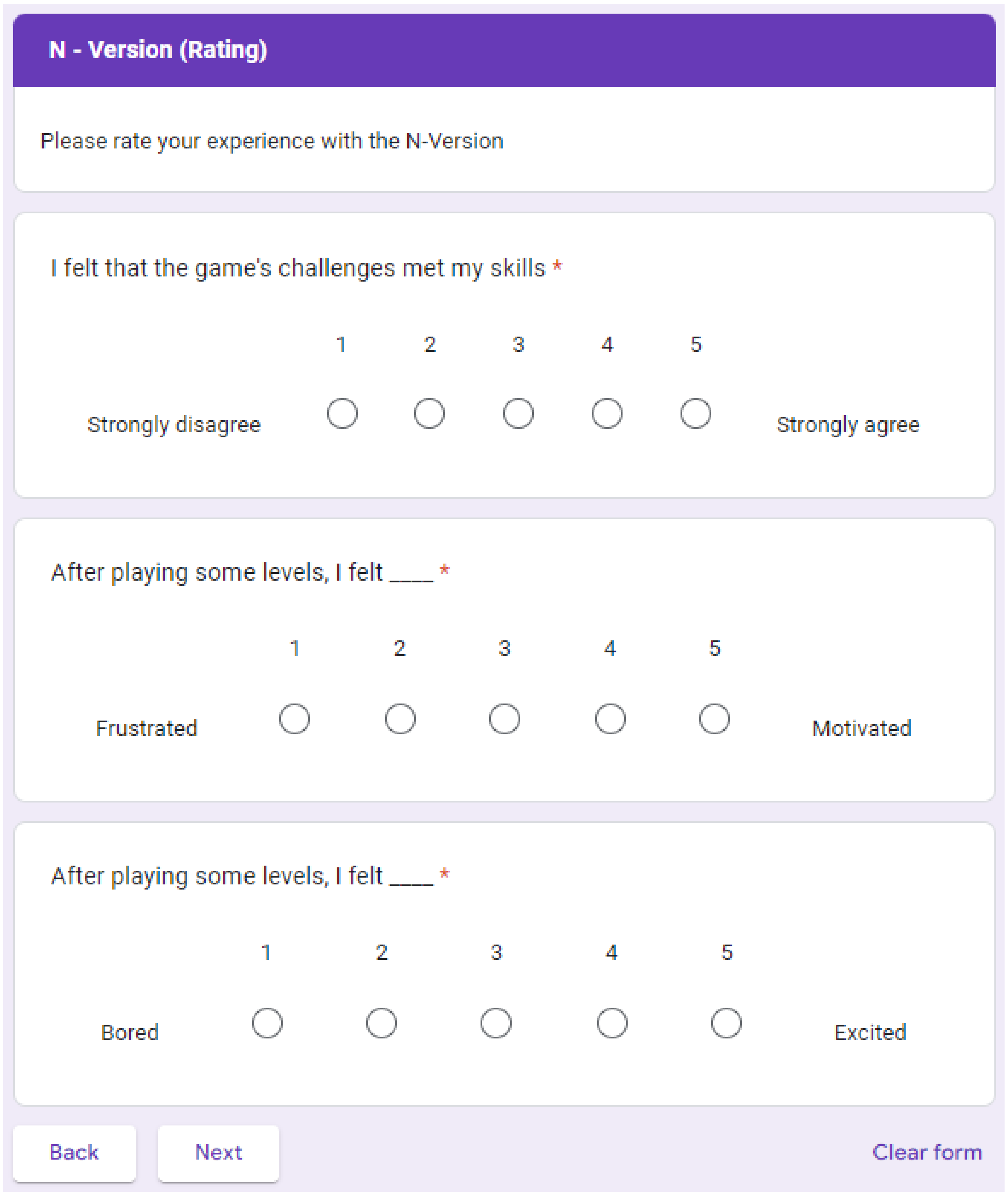
I found level 5 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable) *

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

Enter the P value of the level: *

Your answer

Screenshot 40: *Rating level 5 in the N-Version.*



N - Version (Rating)

Please rate your experience with the N-Version

I felt that the game's challenges met my skills *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

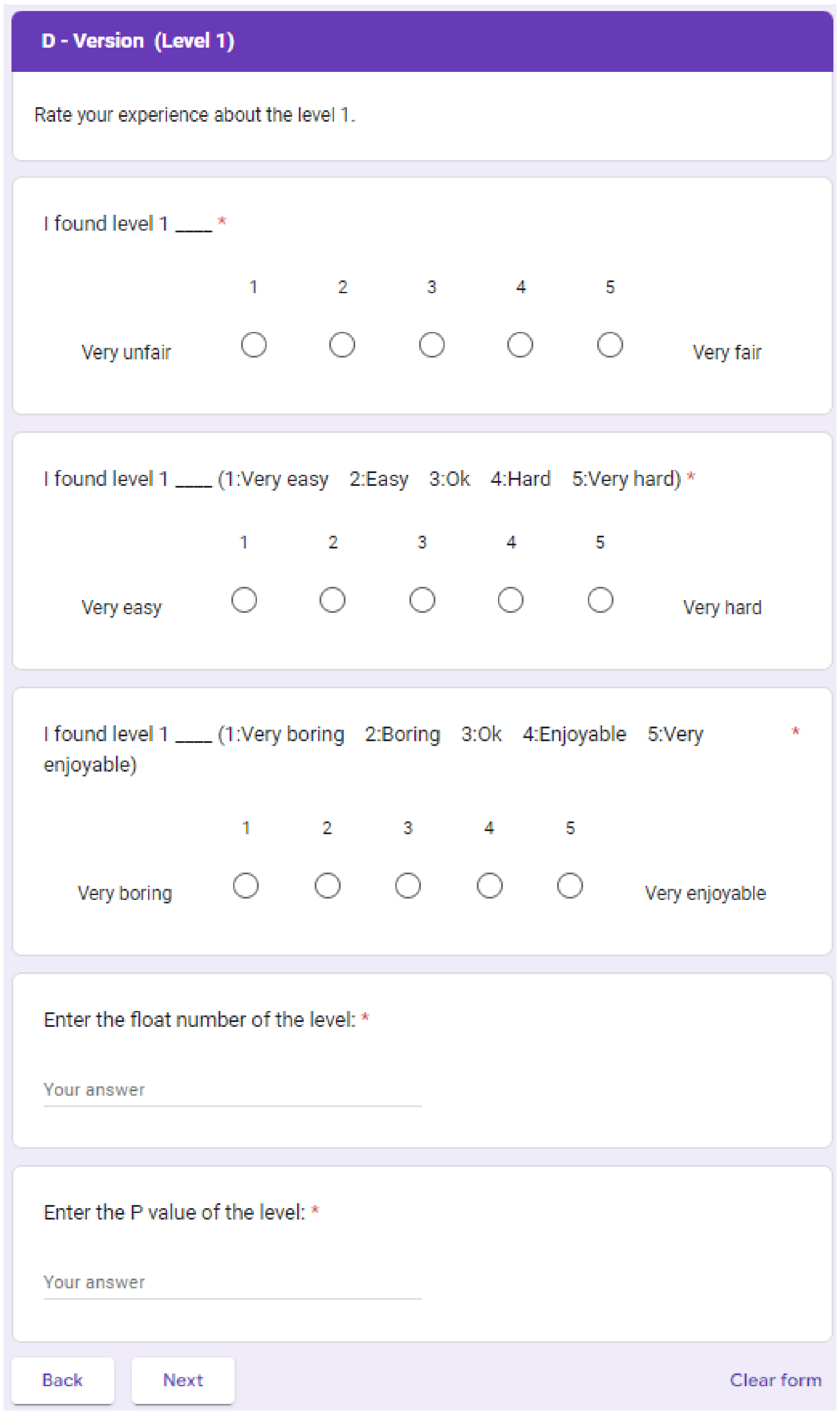
After playing some levels, I felt ____ *

	1	2	3	4	5	
Frustrated	○	○	○	○	○	Motivated

After playing some levels, I felt ____ *

	1	2	3	4	5	
Bored	○	○	○	○	○	Excited

Screenshot 41: *Rating the N-Version's experience.*



D - Version (Level 1)

Rate your experience about the level 1.

I found level 1 ____ *

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 1 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard) *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

I found level 1 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable) *

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

Enter the float number of the level: *

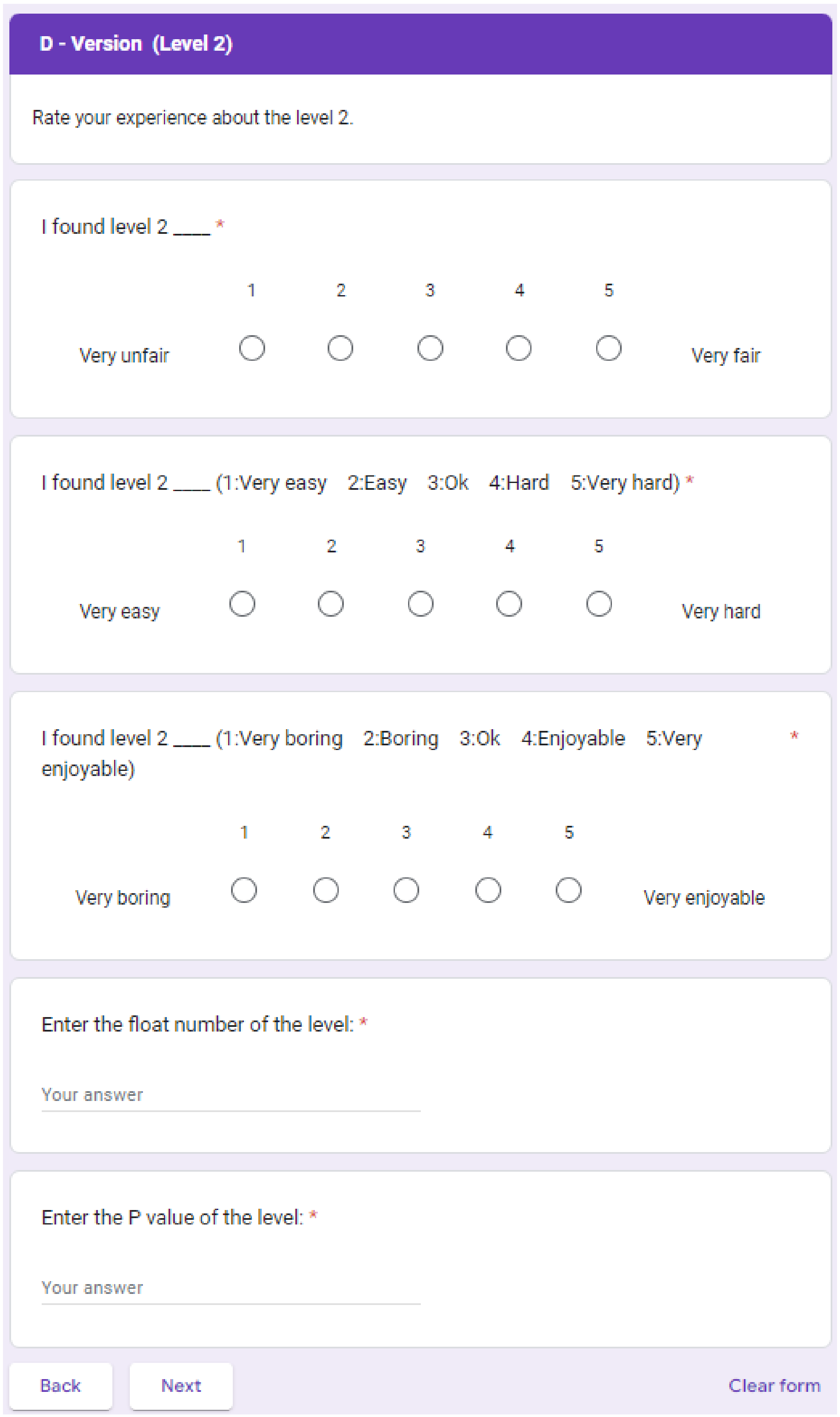
Your answer

Enter the P value of the level: *

Your answer

Screenshot 42: *Rating level 1 in the D-Version.*

Note: *Float number is the level's difficulty.*



D - Version (Level 2)

Rate your experience about the level 2.

I found level 2 ____ *

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 2 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard) *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

I found level 2 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable) *

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

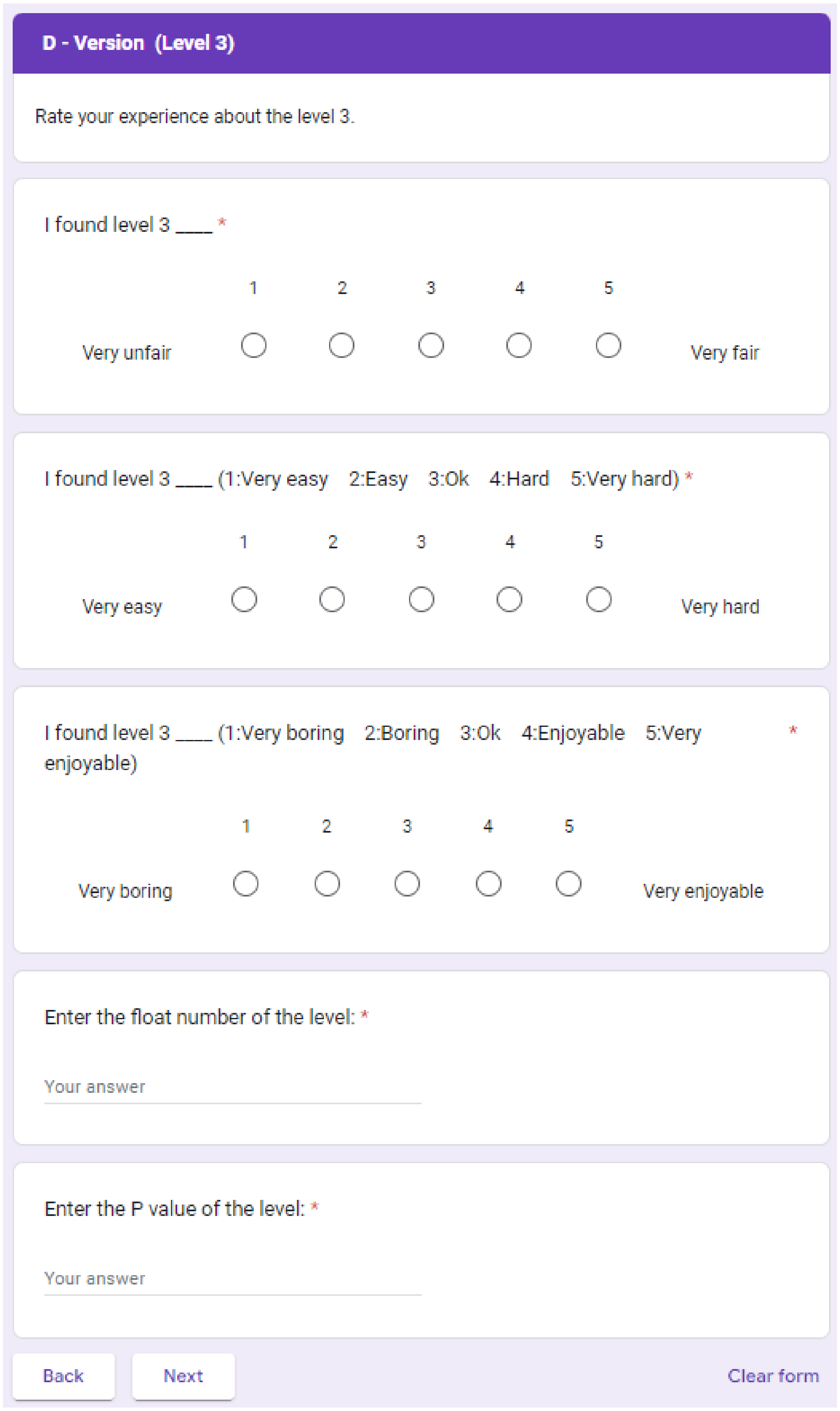
Enter the float number of the level: *

Your answer

Enter the P value of the level: *

Your answer

Screenshot 43: *Rating level 2 in the D-Version.*



D - Version (Level 3)

Rate your experience about the level 3.

I found level 3 ____ *

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 3 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard) *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

I found level 3 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable) *

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

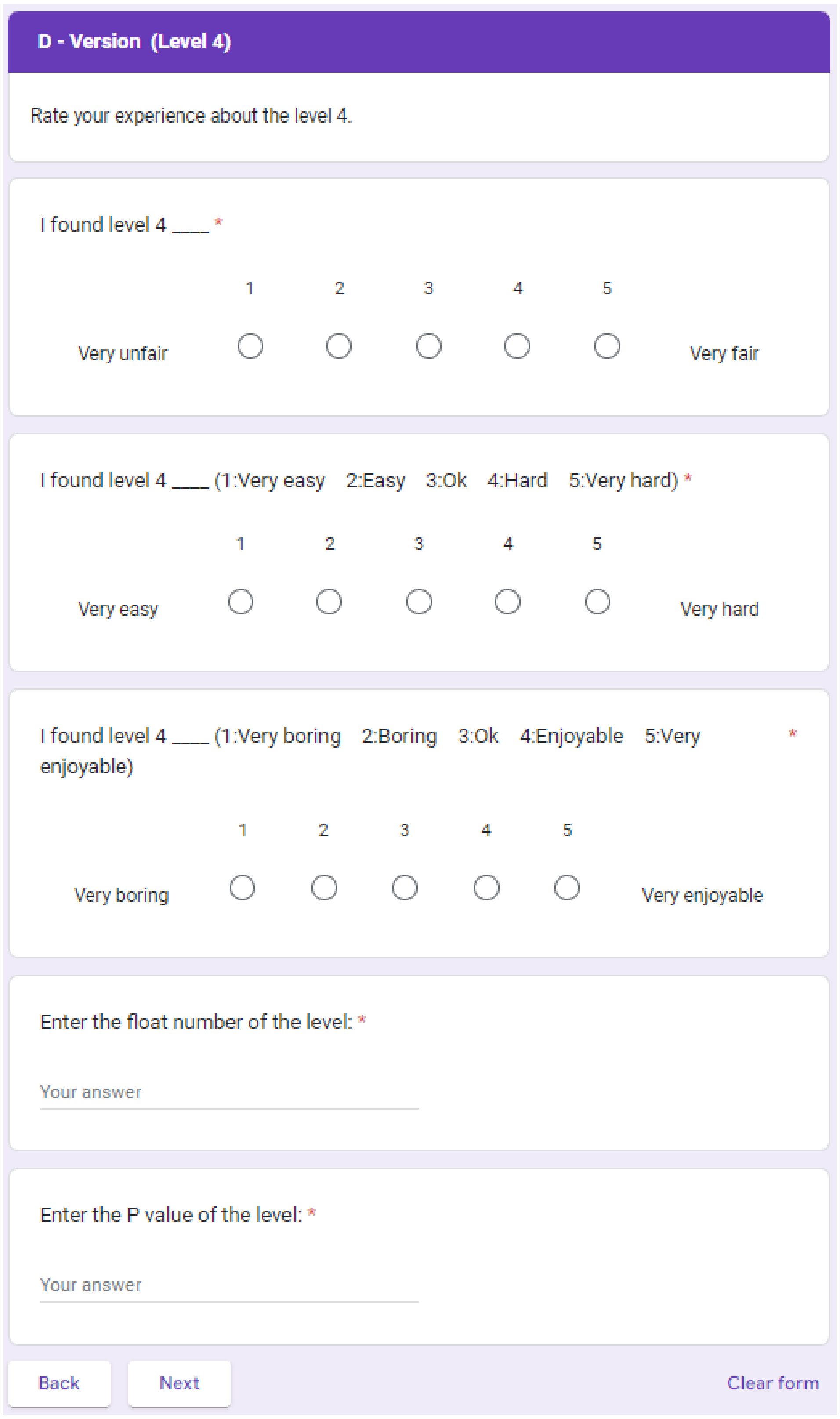
Enter the float number of the level: *

Your answer

Enter the P value of the level: *

Your answer

Screenshot 44: *Rating level 3 in the D-Version.*



D - Version (Level 4)

Rate your experience about the level 4.

I found level 4 ____ *

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 4 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard) *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

I found level 4 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable) *

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

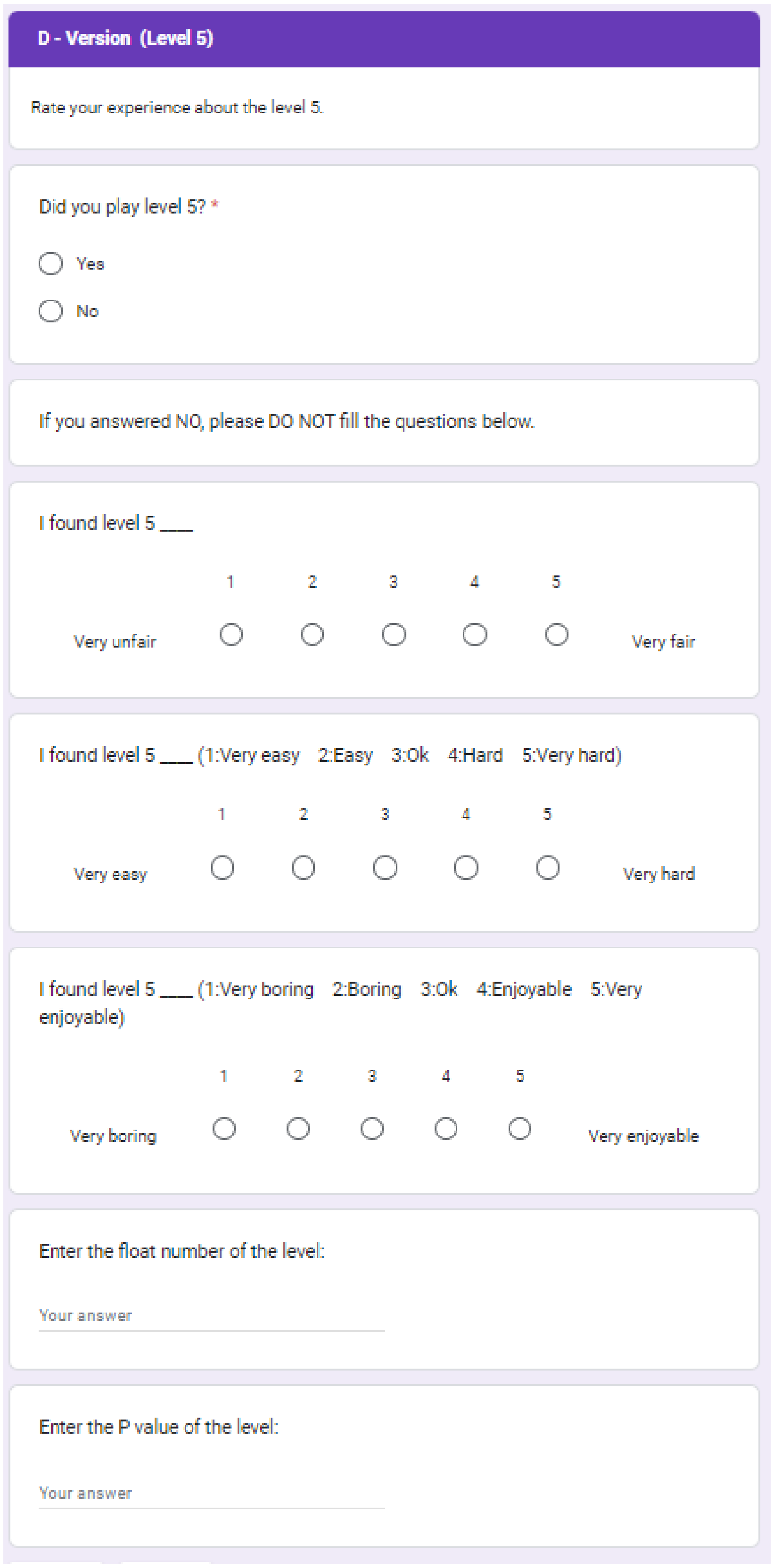
Enter the float number of the level: *

Your answer

Enter the P value of the level: *

Your answer

Screenshot 45: *Rating level 4 in the D-Version.*



D - Version (Level 5)

Rate your experience about the level 5.

Did you play level 5? *

Yes

No

If you answered NO, please DO NOT fill the questions below.

I found level 5 ____

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 5 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard)

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

I found level 5 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable)

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

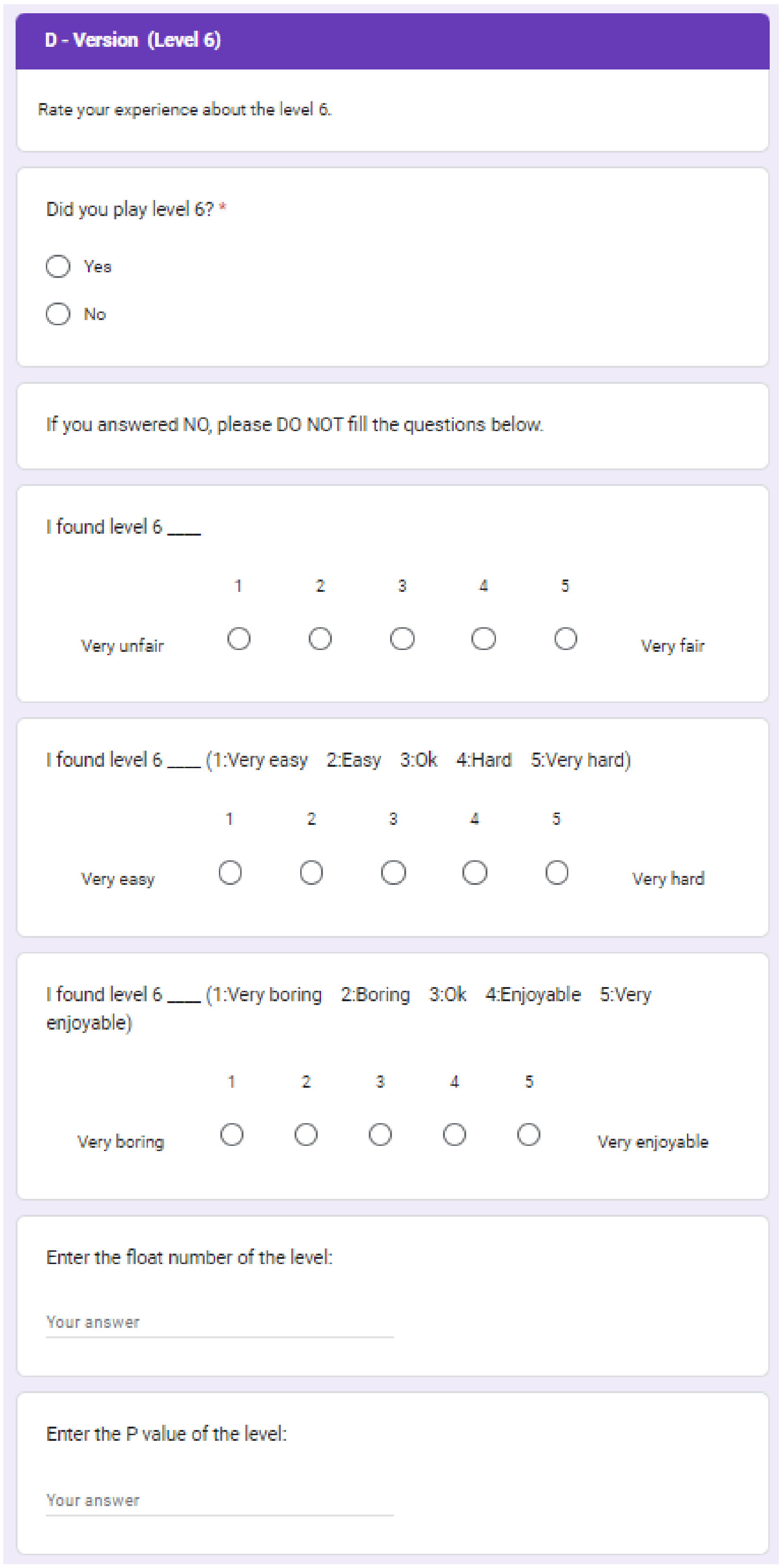
Enter the float number of the level:

Your answer

Enter the P value of the level:

Your answer

Screenshot 46: *Rating level 5 in the D-Version.*



D - Version (Level 6)

Rate your experience about the level 6.

Did you play level 6? *

Yes

No

If you answered NO, please DO NOT fill the questions below.

I found level 6 ____

	1	2	3	4	5	
Very unfair	○	○	○	○	○	Very fair

I found level 6 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard)

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

I found level 6 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable)

	1	2	3	4	5	
Very boring	○	○	○	○	○	Very enjoyable

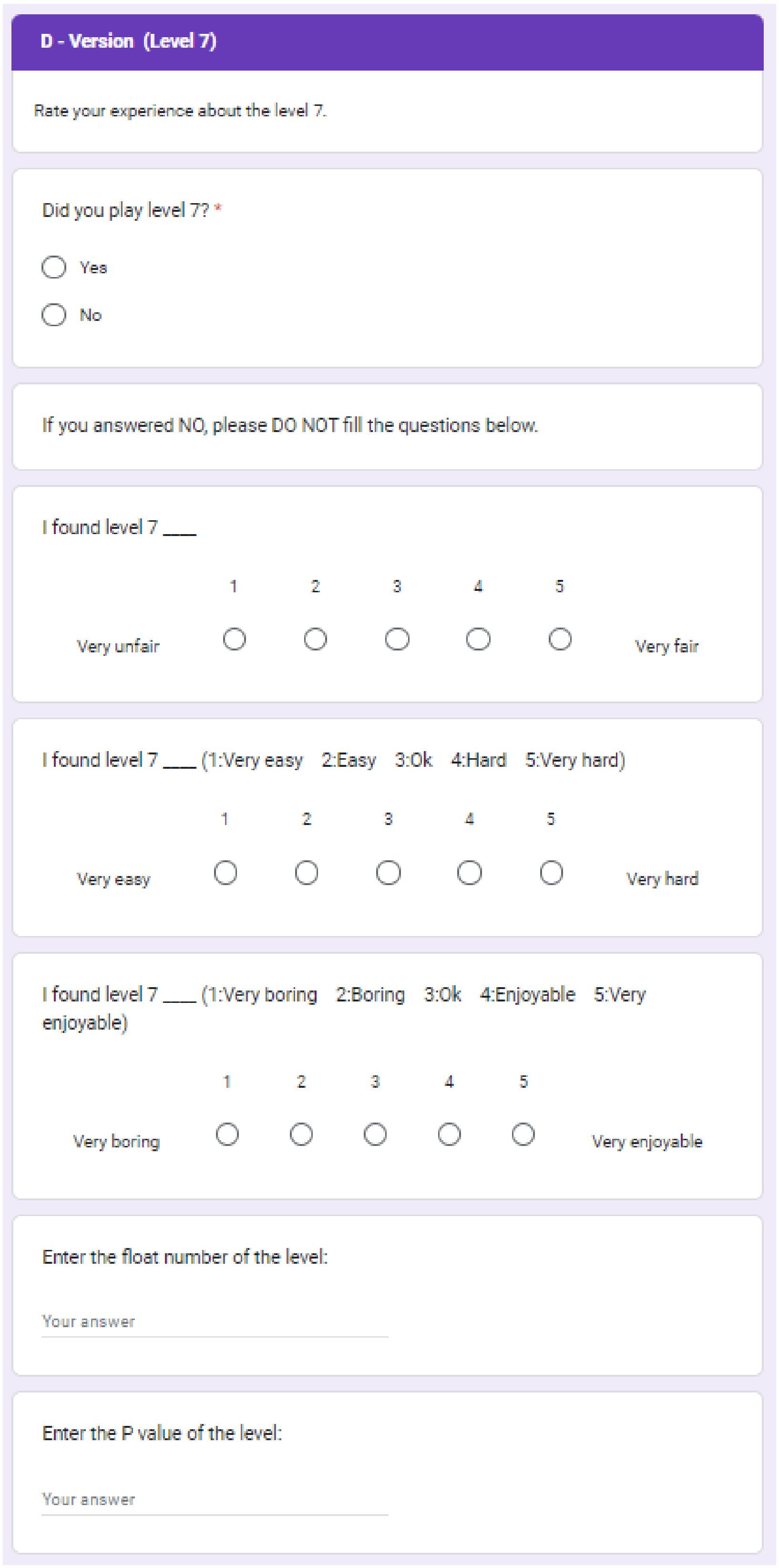
Enter the float number of the level:

Your answer

Enter the P value of the level:

Your answer

Screenshot 47: *Rating level 6 in the D-Version.*



D - Version (Level 7)

Rate your experience about the level 7.

Did you play level 7? *

◯ Yes

◯ No

If you answered NO, please DO NOT fill the questions below.

I found level 7 ___

	1	2	3	4	5	
Very unfair	◯	◯	◯	◯	◯	Very fair

I found level 7 ___ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard)

	1	2	3	4	5	
Very easy	◯	◯	◯	◯	◯	Very hard

I found level 7 ___ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable)

	1	2	3	4	5	
Very boring	◯	◯	◯	◯	◯	Very enjoyable

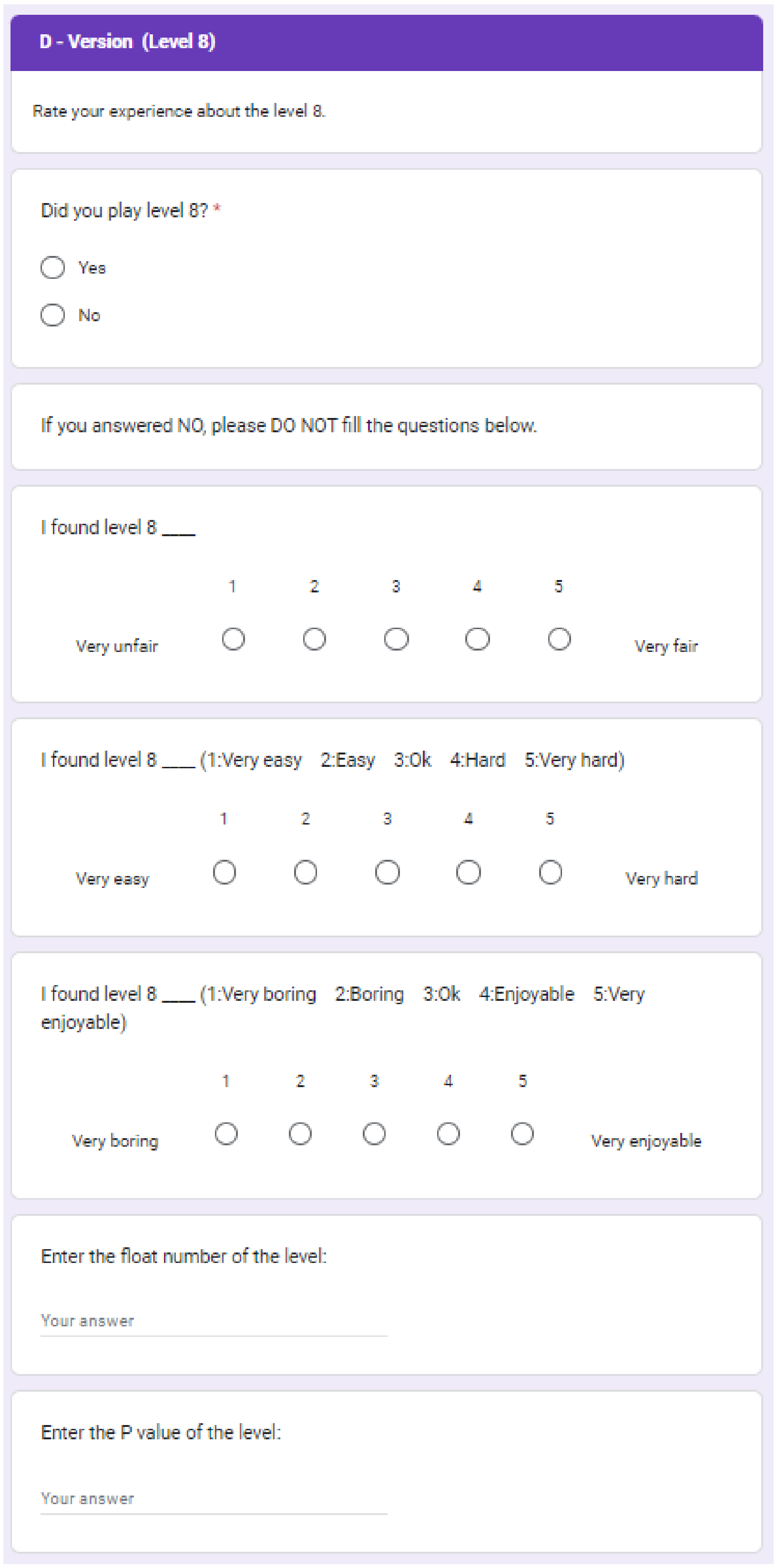
Enter the float number of the level:

Your answer

Enter the P value of the level:

Your answer

Screenshot 48: *Rating level 7 in the D-Version.*



D - Version (Level 8)

Rate your experience about the level 8.

Did you play level 8? *

- ◯ Yes
- ◯ No

If you answered NO, please DO NOT fill the questions below.

I found level 8 ____

	1	2	3	4	5	
Very unfair	◯	◯	◯	◯	◯	Very fair

I found level 8 ____ (1:Very easy 2:Easy 3:Ok 4:Hard 5:Very hard)

	1	2	3	4	5	
Very easy	◯	◯	◯	◯	◯	Very hard

I found level 8 ____ (1:Very boring 2:Boring 3:Ok 4:Enjoyable 5:Very enjoyable)

	1	2	3	4	5	
Very boring	◯	◯	◯	◯	◯	Very enjoyable

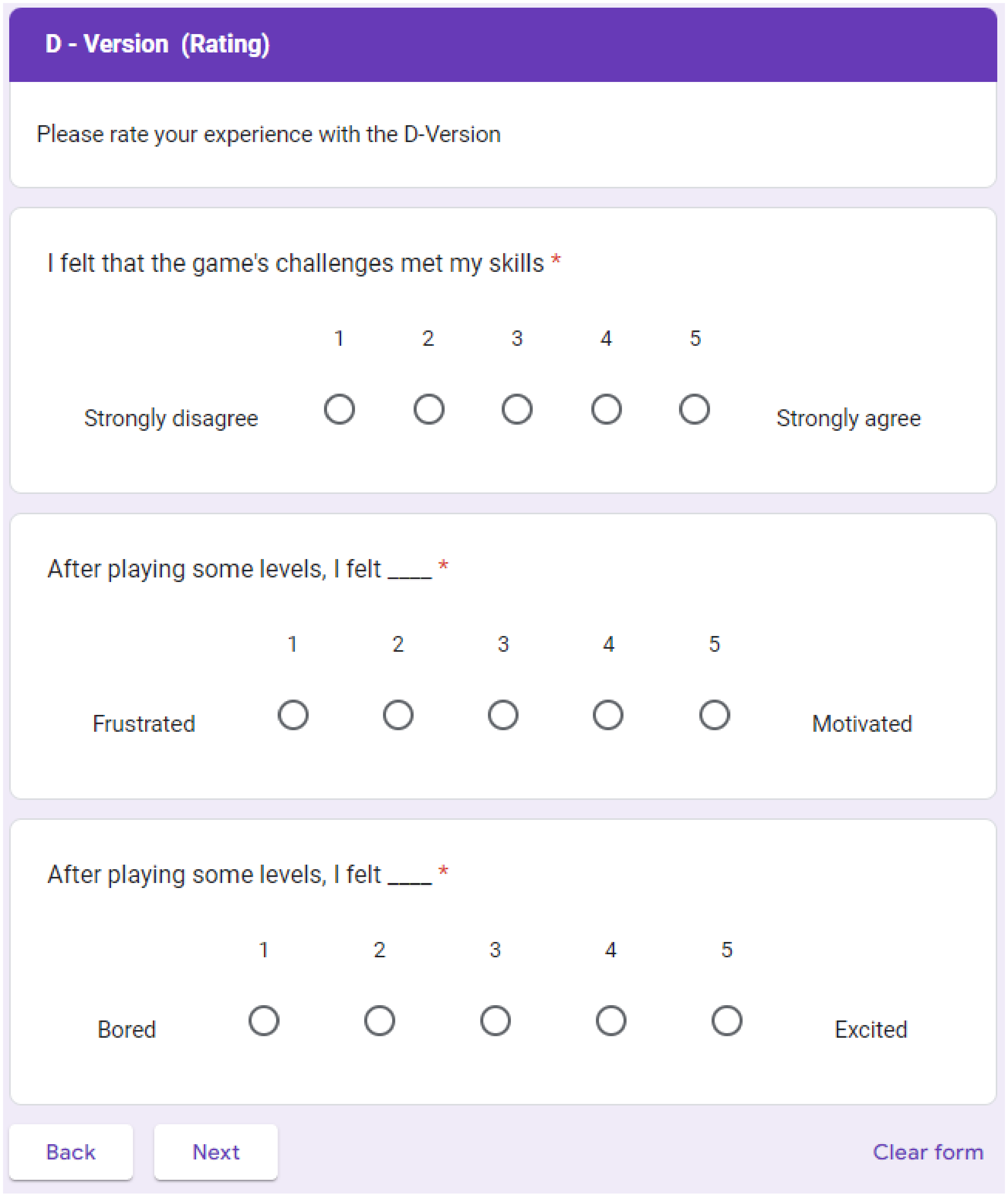
Enter the float number of the level:

Your answer

Enter the P value of the level:

Your answer

Screenshot 49: *Rating level 8 in the D-Version.*



D - Version (Rating)

Please rate your experience with the D-Version

I felt that the game's challenges met my skills *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

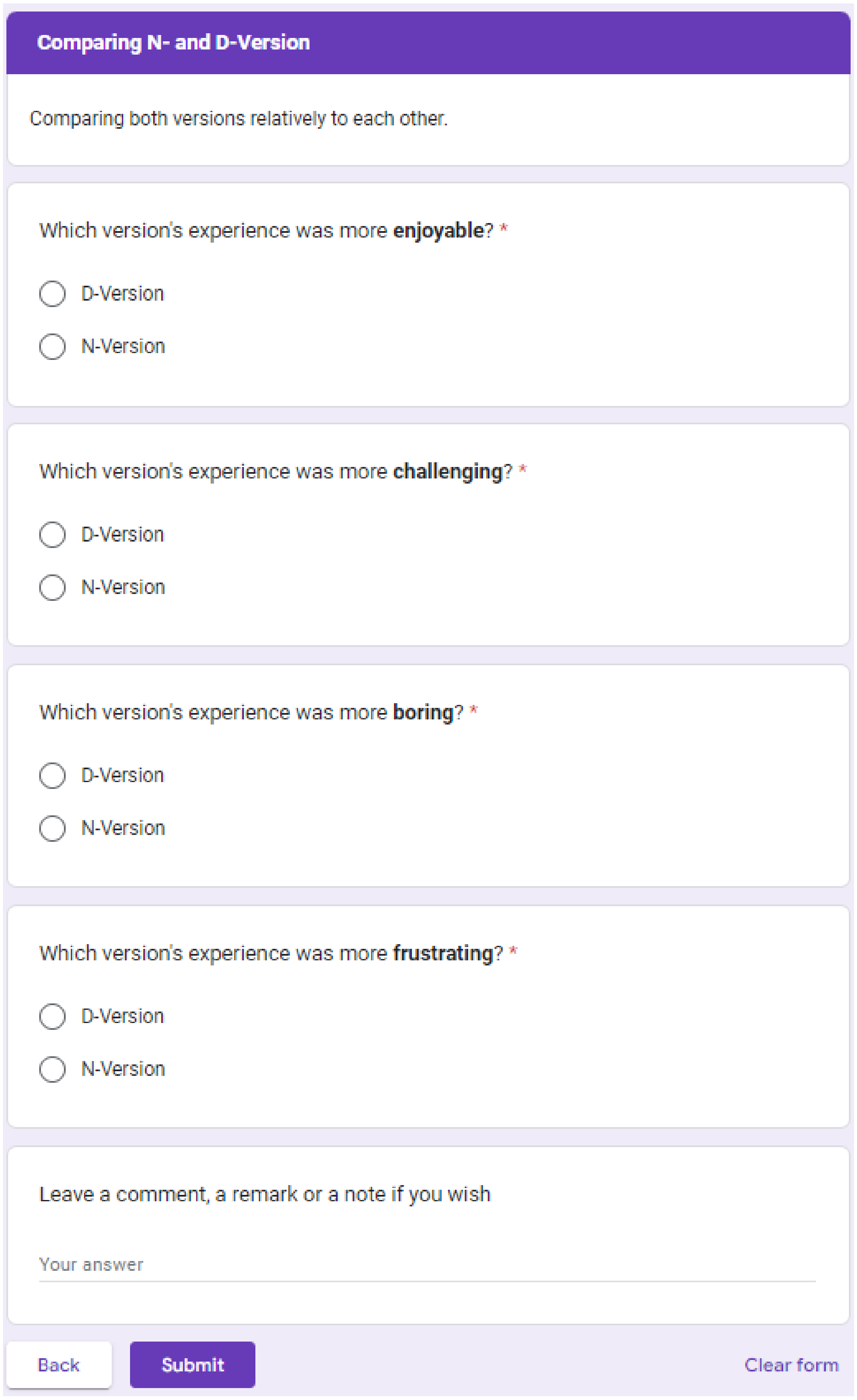
After playing some levels, I felt ____ *

	1	2	3	4	5	
Frustrated	○	○	○	○	○	Motivated

After playing some levels, I felt ____ *

	1	2	3	4	5	
Bored	○	○	○	○	○	Excited

Screenshot 50: *Rating the D-Version's experience.*


Comparing N- and D-Version

Comparing both versions relatively to each other.

Which version's experience was more **enjoyable**? *

- ◯ D-Version
- ◯ N-Version

Which version's experience was more **challenging**? *

- ◯ D-Version
- ◯ N-Version

Which version's experience was more **boring**? *

- ◯ D-Version
- ◯ N-Version

Which version's experience was more **frustrating**? *

- ◯ D-Version
- ◯ N-Version

Leave a comment, a remark or a note if you wish

Your answer

Screenshot 51: *Comparing N-Version with D-Version.*