Gotta adjust them all!

-

Dynamic Difficulty Adjustment of Role-Playing Games
Through Procedural Generation of Non-Player Characters

Andrei Popa
a.popa@student.utwente.nl

Supervisors

Marcus Gerhold
Vadim Zaytsev
Robby van Delden
Dennis Reidsma

May 22, 2023

Abstract

The optimal level and progression of challenge are critical factors for delivering an enjoyable gaming experience. In this study, I have developed a novel system that generates non-player character's (NPC) attributes, actions, and affixes in a role-playing game using generative grammars.

The proposed system is designed to adapt the difficulty of the game after every level by generating NPCs that match the player's skill level. Through a series of experiments, the system demonstrated its potential to provide an increased level of curiosity for experienced players compared to the version of the game without the proposed framework. Additionally, it showed an increase in the perceived level of challenge by the player and an increase in the variety of actions used during gameplay.

These results demonstrate the effectiveness of the proposed system in creating more dynamic and challenging gameplay experiences while maintaining balance and playability. Furthermore, the introduction of augments for the generative grammar to improve the generation of attributes, actions, and affixes represents a novel contribution to the field of dynamic difficulty adjustment.

Contents

1 Introduction

Since the breakthrough of the game Pong© in 1972, the video game industry has evolved from a focused market to mainstream. Nowadays, video games have reached a point where players around the world take part in international competitions to see who is better. They bring communities together through streaming platforms, where the streamers play different games and others can watch and interact with them. One such example is DotA International, the world DotA 2 championship, which had a prize pool of 40 million USD in 2021[1]. The evolution of video games has also allowed the development of different genres and sub-genres. These define the way the game is played and the main mechanics of it. These were some of the more popular genres in 2020[2]: *Battle Royale* games, *Role-Playing* games, *Fighting* games and *First Person Shooter* games.

Challenge is one of the main catalysts of a video game and one of the main aspects of the design and balancing process. There are two types of difficulty present in video games:

- Static difficulty, where the player chooses the level of difficulty, or it is already predefined.
- Dynamic difficulty adjustment (DDA), where the difficulty of the game changes based on the abilities of the player.

Most of the modern video games give the players the option to select the difficulty of their game. While this can provide the needed challenge for some players, it can happen that players do not achieve complete immersion, also known in more scientific terms as the flow state [38]. Moreover, for a player to choose the difficulty suited for their skills, they need previous experience. Consequently, newcomers might think they can play on the hardest difficulty, but this will just ruin their experience and immersion.

The flow state, also known as the state of being in the zone, is a mental state in which the person performing some activity, like video games, sports, reading, gradening or rock climbing [43], becomes fully immersed by it. M. Csikszentmihalyi [46, §18] says that there are several milestones a player or game has to reach in order for the player to fully achieve flow state:

- There is a right balance between the skills of a player and the challenge of the game.
- The player reaches a sense of ecstasy.
- The player is completely involved in their actions, and they know exactly what to do.
- The player feels like time passes way faster.
- The player experiences a sense of serenity.
- Whatever produces the flow becomes the reward.

If the game is too difficult, then the player will get frustrated, otherwise they will get bored [46, §18]. In the book The Concept of Flow, it is said that "The flow state is intrinsically rewarding and leads the individual to seek to replicate flow experiences" [37, p. 244]. Hence, it can be said that by designing a game experience where the player will reach the flow state, their engagement will increase as they strive to replicate it.

Designing a game experience to reach flow can differ from player to player. Bartle [5], Koster [30] and Marczewski [35] have each in their own ways established that there are different types of players which have different ways towards learning and motivation. This translates to some players that would prefer to play on an easier difficulty at first and then either steadily increase it, while others jump directly to the highest one. For games with a static difficulty, the player would have to stop or pause the game or finish the current level to change the difficulty in the settings, which can break immersion. Zohaib [54] mentions that the flow model prevents frustration during difficult moments and boredom during simple situations. Moreover, he says that the role of DDA is to retain the interest throughout the game and offer an appropriate challenge level. Therefore, I hypothesize that using DDA, the flow state will be maintained, while the level of the challenge will remain appropriate for the player's

[1] `https://www.dota2.com/esports/ti10/watch/0/0/series`

[2] `https://straitsresearch.com/blog/top-10-most-popular-gaming-genres-in-2020/`

abilities. In addition, using DDA the difficulty level of the game will be suited for players with varying skill levels and consequently be more welcoming to new players.

Before reaching the state of flow, video games also have to motivate the player. One video game genre which is well know for its ability to immerse players are role-playing games [6]. In role-playing games there are two main motivation factors: mechanistic/tactical play and character-based/social play, both being tied to main characteristics of the genre [49]. Role-Playing games (RPGs) are a genre of video games, where the player takes control of a character, either predefined or created by them, and immerses themselves in a world full of quests, non-playable characters (NPCs) and other exploration objectives. There are several characteristics that each RPG has [50]:

- Story and Setting
- Exploration and Quests
- Items and Inventory
- Character Attributes and Actions
- Experience and Levels and Combat

For this thesis, I am going to focus on character's attributes, actions and affixes. Attributes represent basic characteristics of a player or NPC, such as strength, stamina, vitality which can influence the health points (HP) and physical/magical defense or attack of the character. Actions represent what a character can do in a game. These can be attacking, healing or even running away. Not as common as the attributes or actions are the affixes (passive abilities), unique keywords which set enemies apart from others by giving them unique abilities. These are usually introduced in games to make enemies stronger without having to increase their attributes. One such example are the Rare and Champion monsters[3] from the game Diablo III©. These monsters have a yellow or blue glow, have increased health, are stronger than normal ones and have one or more affixes. Because they are harder to defeat, they also drop more items. The existence of attributes, items and affixes in a game gives the player a feel of character depth and character uniqueness, which is part of the character-based/social play motivation factor [49]. Using enemies with different attributes and affixes, the game would give the player the feel of variety during their playthrough and as shown by Bagheri and Milyavskaya [4], novelty-variety can be considered a basic psychological need. Having these things in mind, I propose a DDA system that generates attributes, actions and affixes for role-playing games. The intent is to measure the player's engagement and to push them into adapting to different situations and not use the same tactics or actions.

The use of NPCs in adjusting the difficulty level of a game is not a new concept, as I will show in the following section. However, the innovation of this thesis lies in the method used to generate the NPCs' attributes, actions, and affixes and how they are utilized to adapt the difficulty of each level. This thesis proposes a dynamic approach for the creation of NPCs that are tailored to the player's performance and the game's difficulty level. Furthermore, the method used in this thesis builds upon existing literature on dynamic difficulty adjustment and procedural generation techniques. It extends existing approaches by incorporating a novel approach of using generative grammars to generate NPCs with varying attributes, actions, and affixes. This use of generative grammars allows for the creation of NPCs that are highly adaptable to the player's play style and can provide a challenging and engaging gameplay experience for them. Overall, the novelty of this thesis lies in its innovative approach to NPC generation and its potential to enhance the player experience through dynamic difficulty adjustment.

This thesis is divided into eight sections, each covering an important aspect of the research. In Section 1, I explored the role of flow state in video games, the concept of immersion in role-playing games, and explained the novelty of this thesis. In Section 2, I will provide a review of the current literature and justify the use of generative grammars. Section 3 outlines the methodology I used to complete this thesis, including the main research question and supporting questions. Section 4 focuses on the game used, its changes, and the generative grammar developed. In Section 5, I evaluate the grammar by analyzing the logs collected during gameplay and the generated content. Section 6 is dedicated to answering the supporting research questions. In Section 7, I reflect on the thesis and discuss potential future updates. Finally, in Section 8, I provide a conclusion to the main research question.

[3]`https://diablo.fandom.com/wiki/Elite_Monsters`

2 Literature Review & Justification

According to Zohaib [54] current literature shows that adapting the difficulty of a game can be done through the adaptation of the AI, using a heuristic or through generation of easier/harder levels based on the player's abilities, but it does not look into how the generation of NPCs with different strong and weak points can affect the difficulty of the game. In this section, I will review the current literature and give a justification on why I chose the goal described previously.

2.1 Dynamic Difficulty Adjustment

Dynamic Difficulty Adjustment (DDA) is the process of automatically changing the parameters, scenarios or behaviour of video games in real-time, such that it would match the player's skills. This method is used to ensure that the player would not feel bored, when the game is too easy, or frustrated, when the game is too hard [54].

Spronck et al. uses a method to adapt the difficulty, which is called dynamic scripting [47]. Each NPC has a set of rules (possible actions) and each rule has a weight, which dictates the chance of the NPC using that rule. The weights of the rules are increased or decreased after the completion of a fight, based on the efficiency of them during the fight. While this method shows similarities to my proposed idea, the main difference resides in the fact that dynamic scripting would increase the chance of an action being used if it is effective while my objective is to leave the probability the same and adapt the power of the character based on the player's skills. Dynamic scripting changes the behaviour and could be a useful future extension, because the game used by Spronck is also a turn based RPG (Neverwinter Nights©).

A method used to bring DDA in fighting games is the Monte-Carlo Tree Search based AI. Demediuk et al. [15], proposed three solutions that adapt the AI during the fight based on the player's abilities. The three methods used resulted in an AI which was either acting in an unrealistic way when the player had the advantage or it would need a certain type of opponent to work. The solution presented by Ishihara et al. [26] builds on top of the solution of Demediuk et al. [15] and introduces a believability parameter to fix the unrealistic behaviour of the AI. These methods also show how the difficulty can be changed by adapting the behaviour of the enemy and they were used for fighting games. However, compared to Spronck et al. [47], for these methods to be considered as a future update, they first have to be modified so they work for turn-based RPGs.

Another method presents the use of probabilistic graphs to model the player's progress. The model is two dimensional, which means that from the current node the player is on the graph there are only two possible transitions, the player either remains in the same state or they transition to the next one [53]. The assumption taken is that increasing the number of transitions indicates that the player is engaged. While this system was tested for a match three game and uses a probabilistic graph, the idea of the system was used when designing the difficulty decrease method. The idea of increasing the probability to transition to the next level was used when designing the difficulty decrease method. In my system the more the player would spend time repeating the same level, the weaker the enemies would become.

A heuristic method is used in [21], where the probability to die at the current state is calculated. If the probability is high, then the game becomes easier for the player. Conversely, if the probability is lower it becomes harder. This method was not used when designing the DDA system, because in a turn-based RPG, the player can heal at any time, while in the game they use the amount of heals are more limited, so the probability of dying would be harder to calculate. Furthermore, in my game, the difficulty would change after the player loses, not during the level.

Hamlet is an interactive DDA system which adjusts core inventory mechanics and the game economy if the player is outside their comfort zone [25]. Hamlet was not considered as it does not change the enemies, but the availability of items for the player. Besides, my game does not have items, but can be useful as a future improvement once items are added into the core game loop.

Markov Networks, together with orthogonally evolved AI were used to control and train agents to adapt the difficulty of a predator and prey game [23]. The agents are either friendly or hostile and the difficulty of the game changes by using combinations of friendly or hostile NPCs at different stages in their evolution. This method introduces the existence of friendly NPCs which do not exist in the game I use, but can be considered for future updates as turn-based RPGs can have friendly NPCs which help the player during play. This method adapts the difficulty by changing the behaviour of NPCs, which could be a useful future extension.

Another method uses Hidden Markov Models which create a player model, by collecting player's logs [8]. These logs contain information like movement, quest details, dialogue choices, interactions with the environment etc. The created model can be used to adapt the difficulty of the game to the player type. The use of player's logs, to determine their play style was used for this system as well, in order to reinforce the idea that enemies would adapt to the player, thus the player having to change their strategy.

Finally, procedural generation of the level was used in order to adapt the difficulty of the game [27, 52, 51]. The first method creates a model of the player, through a series of demo runs and questionnaires, and uses a ranking state vector machine during gameplay to determine the model of the player and generate the next chunk of the environment [27]. The second one uses agents and Interactive Evolutionary Computation to create the parts of the environment, while the final one calculates the difficulty penalty of the current chunk of the environment. Based on that, both methods adapt the difficulty by adding a new chunk that has either more or less obstacles [52, 51]. These methods adapt the game by changing the environment, so they were not used when designing the DDA system, as the main goal was to adapt how strong the enemies are. However, there are games that have weather or environment that influence the characters. For example, in the game Pokémon the weather[4] can influence a battle by strengthening some monsters, while weakening others. This means that there is room for DDA through environment generation for turn-based RPGs which have mechanics similar to the weather effects in Pokémon.

2.2 Procedural Content Generation

Procedural content generation (PCG) is a method of creating data in an algorithmic way, through the combination of assets created by humans and algorithms, coupled with computer-generated randomness [44]. There are several methods which are used for procedurally generating the content, like:

- Cellular automata [20]
- Markov Networks and Random Markov Fields [33]
- Genetic algorithms [40]
- Machine learning [28]
- Grammar [14]

PCG has also been shown to be a great method for creating levels in games such as Minecraft [28], Mega-Man [33] or Super Mario Bros [48]. Grammars, or formal grammars, are a set of rules which dictate how to form sentences using the alphabet of the language, such that the results are valid according to the syntax [34]. The grammars used for PCG are also known as *generative* grammars, which define a set of rules that generate combinations of words that form grammatical sentences in a given language. Grammars are distinguished by their expressive power, ranging from very limited but fast and provable regular grammars, to context-free and beyond, with the last level as expressive as a universal Turing machine [11].

One of the aims of the project is to generate NPCs data using generative grammars. Several papers talk about the use of PCG to generate characters or character attributes. Ruela and Guimarães [40], present the use of genetic algorithms to generate the troops for the enemy NPC heroes as well as their rewards. The generation of rewards can be considered as a future improvement. Another method presents the generation of a character's background by assigning random values to fields like age, date of birth etc [32]. The rest of the information is generated through a heuristic defined by the author and the user's input. This method shows that attributes, like age and birth date, are dependent on one another and there should be a correlation between those during generation. This method was taken into consideration, because there are attributes that can empower the affixes or moves of a NPC. Anagnoste [3] presents three PCG methods to generate the skill tree of player: naive random

[4]https://bulbapedia.bulbagarden.net/wiki/Weather

generation, L-system grammars and graph grammars. These techniques show that grammars can be used to create complex structures that are able to give character depth.

Other papers that tackle the use of PCG in video games show how the method can be used for creating environments. Using guidelines from already existing games, PCG is used to generate a map specific for a MOBA (Multiplayer Online Battle Arena) game [10]. Another method uses a combination of context free grammars and cellular automata-inspired process to generate simple dungeon style level layouts that are always playable [20]. Finally, Generative Adversarial Networks and Generative Graph grammars were used to generate a complete dungeon layout for the game Legend of Zelda [22]. These methods show the power of PCG when generating the environment by also employing the use of generative grammars and will be used as possible examples when defining the generative grammar.

Finally, PCG was also used for the generation of missions, rules, mechanics or narrative of the game. Graph grammars were used to generate the missions of a level, by defining a graph with nodes for entrance, tasks, goals, locks and keys [44, §5]. For the generation of rules, evolutionary algorithms were used as well as the encoding of the game rules [44, §6]. After encoding the rules of the game, they are set as the initial population of an evolutionary algorithm. Ludii and Angelina are two existing and successful systems that were used to generate board and arcade games, respectively [7, 13]. Planning algorithms are used for the generation of stories in games, which find a path from a given starting state to a given goal state. Moreover, a domain model of the game world is added so the planner algorithm can generate the complete plan of the narrative [44, §7]. These methods show that grammars can be used to also generate other mechanics in games, not only environments. The grammars will be used as possible examples and, as mentioned before, the generation of environment will be considered as a future improvement if the game supports environments that influence the combat as a mechanic.

2.3 The benefits of PCG in DDA

As presented in Section (2.1), most of the papers show the use of an artificial intelligence algorithm or other stochastic techniques. These methods are used to either change the behaviour of the computer-controlled NPCs or give other seeds of a puzzle or change the environment based on the player's progress. PCG and DDA have been shown to be able to work together, to adapt the difficulty of the game. Polymorph is a system which adapts the level, by creating a model of the player, through a series of demo runs and questionnaires, and uses a ranking state vector machine during gameplay to determine the model and generate the next chunk of the environment [27]. Another DDA system uses an artificial neural network to generate content for arcade games based on the player's skills, using an artificial neural network [42]. Finally, DDA together with PCG were used to adapt the game difficulty to the rehabilitation goals of patients with injured legs [16]. These methods will not be considered, as they adapt the difficulty by generating new pieces of the environment and do not use generative grammars. One method that uses grammars and updates the strength of an enemy is used in the game Mega-Man to generate bosses with complex attack patterns [9]. This method shows that DDA can be combined with PCG to adapt the AI by giving the boss complex attack patterns. This method is different from the system I propose, because it adapts the difficulty by changing the attack patterns and not by changing the stats, moves and affixes of the enemies.

In the video game industry, most games use static difficulty, but this cannot provide a complete immersive experience for all types of players, as mentioned in Section (1). Shadow of Mordor© and the sequel Shadow of War©, while still having static difficulty adjustment, introduced the Nemesis System [39], which would generate NPCs, such that they are unique. Moreover, the elite NPCs had different affixes, strong and weak points that made them feel unique, giving the player the feel that they would have to learn how to tackle them.

With this idea in mind, I want to use the evolution of enemies by using attributes and affixes to increase the level of difficulty. At the same time, this method should make the player feel like they have to tackle the NPCs in more inventive ways, instead of repeating the same pattern of attacks. To change the behaviour of an AI, the change should come from re-building the NPC, by giving them new attributes and affixes. This can be done using PCG and based on the method used this would need, to a certain extent, manual intervention.

As shown in Section (2.2) there are several algorithms that can be used to generate environments, characters or other game assets in a procedural way, but not all of them are completely suited for the generation of attributes and affixes of NPCs.

- **Cellular Automata**
  A cellular automaton consists of a grid where each cell has a state. Each cell also has a set of neighbours whose state is relative to the specific cell. While this method has been used to generate environments [20], the use of a grid is not as suited for generating a structured set of affixes or attributes as other methods. The reason is that each cell in the grid represents a state, but attributes are not states, they are values. There can exist a state for each value which can result in the need of defining too many states and that is out of scope. Moreover, since each cell would correspond to an attribute, action or affix, then the set of states for each cell can differ and this would go against the idea of a cellular automata.

- **Genetic Algorithms**
  This method has already been used for both DDA [23] and PCG [40], but it was not used to generate the attributes and affixes of characters. This method can be suited for it, and to adapt the difficulty NPCs generated at different stages should be picked and be put against the player, which also matches how the characters will be generated after every encounter. However, there are multiple types of strategies a player can use as well as multiple types of players, so having to create a player to be used against the enemy for each strategy and type is out of scope.

- **Machine Learning**
  While machine learning was used for both DDA [23] and PCG [29], for it to be successful a well-trained agent is needed. For this thesis, it would mean that for every possible player type every type of enemy NPC needs to be given as input for training. Initially, this method can be time consuming because the types of enemies suited for each player need to also be balanced such that the player can reach the flow state. Moreover, the generated NPCs might not be very accurate. Hence, human intervention is needed to pick the generated NPCs that are accurate for the current difficulty of the game.

- **Markov Networks & Random Markov Fields**
  This method was shown to work on both DDA [23] and PCG [45]. The generation was shown to be done fast and can create aesthetically varied maps, but it is unable to capture long range dependencies. In an RPG, attributes, actions and affixes are interdependent, which means that the inability to capture long range dependencies can be a detriment during generation.

- **Grammars**
  This method was used for both DDA [19] and PCG [20, 3]. It was shown to be able to create complex structures that are able to give character depth as well as environments and levels of a game. Moreover, it gives the most control on what needs to be generated and through the rules defined it can be ensured that terminal nodes, which could represent mandatory affixes or attributes, are picked [20]. One disadvantage of it is the need to design the grammar and define its rules. In case the rules are not properly defined or do not produce the wanted results, they need to be changed by the person who designed them.

A strong point of grammar-based generation is having the most control on what needs to be generated, and out of all the possible methods presented to generate it has the least impactful weaknesses. This is backed by Gellel and Sweetser [20], who say that grammars are able to offer a rigid structure which maintains a natural hierarchy.

As explained in Section (1), the player and the enemy NPCs in an RPG can be seen as a structured list of attributes, actions and affixes. Generative grammars are able to provide a structured way of generating sentences through the definition of rules, which can provide a lot of control on what is generated. By having the affixes, actions and attributes as terminal rules, we ensure that they are enforced on the creation of the character, thus being closer to accomplishing the idea of having NPCs evolve. In addition, the use of grammars seems the perfect fit for this research, because there are attributes, actions and affixes that can have an interdependence relation, which makes it a better candidate than Markov networks or cellular automata. Through the definition of a hierarchy, the final result does not need human intervention to make sure whether it is accurate or not. Human intervention is still needed when designing the rules for generating the NPCs, but this intervention is minimal compared to using machine learning or genetic algorithms which would meet continuous intervention until the wanted result is achieved. Given the technical benefits of grammars compared to the other methods, I propose the use of generative grammars for the DDA system. Beside the use of a generative grammar, I will also introduce new augments for it to better adapt the NPCs to the player's skills. These will be explained in Section 4, together with the complete grammar that was used to generate the NPCs.

In this section, I discussed the current state of research on DDA and PCG. Most of the literature on DDA focuses on adjusting different gameplay parameters to make the game easier or harder [53, 21, 25, 27, 52, 51]. Some literature also shows how the difficulty can be adjusted by altering the behaviour of NPCs [47, 9, 26, 15]. In contrast, the system I propose contributes to the existing DDA literature by adapting NPCs through the generation of their attributes, actions, and affixes. While this method can be likened to the methods used to adapt the behaviour, it stands out by not changing how the NPC behaves, but by making it stronger or weaker based on their attributes, actions and affixes. The existing literature has already demonstrated that PCG using generative grammars can be used successfully in conjunction with DDA [27, 52, 51]. Furthermore, PCG was also used to adapt the difficulty of games by modifying the NPCs, as seen in the work of Butler et al. [9]. For the generation of NPCs, the addition of new augments for the grammar represent my contribution to the methods that were already used successfully.

3 Methodology

As briefly mentioned in Section (1), the intent of this thesis is to see whether the implemented DDA system can be engaging for the player, as well as push the player into adapting to different situations and not use the same tactics. The system will generate attributes, actions and affixes for NPCs in a role-playing game, using generative grammars. It will also adapt the difficulty of the game, manipulating the magnitudes of the generated attributes, actions and affixes, by tracking the performance of the player in the previous encounters. Through the generation of new enemy NPCs, a level of variety can be brought for each encounter. This can be beneficial for the player's need for novelty and variety [4].

In order to fulfil the objective of this thesis, I propose the following research question that needs to be answered:

What is the correlation between the engagement of the player and the variety added by using DDA and PCG?

To get a better understanding on how this will be answered and the data that will be collected, two supporting questions need to be answered before providing a conclusion.

1. *Are players more engaged by a game using DDA than the same game using static difficulty?*
   To answer this question, I am going to conduct playtest sessions where participants will play both the static difficulty version of the game and the DDA version. The players will play the games in alternating order. For example, if the first participant will first play the static difficulty version of the game and then the dynamic difficulty version, then the second one will first play the dynamic difficulty version and then the static one and then the third one in the same order as the first and so on. Moreover, there are two types of participants that I will want to have as playtesters: people that have previous experience with turn based RPGs (i.e. Pokémon, Final Fantasy, Persona) and those that have no experience with those games.

   D. Dziedzic and W. Włodarczyk [18] show three different methods of measuring DDA: the use of a formal model of the game, the use of selected game parameters or direct examination of the player. I will use the direct examination of the player, which means that after playing a version the player will be asked to fill in a survey. The survey used will be the Player Experience Inventory (PXI) [1]. From this questionnaire I will only use certain constructs: *meaning, curiosity, mastery, immersion, challenge, enjoyment.* The reason is that the DDA system will be integrated in the combat system of the game, so constructs like *ease of control* or *audiovisual feedback* will be the same for both versions of the game. Beside this questionnaire, I will observe the player's behaviour during play to see their reaction to each encounter and after the participant has played both versions I will ask them about their perceived challenge for both versions, whether the DDA version felt unfair or not and which version they enjoyed more.

   For this question, I expect two scenarios to happen for both types of participants. In the first one, there is an increase in the perceived challenge for the DDA version of the game, compared to the static one, as well as an increase in the enjoyment, curiosity and immersion. This means that the DDA system is able to better maintain the player's attention and also provide a beneficial experience. In the second scenario, I expect there to be no difference between the mentioned constructs, which means that the DDA version doesn't stand out with anything compared to the static one.

2. *How does the variety of actions used by a player change during a playthrough?*
   During the playtest sessions I will collect player logs which will be used to answer this question. These logs contain the actions used by a player during each enemy wave, as well as the enemy NPCs' actions. From these logs I will extract how many actions were used for each wave as well as how varied they are. The scope of the framework is to generate NPCs which should be tackled in different ways and not the same way every time, thereby the variety of actions should increase over time.

   By default, there is a natural increase in the game's difficulty and variety of actions needed. By comparing the logs from the static difficulty and the DDA environments it will be possible to see whether the variety of actions can go beyond the natural scaling of the game. I expect two scenarios to happen when measuring the variety:

- In the first scenario, the variety when using the DDA framework is always bigger than the natural variety. This would mean that the player is quite experienced and does not struggle completing each level and the difference in variety is caused by the difficulty difference between the two versions. The increase in variety for this scenario can be seen in Figure 1.
- In the second scenario, the variety when using the DDA framework is also bigger but it fluctuates throughout the levels. This means that the difficulty increases until the player gets overwhelmed. After that it decreases to adapt to the player and finally, increases again until the player is overwhelmed again. The change in variety for this scenario can be seen in Figure 2.

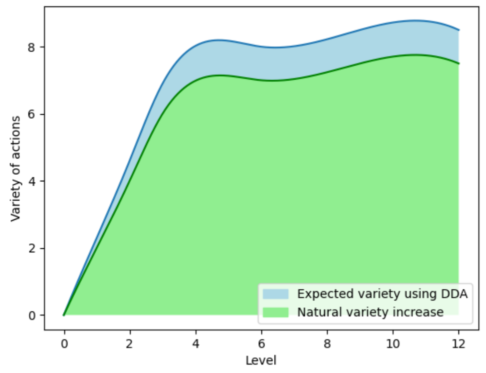


Figure 1: Expected variety when difficulty only increases

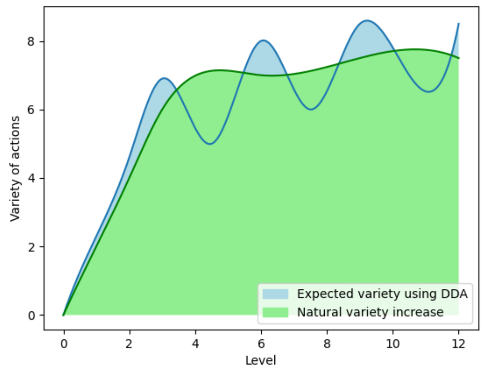


Figure 2: Expected variety when difficulty has to decrease

After collecting the measurements and providing an answer to these two questions, I will compare the results and conclude whether there is a positive correlation between the engagement of the player and the variety added through the use of DDA and PCG. To ensure the success of the thesis, the following steps will be taken:

1. Create game environment
   The game environment will be created using *Unity*, as it provides the necessary tools and libraries for creating a game really fast.
2. Design and implement grammar
   The grammar will be implemented using *C#*, as it is the main language used for *Unity*, so there will be no issues with integrating the grammar into the game environment.
3. Connect game to grammar
4. Perform play test sessions with candidates
5. Interpret the answers from each survey
6. Provide an answer to the secondary research questions
7. Provide a conclusion to the main research question

4 Design

In this section the following will be presented: the game design, which includes the chosen game and how it was adapted for the addition of the grammar, the grammar design, which includes the parsing, augments, how the NPCs are generated from the grammar, assumptions taken, and how the grammar connects to the game and how the difficulty changes.

4.1 Game

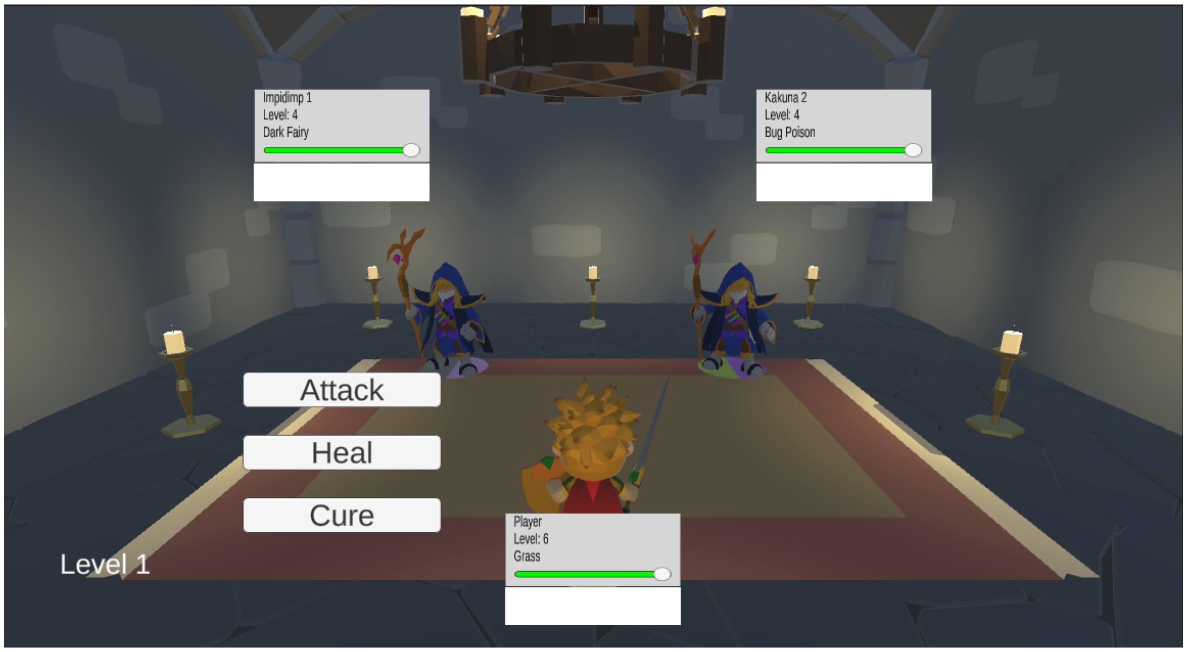


Figure 3: Image of the game environment

The chosen game to help as an environment for testing the grammar is Pokémon. Pokémon (also known as Pocket Monsters) is a turn based role-playing game, where the player takes the mantle of a trainer who can catch monsters to build their own team and become the best trainer, by earning badges and becoming the monster trainer champion. It is one of the biggest media franchises[5] which spanned 9 main generations of games[6] (each generation has usually 2 or 3 games, which are similar, the main difference being the available monsters the player can catch) accompanied by a world championship[7], an anime series and a trading card game. Furthermore, the long existence of this series and amount of tournaments that were organised is a sign that the combat is already balanced. This is the main reason for choosing it, as the aim is to not create a new game but a DDA system. Furthermore, there already exists an open-source project, called Pokémon Showdown, which contains all the moves, monster stats and other data that would ease setting up the environment[8]. In Pokémon, the fights happen between monsters, so the DDA system will update the difficulty after each fight that was either lost or won. The difficulty is defined by the total amount of stats a monster has. Because of this, the structure of the monster needs to be understood before the grammar is defined. The complete structure of a monster can be seen in Figure (4) and they contain the following data:

- Base stats: These include health points, attack, defence, special attack, special defence and speed. These stats influence the order in which the enemies attack, their resistance when attacked and how much damage they would deal. They do not represent the actual stats the monster has, but are more of an indication on how powerful the monster can be at maximum level. After each

[5]`https://www.titlemax.com/discovery-center/money-finance/the-25-highest-grossing-media-franchises-of-all-time/`
[6]`https://bulbapedia.bulbagarden.net/wiki/Generation`
[7]`https://worlds.pokemon.com/en-us/`
[8]`https://github.com/smogon/Pokemon-Showdown`

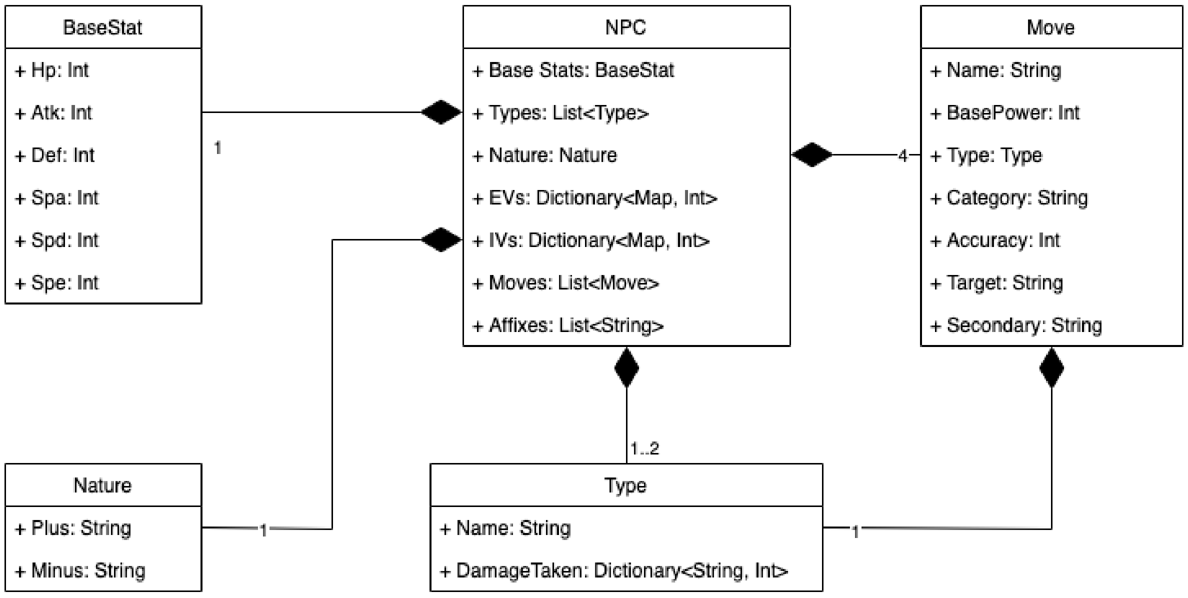


Figure 4: NPC Structure

monster level increases by one, also the value of the actual stats increase, based on the following formulae.

$$HP = \left\lfloor \left(2 \cdot Base + IV + \left\lfloor \frac{EV}{4} \right\rfloor \right) \cdot \frac{Level}{100} \right\rfloor + Level + 10 \tag{1}$$

$$Stat = \left\lfloor \left(\left\lfloor \left(2 \cdot Base + IV + \left\lfloor \frac{EV}{4} \right\rfloor \right) \cdot \frac{Level}{100} \right\rfloor + 5 \right) \cdot Nature \right\rfloor \tag{2}$$

- Type: In Pokémon, each monster has one or two types that influence whether attacks against them are stronger or weaker. For example, a water attack will be stronger against a fire type monster than a normal type one. There are 16 types in total and each type and which types they are weak and strong against can be seen in Appendix A, Figure 9.
- Nature: The nature adds an additional stat growth rate for when a monster levels up. It only influences two stats, one positively with a rate of 1.1 and one negatively with a rate of 0.9. Every other stat is not influenced. The complete list of natures and the stats they influence can be seen in Appendix A, Figure 10.
- Effort Value (EV): The effort value is an additional value between 0 and 252 which can be added to each of the base stats of a monster. These influence the actual value of the stats as it can be seen in formulae (1) and (2). One limitation of the EVs is the fact that only at most 510 EV points can be given to each monster.
- Moves: These are the attacks that a monster can use, and each monster can have only 4 moves. Every move has a name, a type, a category (special attack, normal attack and status affecting attack), a target type (for this system the moves are limited to only those that can affect oneself or the targeted enemy) and a possible secondary effect (here the moves were limited to only those that can sleep, paralyze, burn or decrease or increase the stats of the target). When a monster attacks, if the type of the move used matches the type it has, then a *Same-Type Attack Bonus* (STAB) is added to the power of the attack.
- Abilities: These are affixes which offer a passive boon and are limited to only one per monster. For this system, only 10 of them were picked and, to help with the idea that the enemy would adapt to the play style of the player, they are limited to two per monster.
- Individual Value (IV): These are values similar to the EVs. They are random numbers between 0 and 31, and have no other limitations. The main design idea behind these was that each monster is born with a certain innate potential, and that is why they are not generated by the grammar.

4.1.1 Changes to the Base Game

In Pokémon, battles happen between the player's monster and one enemy monster at a time. The player controls a team of up to 6 monsters and can swap them at any time during the fight. The swap mechanic was removed, because the player would have to first create their own team. This would imply that they have some previous knowledge of Pokémon and it is out of scope, since the main system that was tested is the combat. However, the player still needs one starting monster to fight, so I gave them the option to choose a starting one, with predefined moves. Furthermore, I presumed that by using the same monster, the player is going to get bored and, in addition to this, the grammar is going to generate mostly enemies that would have an advantage against the player. To solve these possible issues, I added the option for the player to change one of their moves, one of their types and change the values of their base stats every time they would lose or win a level.

In Pokémon, the player is able to use items during combat, which can either heal or remove status ailments (i.e. burning, poisoning, paralysis). For this prototype I decided to remove items and instead replace them with two possible actions the player can do, *heal* and *cure*. *Heal* action replaces the items that would restore health for the monster during combat, while *cure* replaces the items that would remove status ailments.

Another change made to the combat system was the switch from fighting one enemy to fighting two enemies. While in Pokémon it is common to only have 1 Vs 1 battles, in other turn-based RPGs like Final Fantasy©, there would be fights where the player is outnumbered. Because of this, I wanted to have the battles as 1 Vs 2, the player would fight 2 enemies at a time. The reason was to reinforce the idea that the player should adapt their play style to the generated enemies. In addition, this can provide the starting point for a future update where the amount of enemies could also be used to adapt the difficulty. The change to 1 Vs 2 enemies also came with its own difficulties, as the game was balanced around 1 Vs 1 battles. To balance the game I decided to set a level difference between the player and the enemies, which was done through multiple play tests done manually, without using the DDA system. In the end, I found that a level difference of two between the player and enemies was the point where the combat would feel slightly challenging but not unfair.

The final change made to the game is the possibility of collecting logs. There are two types of logs that are collected by the game. The first type is collected during the start and end of a level and is used for generating new NPCs. These logs contain the types, base stats and types of moves used by both the player and the NPCs. The second type of logs are used for answering the research question *How does the variety of actions used by a player change during a playthrough?*. These contain the moves a player has used for each level played and the difficulty of the game at each level, so it can be seen how the difficulty changes over time.

4.2 Grammar design

As stated in Section (3), one of the objectives of this research is to design a grammar to generate NPCs for turn-based RPGs. A formal grammar is a set of rules that describe how to create valid strings, according to the language's syntax, from a language's alphabet [34]. There are different types of grammars, like context-free grammars, regular grammars or attribute grammars. Grammars are distinguished by their expressive power, ranging from very limited but fast and provable regular grammars, to context-free and beyond, with the last level as expressive as a universal Turing machine [11]. When designing it, the starting point was an attribute grammar [2], to help with generating enemies more suited to the play style of the player. An attribute grammar is a formal grammar augmented with semantic information processing. The information is stored in attributes associated with terminal or nonterminal symbols. One benefit of using attributes is the fact that attributes allow the transfer of information between each node of the abstract syntax tree in a formal and controlled way. This means that at any point during generation the attributes can be called and used. This is important for generation, because these can be used to influence the generation of stats, moves or other elements of an NPCs which change the difficulty of a level.

However, an attribute grammar is not enough to fulfil the objective of this thesis. The NPCs have to adapt to the player's play-style and some stats are directly tied to the player's stats. In addition, not all moves or affixes can be used against a player. For example, if the player has a higher *special defence* than *physical defence*, then only having *special attack* moves will not be effective against the player. At the same time, only having moves that would be very effective against the player will make the player feel like the game is cheating. Another possible situation is when the generated enemies are weaker than the player, because their moves, stats or affixes are less effective against the player. The solution for this problem is the addition of augments to the grammar. Beside the addition of augments, the structure of the grammar was also updated to ease the parsing and other design decisions were taken to ensure the scope of the generative grammar matches the game used. All these additions will be explained in the following two subsections.

4.2.1 Grammar Structure for Parsing

Before the generation of NPCs, the grammar rules have to be parsed and transformed into an abstract syntax tree. In theory grammar rules are written quite straightforward, as it can be seen in example (3). While this structure is easy to read, several decisions were taken in order to ease the parsing.

$$\begin{array}{lll} 1. & S & \rightarrow aA \\ 2. & A & \rightarrow aA \\ 3. & A & \rightarrow bB \\ 4. & B & \rightarrow b \mid B \\ 5. & B & \rightarrow \epsilon \end{array} \tag{3}$$

First, a tilde ($\sim$) was used to indicate that two symbols separated by it will both be expanded. For example, the following rule $A := B \sim C$ means that from rule A we expand to both rules B and C. This was added to ease the parsing of rules. When the left-hand side of a rule is split into tokens, by first splitting at tilde the following rules are extracted. However, by splitting at spaces, if attributes are used then each token would have to be verified first to see whether it represents a rule, an attribute or another symbol that is used for a condition or mathematical equation. In the case, where only one of the two nodes will have to be expanded, a bar (|) is used, which is the standard in literature [31]. If rule A looks like this $A := B \mid C$, then it means that only one rule has to be picked between B and C. With this in mind the rules in example (3) would look the following way:

$$\begin{array}{lll} 1. & S & \rightarrow a \sim A \\ 2. & A & \rightarrow a \sim A \\ 3. & A & \rightarrow b \sim B \\ 4. & B & \rightarrow b \mid B \\ 5. & B & \rightarrow \epsilon \end{array} \tag{4}$$

Beside rules that expand to other rules, there are also rules that generate a number. These are written as intervals as it can be seen in rule (5). Here, A is a rule, while B and C are integers, $B < C$. The squared brackets are used to indicate that both B and C are included when picking a number and the two dots (..) are used to show that the number ranges from B to C.

$$A := [B \,..\, C] \tag{5}$$

Because the starting point is an attribute grammar, every rule can have attributes attached to the rules it expands to, separated by a colon. For example:

$$\begin{array}{lll} A & := & B \,:\, b \leftarrow \mathit{1} \,\sim\, C \,:\, c \leftarrow 2 \\ A & := & B \,\sim\, C \,:\, b \leftarrow \mathit{1},\; c \leftarrow 2 \end{array}$$

Attributes can be written after every rule and in any order and if there are more attributes written they have to be separated by a comma. Also, if the rule is of the form *A := B / C* and attributes are set only on the side of B then if C is picked they will not be taken into consideration. There are cases when an attribute can take multiple values, based on the logs of the player. To adapt to this case, support for inline *if statements* was added and the values of the attributes are chosen during the generation.

$$A \;:=\; B \,:\, \mathit{condition} \,?\, b \leftarrow 1 \,:\, b \leftarrow 2$$

Besides rules, conditions or other augments can be written as simple mathematical equations, which also have to be parsed. These equations can contain attributes and their result is either a boolean or integer. To parse them, first the string with the equation is split at spaces into a list of tokens. If a token is an operand or constant then it is left, otherwise it is replaced with the value of the attribute. The result of the equation is calculated using the *DataTable* data type, which mimics how a spreadsheet works. The equation is inserted into a cell in the table and the result is calculated automatically.

The parsing of rules that contain attributes, weights and other augments can come with some intricacies and in order to keep the scope of the grammar in check, several assumptions have been made. The first one is that rules that have the source augment are either terminal or nonterminal and all the following rules are terminal. The second assumption is related to the chosen game. Because Pokémon is used as a base game, the only supported data types are int, boolean and string, because, as it can be seen in formulae (1) and (2), the result of every division is rounded down. The only supported functions are MIN, MAX, SIZE and DISTINCT. *TYPE.DamageTaken* function is a custom method and is hard coded, because of the difficulty of implementation in a general way. If $\implies$ (imply) is used in a condition, it is used only once and is always of the form $A \implies B$, where A and B are boolean equations that have no imply.

4.2.2 Augments

While the use of attributes can already help transfer information to every node of the abstract syntax tree, there are cases where they cannot influence the generation process itself. For example, a monster can have at most two types, which have to be different, so the following rules can be written:

$$\begin{aligned} NPC &:= TYPES \\ TYPES &:= TYPE \sim TYPE \mid TYPE \\ TYPE &:= \text{“}Fire\text{”} \mid \text{“}Grass\text{”} \end{aligned}$$

These rules do not show that the NPC can have two different types. In order to solve this issue, weights [41] were added as an augment to indicate the chance of picking one of the possible next rules. The values of the weights can also be taken from attributes used in previous rules, by just using the name of the attribute, instead of a value. By adding weights and attributes, the rules for generating types are going to look the following way:

$$\begin{aligned} NPC &:= TYPES \,:\, WT \leftarrow 1,\ WF \leftarrow 1,\ WG \leftarrow 1 \\ TYPES &:= [WT]\ TYPE \sim TYPES \,:\, WT \leftarrow 0 \mid [1]\ TYPE \\ TYPE &:= [WF]\ \text{“}Fire\text{”} \,:\, WF \leftarrow 0 \mid [WG]\ \text{“}Grass\text{”} \,:\, WG \leftarrow 0 \end{aligned}$$

In this case, we have the attribute *WT, WG* and *WF*, which are defined in the *NPC* rule. When the *TYPES* rule is reached the probability of picking the first alternative side is $\frac{WT}{WT+1}$, then WT will become 0. This means that it will not be picked a second time and the probability of picking the second alternative becomes 1 after first picking the first one. After *TYPES*, the *TYPE* rule is picked, and here, similar to the previous rule, based on the chosen type, its weight will become 0, meaning that there is not going to be a second type that is the same as the first one.

While weights can be useful when having to choose which rule to pick next, they can become cumbersome when encoding everything explicitly. In this case another augment was added: a *condition.* One of the roles of conditions is to ensure that the difficulty of the game is adjusted based on the logs of the player. In order to add a condition, or multiple conditions, to a rule, the *condition* tag has to be added under a rule and the conditions can be written after, separated by |, in case there are multiple ones. An example of a condition can be seen in rule (6), where there is the *condition* tag underneath it.

$$\begin{aligned} BASE &:= HP \sim ATK \sim DEF \sim SPA \sim SPD \sim SPE \qquad (6) \\ source &: \texttt{basestats.json} \\ condition &: LOGS.PlayerDefence = \text{”}Special\text{”} \implies ATK \geq SPA \\ HP &:= [MIN(EnemyStats.HP) + 1..195] \qquad (7) \end{aligned}$$

In this case, we have the *BASE* rule, which is used to generate the main stats of an NPC (health points, attack, defence, special attack, special defence and speed). All of the following rules are going to be expanded. In addition to this, to adapt the encounter, the condition to check if the player's main defence is *Special* (special attacks are weaker than physical attacks), then the base attack of the NPC should be bigger or equal than the base special attack was added. When writing conditions the following operations are supported: AND, OR, NOT, $=$, $\leq$, $<$, $\geq$, $>$, $\implies$ (IMPLY) and IN (used for checking if an attribute or constant is inside a list).

As mentioned in (4.1), the chosen game is Pokémon, which means that the moves, types or base stats of a monster are already predefined. In this case an augment was added to indicate a *source* for values to be picked from a file instead of being generated. This augment was added because there is more familiarity with already existing names and values for some players and generating values that fulfil a condition is harder than just picking it. The rules that can have the source tag, are only nonterminal rules, whose next rules are terminal or terminal rules, and are called source rules. The supported file type is only JSON. One example of a source rule is the *BASE* rule in rule (6). When the source is indicated for a rule, then every following rule will act as a filter on the items in the file. This means that each object in the JSON file has to have fields with the name of the symbols given in the rule. In the example shown above, the rule *BASE* has 6 symbols it can expand to, *HP*, *ATK*, *DEF*, *SPA*, *SPD* and *SPE*, which means that each object in the `basestats.json` file contains each of these symbols as fields. When the rules that follow from a source rule are used for picking a random number, then these act as filters for the values of objects' fields in the JSON file. One example can be seen in rule (7), which means only *base stats*, whose *HP* is between the minimum *HP* value of the previous two NPCs plus one and 195 can be picked. For the remaining stats the rules are the same as the one for *HP*. By applying this filter, it means that the new NPCs base stats are going to be bigger than those of the previous ones that were generated.

The final augment that was added allows the passing of parameters in the first rule of the grammar. This has been added to introduce the level logs in the process of generation as an attribute. At the moment the grammar supports only one parameter given, as it was out of scope to have multiple parameters. To pass the logs of the player to the grammar, the first rule needs to contain the name *LOGS* as a parameter, as it can be seen in (8).

$$\mathit{NPCS}(\mathit{LOGS}) := \mathit{NPC} \sim \mathit{NPC} \tag{8}$$

4.3 Proposed Grammar

Having the structure of the grammar explained, I will explain the proposed grammar and how the generation of NPCs works. The complete grammar is available under open source license on my GitHub repository[9].

$$\begin{aligned}
\mathit{NPCS}(\mathit{LOGS}) \;&:=\; \mathit{NPC} : \mathit{DomainAffix} \leftarrow 1, \mathit{LinkAffix} \leftarrow 1, \mathit{Weakness} \leftarrow 1,\\
&\; \mathit{LOGS.HasAilments}\ ?\ \mathit{Ailment} \leftarrow 2 : \mathit{Ailment} \leftarrow 1, \mathit{Heal} \leftarrow 0 \sim \mathit{NPC}\\
\mathit{NPC} \;&:=\; \mathit{TYPES} : \mathit{WT} \leftarrow \mathit{LOGS.CurrentLevel}, \mathit{WBug} \leftarrow 1, \mathit{WDark} \leftarrow 1,\\
&\; \mathit{WDragon} \leftarrow 1, \mathit{WElectric} \leftarrow 1, \mathit{WFairy} \leftarrow 1, \mathit{WFighting} \leftarrow 1,\\
&\; \mathit{WFire} \leftarrow 1, \mathit{WFlying} \leftarrow 1, \mathit{WGhost} \leftarrow 1, \mathit{WGrass} \leftarrow 1, \mathit{WGround} \leftarrow 1,\\
&\; \mathit{WIce} \leftarrow 1, \mathit{WNormal} \leftarrow 1, \mathit{WPoison} \leftarrow 1, \mathit{WPsychic} \leftarrow 1,\\
&\; \mathit{WRock} \leftarrow 1, \mathit{WSteel} \leftarrow 1, \mathit{WWater} \leftarrow 1 \sim \mathit{STATS} \sim \mathit{MOVES} \sim\\
&\; \mathit{AFFIXES} : \mathit{WA} \leftarrow \mathit{LOGS.CurrentLevel}
\end{aligned} \tag{9}$$

In the first two rules, two NPCs are created, with no stats or moves or types, and attributes which will be used as weights for some affixes and types are given an initial value. The attributes that will influence the number of types and affixes are given the value of the current level. While the number of types and affixes is not tied to the difficulty of the game, their number can influence what actions the player uses. Types are the first characteristic of the NPC that are generated. These are chosen from a file, `typechart.json`, as it can be seen in rules (10) (for simplicity I only included three types in the *TYPE* rule below, but all of them are similar). Each type has a weight of one and whenever one is picked, its weight becomes 0. Moreover, if the types the player used to attack the enemy in the

[9] `https://github.com/andreipopa90/Generative-Grammar-for-NPCs`

previous level is strong against that specific type, then the weight for *Reverse Weakness* is increased. This is used as a way for the NPC to adapt to the player's play style, by making them use moves with weak types instead of moves with strong types against the NPC. The types are taken from a source, because the json file for types also includes the types they are strong and weak against. Because of this, the weaknesses

$$\begin{aligned}
TYPES &:= [WT]TYPE \sim TYPES : WT \leftarrow 0 \mid [1]TYPE \\
TYPE &:= [WBug]\text{"}Bug\text{"} : WBug \mathrel{-}= 1, \\
&\quad TYPE.DamageTaken(LOGS.PlayerTypes) = 1 \;?\; Weakness \mathrel{+}= 1 \mid \\
&\quad [WDark]\text{"}Dark\text{"} : WDark \mathrel{-}= 1, \\
&\quad TYPE.DamageTaken(LOGS.PlayerTypes) = 1 \;?\; Weakness \mathrel{+}= 1 \mid \\
&\quad [WDragon]\text{"}Dragon\text{"} : WDragon \mathrel{-}= 1, \\
&\quad TYPE.DamageTaken(LOGS.PlayerTypes) = 1 \;?\; Weakness \mathrel{+}= 1 \mid \\
from &: \texttt{typechart.json}
\end{aligned} \tag{10}$$

Next step is the generation of base stats, nature and EV points. At first the base stats are generated, because their values are further used to pick the nature and EVs. The base stats are picked from the `basestats.json` file, because in Pokémon there is no maximum amount of stats, so it is easier to pick the predefined stats than just generate them. From rules (11) it can be seen that each stat should be bigger than the smallest stat of the previous NPCs, which ensures that the difficulty increases. Furthermore, if the player used more physical attacks than special ones, then the defence should be bigger or equal than the special defence otherwise the opposite should hold. The same applies for the type of defence the player has. If the player has more defence than special defence, then the NPC should have the special attack bigger or equal than the physical attack. While I mentioned that there is no maximum amount of stats for each NPC, the stat rules are written as intervals because they are applied as filters when picking the base stats.

$$\begin{aligned}
STATS &:= BASE \sim NATURE \sim EVS \\
BASE &:= HP \sim ATK \sim DEF \sim SPA \sim SPD \sim SPE \\
from &: \texttt{basestats.json} \\
condition &: LOGS.Playerdefence = \text{"}Special\text{"} \implies BASE.ATK \geq BASE.SPA \mid \\
&\quad LOGS.Playerdefence = \text{"}Physical\text{"} \implies BASE.SPA \geq BASE.ATK \mid \\
&\quad LOGS.PlayerAttack = \text{"}Special\text{"} \implies BASE.SPD \geq BASE.DEF \mid \\
&\quad LOGS.PlayerAttack = \text{"}Physical\text{"} \implies BASE.DEF \geq BASE.SPA \\
HP &:= [MIN(LOGS.EnemyStats.HP) + 1..195] \\
ATK &:= [MIN(LOGS.EnemyStats.ATK) + 1..195] \\
DEF &:= [MIN(LOGS.EnemyStats.DEF) + 1..195] \\
SPA &:= [MIN(LOGS.EnemyStats.SPA) + 1..195] \\
SPD &:= [MIN(LOGS.EnemyStats.SPD) + 1..195] \\
SPE &:= [MIN(LOGS.EnemyStats.SPE) + 1..195]
\end{aligned} \tag{11}$$

After the stats are picked, it is time to give the NPC a nature. The nature influences the growth of two stats at every level, as it can be seen in equations (1) and (2). One is influenced positively, named *plus*, and one negatively, named *minus*. Because one stat is influenced positively and one negatively, this does not actually influence the overall difficulty of the level, but each plus gets a weight the value of the stat it increases. This means that, on average, the highest stat will be influenced positively.

$$\begin{aligned} NATURE &:= PLUS \sim MINUS \\ from &: \texttt{natures.json} \\ PLUS &:= [BASE.ATK]''atk'' \mid [BASE.DEF]''def'' \mid [BASE.SPA]''spa'' \mid \\ &\quad\; [BASE.SPD]''spd'' \mid [BASE.SPE]''spe'' \\ MINUS &:= [1]''atk'' \mid [1]''def'' \mid [1]''spa'' \mid [1]''spd'' \mid [1]''spe'' \end{aligned} \tag{12}$$

Finally, the EV values are picked. Each EV can be at most 255 and their value influences a stat only if they are divisible by 4, as it can be seen in formulae (1) and (2), and their sum should be at most 510. The intervals in rules (13) were created to pick the values such that they are not bigger than 252, because 255 is not divisible by 4, and their sum is still 510. The minimum value of each EV can be 0, and the maximum possible value is the minimum between 252 and $510 - 4 \cdot \sum ev$, divided by 4. The reason I took the minimum, is because when the *ev* for *hp* is picked, then each *ev* is 0, because it is the first EV rule, so then the hp EV can take the maximum value, 252. I divide it by 4, because each EV should be a multiple of it and because the grammar only supports integers, then the result from the division ensures this. Later in the game when the actual stats are calculated, the EVs are multiplied by 4.

$$\begin{aligned} EVS &:= HPEV \sim ATKEV \sim DEFEV \sim SPAEV \sim SPDEV \sim SPEEV \\ HPEV &:= [0..\tfrac{MIN(252,510-4\cdot ATKEV-4\cdot DEFEV-4\cdot SPAEV-4\cdot SPDEV-4\cdot SPEEV)}{4}] \\ ATKEV &:= [0..\tfrac{MIN(252,510-4\cdot HPEV-4\cdot DEFEV-4\cdot SPAEV-4\cdot SPDEV-4\cdot SPEEV)}{4}] \\ DEFEV &:= [0..\tfrac{MIN(252,510-4\cdot ATKEV-4\cdot HPEV-4\cdot SPAEV-4\cdot SPDEV-4\cdot SPEEV)}{4}] \\ SPAEV &:= [0..\tfrac{MIN(252,510-4\cdot ATKEV-4\cdot DEFEV-4\cdot HPEV-4\cdot SPDEV-4\cdot SPEEV)}{4}] \\ SPDEV &:= [0..\tfrac{MIN(252,510-4\cdot ATKEV-4\cdot DEFEV-4\cdot SPAEV-4\cdot HPEV-4\cdot SPEEV)}{4}] \\ SPEEV &:= [0..\tfrac{MIN(252,510-4\cdot ATKEV-4\cdot DEFEV-4\cdot SPAEV-4\cdot SPDEV-4\cdot HPEV)}{4}] \end{aligned} \tag{13}$$

After the base stats, the EVs and a nature are picked for the NPC, it is time to pick its moves. For moves a file was used, `moves.json`, as it can be seen in rules (14). A monster can have at most 4 moves and whenever a move matches the type of it, then it receives a damage boost when attacking an enemy. Knowing this, a condition was added which indicates that the type of the move should be the same as the type of the NPC, unless the move is a move that influences the stats of a character. Moreover, the moves have to be distinct and there should be at least one move with a power bigger than 0, so at least one attacking move. Every move has a category, either *physical, special* or *status.* Physical moves use the attack stat to calculate their damage, while special moves use the special attack. Status moves are not influenced by any stat, but they can either increase or decrease the value of stats. Because of this, each move will have their weights of being picked based on the values of the player's defence, the defence for special moves, special defence for physical moves and the minimum of both for status moves. It was done this way to give a chance for the NPC to counter the player. I chose to give a chance to counter instead of always choosing the most suited moves to avoid the possibility of making the game unfair.

$$
\begin{aligned}
\mathit{MOVES} &:= \mathit{MOVE} \sim \mathit{MOVE} \sim \mathit{MOVE} \sim \mathit{MOVE} \\
\mathit{MOVE} &:= \mathit{CATEGORY} \sim \mathit{MOVETYPE} \sim \mathit{TARGET} \\
\mathit{from} &: \texttt{moves.json} \\
\mathit{condition} &: \mathit{MOVE.MOVETYPE\ IN\ TYPES\ OR} \\
&\quad \mathit{MOVE.TARGET} = \text{''}\mathit{self}\text{''}\ \mathit{OR\ MOVE.CATEGORY} = \text{''}\mathit{Status}\text{''} \mid \\
&\quad \mathit{DISTINCT}(\mathit{MOVES}) \mid \mathit{SUM}(\mathit{MOVES}, \text{''}\mathit{BASEPOWER}\text{''}) \geq 1 \\
\mathit{CATEGORY} &:= [\mathit{LOGS.PlayerStats.DEF}]\text{''}\mathit{Special}\text{''} \mid \\
&\quad [\mathit{MIN}(\mathit{LOGS.PlayerStats.SPD}, \mathit{LOGS.PlayerStats.DEF})]\text{''}\mathit{Status}\text{''} \mid \\
&\quad [\mathit{LOGS.PlayerStats.SPD}]\text{''}\mathit{Physical}\text{''} \\
\mathit{TARGET} &:= [1]\text{''}\mathit{self}\text{''} \mid [1]\text{''}\mathit{normal}\text{''} \\
\mathit{MOVETYPE} &:= [1]\text{''}\mathit{Bug}\text{''} \mid [1]\text{''}\mathit{Dark}\text{''} \mid [1]\text{''}\mathit{Dragon}\text{''} \mid [1]\text{''}\mathit{Electric}\text{''} \mid [1]\text{''}\mathit{Fairy}\text{''} \mid [1]\text{''}\mathit{Fighting}\text{''} \mid \\
&\quad [1]\text{''}\mathit{Fire}\text{''} \mid [1]\text{''}\mathit{Flying}\text{''} \mid [1]\text{''}\mathit{Ghost}\text{''} \mid [1]\text{''}\mathit{Grass}\text{''} \mid [1]\text{''}\mathit{Ground}\text{''} \mid [1]\text{''}\mathit{Ice}\text{''} \mid \\
&\quad [1]\text{''}\mathit{Normal}\text{''} \mid [1]\text{''}\mathit{Poison}\text{''} \mid [1]\text{''}\mathit{Psychic}\text{''} \mid [1]\text{''}\mathit{Rock}\text{''} \mid [1]\text{''}\mathit{Steel}\text{''} \mid [1]\text{''}\mathit{Water}\text{''}
\end{aligned}
\tag{14}
$$

Finally, the affixes are picked, and the rules for that can be seen below. Here, most of the attributes that are first initialized in the first rule are used as weights for some affixes. There can be at most two affixes, but there is also the chance for an NPC to have no affixes. The reason there is also an option for no affixes is because at early levels I wanted to give the feel that enemies with affixes are more rare, and their occurrence would increase once the level increases. Because of this, the weight for having two affixes is the same as the previous level.

$$
\begin{aligned}
\mathit{AFFIXES} &:= [1]\mathit{AFFIX} \mid [\mathit{WA}]\mathit{AFFIX} \sim \mathit{AFFIXES} : \mathit{WA} \leftarrow 0 \mid [1]\text{''}\mathit{NoAffixes}\text{''} \\
\mathit{condition} &: \mathit{DISTINCT}(\mathit{AFFIXES}) \\
\mathit{AFFIX} &:= [\mathit{LinkAffix}]\text{''}\mathit{HealthLink}\text{''} : \mathit{LinkAffix} > 1\ ?\ \mathit{LinkAffix} \leftarrow 1 : \mathit{LinkAffix} \mathrel{+}= 2 \mid \\
&\quad [\mathit{DomainAffix}]\text{''}\mathit{Domain}\text{''} : \mathit{DomainAffix} \mathrel{-}= 1 \mid [\mathit{Heal}]\text{''}\mathit{Medic}\text{''} : \mathit{Heal} \mathrel{-}= 1 \mid \\
&\quad [1]\text{''}\mathit{Avenger}\text{''} \mid [1]\text{''}\mathit{Evolve}\text{''} \mid [1]\text{''}\mathit{Sturdy}\text{''} \mid \\
&\quad [1]\text{''}\mathit{Lifesteal}\text{''} : \mathit{Heal} \mathrel{+}= 1 \mid [1]\text{''}\mathit{CounterAttack}\text{''} \mid \\
&\quad [\mathit{Weakness}]\text{''}\mathit{ReverseWeakness}\text{''} : \mathit{Weakness} > 1\ ?\ \mathit{Weakness} \mathrel{-}= 1 \mid \\
&\quad [\mathit{Ailment}]\text{''}\mathit{ReverseAilment}\text{''} : \mathit{Ailment} > 1\ ?\ \mathit{Ailment} \mathrel{-}= 1
\end{aligned}
\tag{15}
$$

4.4 Game and Grammar connection

Now that both the game and the generative grammar were explained, it is time to show how these two interact and how the DDA system actually works. This is going to be explained in the following sections.

4.4.1 Difficulty Increase

The difficulty increase happens during the enemy generation, when the player wins a level and it happens in the following order: first, the types of the monster are chosen using a source file, *typechart.json.* After this, the base stats are chosen using another source file, *basestats.json* and every base stat has to be bigger than the smallest base stats from previous enemies, as it can be seen in rule (11) and their defences and attacks needs to be suited to the main damage and defence of the player, either physical or special. At the next step, the nature is chosen and the EVs for each stat are picked. Next, the moves are picked from a source file, *moves.json.* If the player has a higher defence of a certain type, then the moves that would counter that, have a higher chance to be picked. For example, if the player's defence is bigger than their special defence, then special moves have a higher chance for getting picked. Moreover, the moves have to have the same type as the monster if they are damaging type moves. Finally, the abilities are picked. After the generation is finished, the enemies are stronger because they have higher stats than the previous enemies and are more resilient to the player's attacks

When generating new enemy NPCs, the grammar takes the logs of the previous level and uses data such as: the player's stats, player's moves and the previous enemies' stats. This ensures that the new enemies are more adapted to the play style of the player. One such way can be seen in the following example:

$$\begin{aligned}
HP &:= [MIN(LOGS.EnemyStats.HP) + 1..195] \\
ATK &:= [MIN(LOGS.EnemyStats.ATK) + 1..195] \\
DEF &:= [MIN(LOGS.EnemyStats.DEF) + 1..195] \\
SPA &:= [MIN(LOGS.EnemyStats.SPA) + 1..195] \\
SPD &:= [MIN(LOGS.EnemyStats.SPD) + 1..195] \\
SPE &:= [MIN(LOGS.EnemyStats.SPE) + 1..195]
\end{aligned}$$

Here the stats of the newly generated NPCs should be at least bigger by one than the weakest enemy of the previous level. The reason for choosing the starting point for the new stats to be slightly bigger than the minimum, is because otherwise the difficulty increase would be too steep and it would reach the strongest enemies too fast. Another way the grammar makes the NPC adapt to the player is done by the conditions in rules (11) and (14)

In rules (11), the grammar influences which attack and defence stats should be bigger, based on which dominant attack type the player uses and what type of defence the player has. In rules (14), the grammar will only pick moves that have the same type as the NPC, because during battle if the type of the move matches the type of the monster that used it, then the STAB will be added to the overall damage of the move. Furthermore, if the player has a higher defence than special defence for example, then the weight of picking a special type move is bigger.

4.4.2 Difficulty Decrease

As per the definition of DDA, whenever a player loses the difficulty of the game should decrease as well. Because the EVs and Base stats influence the overall strength of a NPC, then the difficulty of an encounter will be determined by the total sum of the EVs and Base Stats. Whenever the player loses one, at first the amount of EVs will be decreased, as they are usually used to further empower the monster and whenever all of them are 0 then Base Stats will be decreased. The formula for decreasing the stats is the following:

$$Decrease(losses) = \begin{cases} EVs - 10^{losses-1} & 0 \leq EVs \\ BaseStats - 10^{losses-1} & 1 \leq BaseStats \end{cases}$$

This formula was chosen after simulating the amounts of losses needed for the player until it can win the level, when fighting against 2 enemies and the player and enemies had the same base stats. The starting base chosen has the hp, attack, defence, special attack, special defence and speed values equal to 120 and all of them use only one move with a very high base power. These base stats and move were chosen because I consider this as the ideal scenario where the player and enemies reach the point where they are evenly matched. There were two scenarios put in place, based on the types of the enemies and the player. In the first one the player would be at a disadvantage because its type would be weak against the enemies move. In this case, if the player loses the level, besides decreasing the difficulty, then their type would be changed to something with no advantage or disadvantage against the type. I did it like this because I expect the player, when play testing the game, to also change their monster type when playing. In the second scenario, the player and enemies type would be the same and they would have no disadvantage against the move.

Two types of decreases of the stats were tested, the first one $x \times losses$ and the second one x^{losses}. The aim was to reach the point where the player would win in a minimum amount of tries, such that whenever the player might encounter this situation they do not lose interest in the game. In Figures (5) and (6) it can be seen that the two functions were tested with x ranging from 1 to 10. From those two figures it can be seen that whenever the enemy has no type advantage, then the amount of losses remains almost constant for both functions and there is almost no difference between the two of them. However, when there is a type advantage, by using x^{losses}, the number of losses is smaller the higher the x is than $x \times losses$. This implies that by using x^{losses}, then the player would undergo the least amount of retries before winning the level. I presume that this is better, because if the player would repeat the same level too many times, the risk of them getting bored would increase.

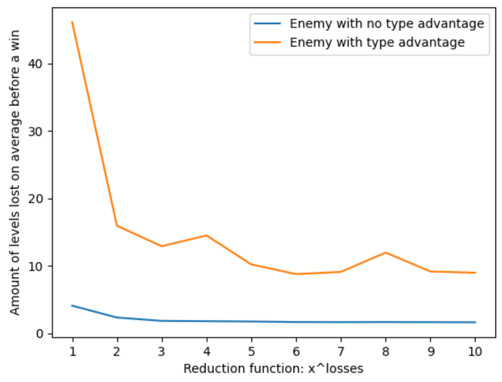


Figure 5: Average amount of losses when Base Stats and EVs are decreased with x^{losses}

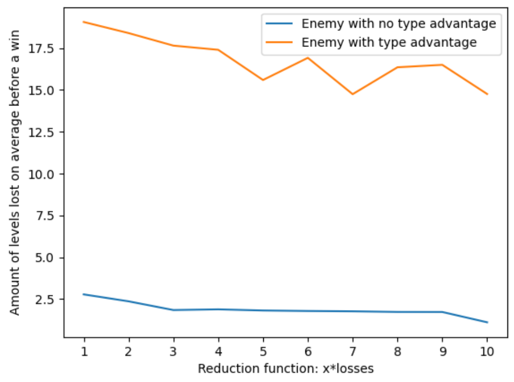


Figure 6: Average amount of losses when Base Stats and EVs are decreased with $x \times losses$

5 Grammar Evaluation

Before going into answering the research question, the grammar has to be evaluated. The aim of this section is to show what is generated by the grammar, by showing level log examples and what is generated based on them. I will explain how this influences the way the player tackles new encounters based on the generated NPCs.

To evaluate the generation of NPCs, I have created two scenarios. In each scenario the monster used by the player is one of the ones available to choose at the beginning of the game and each player's logs are collected at different levels. In the first scenario the logs are collected after the first level and in the second after level 10. I chose these levels to match how such a game usually has an early state, where enemies have mostly one type and the chance for them to have one or two affixes is the same as the chance to have no affixes, and the late state, where enemies have a bigger chance of having two types and two affixes.

5.1 First Scenario

The data contained within the logs for the first scenario is found in Table (1), where it can be seen that the player focused on using physical attacks and they are as resilient against special attacks as they are against physical attacks. In this case, when the two defence types are the same, the *Special* defence is picked and the same rule applies for *attack* and *special attack*. In addition, the moves they used caused no status ailments (i.e. poisoning, burning, paralysis). Also, the previous level from which the logs were collected was level one, which means that during the generation of enemies the chance of having two types is equal to having only one type, while the chances of having two affixes is equal to having only one or no affix.

	First Scenario	**Second Scenario**
Level	1	10
Player Attributes		
Types	Fire, Fighting	Water, Ground
HP	80	100
ATK	120	110
DEF	70	90
SPA	110	85
SPD	70	90
SPE	80	60
Main Defence Type	Special	Special
Main Attack Type	Physical	Physical
Inflicted Status Ailments	No	Yes
Enemies Stat values		
HP	30, 30	120, 126
ATK	56, 25	70, 131
DEF	35, 35	120, 95
SPA	25, 45	75, 131
SPD	35, 30	130, 98
SPE	72, 20	85, 99

Table 1: Logs collected for both scenarios

The enemies generated based on this data can be found in Table (2). In this table, it can be seen that both enemies have their attack higher or equal than their special attack and their special defence is higher or equal than their defence, which matches the condition in rules (11). The moves of the generated enemies also follow the conditions in rules (14), where the move has to have the same type of the NPC, unless it is a *Status* move. With these stats generated the player will be more vulnerable to physical attacks, which in turn will make them use status moves that would increase their defence. In addition, the player will observe that their physical moves used will be weaker this time, therefore pushing them to use special moves as well. The first enemy, *Leavany*, has the *Domain* affix, which makes all moves that match its type stronger as well as make the moves that would have an advantage against its type weaker. The player has the *Fighting* type which is effective against the *Dark* type of the first enemy (see Image (9)). The affix would make those moves weaker, which in turn will push the player to using other moves or swapping their existing moves in case and at the same time prioritize defeating it. The monster *Leavanny* has an *Impish* nature, which influences positively the *defence* and negatively the *special attack*. While its highest stat is the attack, the behaviour is the intended one, because each stat has a chance of being influenced positively based on their value. This can be seen in rules (12), where the weight of a stat influenced positively is its value from the base stats. The other monster has all its stats the same value, so in this case it does not matter which stats are influenced positively and negatively.

Name	Leavanny		Castform-Sunny	
Types	Dark, Fairy		Bug	
Nature	Impish (def↑, spa↓)		Calm (spd↑, atk↓)	
Stat values and EV values				
HP	75	EV: 16	70	EV: 23
ATK	103	EV: 60	70	EV: 51
DEF	80	EV: 18	70	EV: 15
SPA	70	EV: 22	70	EV: 23
SPD	80	EV: 4	70	EV: 14
SPE	92	EV: 4	70	EV: 1
Enemies' Moves				
Moves	Play Rough String Shot	Night Daze Withdraw	Fury Cutter Captivate	Screech Pollen Puff
Move Categories	Physical Status	Special Status	Physical Status	Status Special
Move Types	Fairy Bug	Dark Water	Bug Normal	Normal Bug
Enemies' Affixes				
	Domain			

Table 2: Generated enemies for the first scenario

The generated enemies are stronger now compared to the previous level and they are not impossible to defeat if the player does not change their play style. This is the intended result, increasing the difficulty, pushing the player into thinking about their play style and not making them feel extremely weaker compared to the enemy.

5.2 Second Scenario

In the second scenario, the logs can be found in Table (1) and the generated enemies in Table (3). In this scenario it can be seen that the previous completed level was level 10 and the generated enemies both have two types and two affixes. Compared to the previous scenario this one provides a higher challenge to match the higher level the player has reached. In the previous level, the player afflicted the enemies with status ailments (i.e. burn), which in turn, resulted in the first enemy to have the affix *Reverse Ailment*. This affix returns the ailment to the player, in case they would apply it using a move. Knowing this the player should make sure that they do not use moves that apply status ailment effects against it, otherwise they would be affected negatively during the level. The second enemy has the affix *Avenger*, which increases the NPCs stats whenever an ally dies. This means that if the other enemy NPC dies then this one would become stronger. This affix pushes the player to focus on this enemy first before tackling the second one. Moreover, its types are *Grass* and *Psychic*. *Grass* type is effective against both *Water* and *Ground* types (see Image (9)), which are the player's types. This further enforces that the player should focus on the second enemy first. In this scenario both monsters have the same stats but different natures. The first one has a *Quiet* nature, which boosts its *special attack* and affects the *speed* negatively. This is an edge case, because this monster has only one *physical* attack, which was picked because the player's main defence type is *special*. This is still an intended behaviour, because in Pokémon there are monsters whose nature can influence their best stat negatively. Additionally, there is no rule in the grammar which dictates that moves are chosen based on the influenced stat's type. The nature of the second monster is *Bold*, which affects the *defence* positively and the *attack* negatively. This is the intended result, because the *defence* is the highest stat that can be influenced by the nature. The negative influence on the *attack* stat is a random result, but since most of this monster's attacks are *special* attacks, it has little effect.

Name	Giratina		Giratina	
Types	Ground, Ghost		Psychic, Grass	
Nature	Quiet (spa↑, spe↓)		Bold (def↑, atk↓)	
Stat values and EV values				
HP	150	EV: 39	150	EV: 44
ATK	100	EV: 29	100	EV: 54
DEF	120	EV: 1	120	EV: 26
SPA	100	EV: 8	100	EV: 3
SPD	120	EV: 30	120	EV: 0
SPE	90	EV: 5	90	EV: 0
Enemies' Moves				
Moves	Drill Run Flatter	Feather Dance No Retreat	Hidden Power Grass Psyshock	Trop Kick Screech
Move Categories	Physical Status	Status Status	Special Special	Physical Status
Move Types	Ground Dark	Flying Fighting	Grass Psychic	Grass Normal
Enemies' Affixes				
	Reverse Ailment, Sturdy		Avenger, Evolve	

Table 3: Generated enemies for the second scenario

In this scenario, the wanted result was achieved. Compared to the first scenario, the enemies here are stronger and more complex to tackle. This is in line with the level of the encounter, because this scenario is supposed to show an encounter that happen later in the game. One thing that can be noticed here is the fact that both generated monsters have the same name. During generation, the stats of the enemies always have to be slightly bigger than the stats of the previous ones and this means that for later levels the amount of possible stats from the source file is quite small. This is the intended behaviour of the generation, and while it can break the immersion if the same monster is always generated during gameplay, it can be solved by also choosing the name of the monster during generation. For this thesis, this option was out of scope so it was not implemented.

6 Testing and Results

Before providing a conclusion, the DDA system has to be tested in order to provide an answer to the research question. In this section I will provide the techniques I have used to test the research questions as well as the results and their interpretation. To answer the research questions, play test sessions were held and participants were invited to take part in the testing session. In total there were 14 participants, 12 males and 2 females, with ages ranging from 19 to 25 and 9 of them having previous experience with turn-based RPGs and the other 5 having none.

6.1 Engagement

The first supporting research question that has to be answered is *Are players more engaged by a game using DDA than the same game using static difficulty?*.

To assess their engagement I have used the Player Inventory Experience [1] questionnaire. However, out of all 10 constructs provided by the questionnaire I have only picked 6, because the rest were not applicable to the type of DDA system that was designed and those constructs would not be influenced by the two versions of the game. The constructs that were used are: *Curiosity, Immersion, Challenge, Mastery, Meaning, Enjoyment.* Beside PXI, I have observed the behaviour of each player and after the play test session I held a short interview where I asked the participants about which version of the game they felt was more challenging, which version they liked more and whether they felt that the DDA version was unfair towards them whenever they would win a level. Due to the low number of participants the main focus will be put on qualitative analysis of the data.

6.1.1 Questionnaire Results

Before analyzing the data, it has to be checked to see whether it is normally distributed. To check for this the Shapiro-Wilk and Kolmogorov-Smirnov tests were used and a visual method was used as well, by plotting the data on a normal Q-Q plot. From Tables (4) and (5), it can be seen that for the Shapiro-Wilk test most of the p-values are above 0.05, which means that the data is normally distributed. However, for the Kolmogorov-Smirnov test all p-values are under 0.05, which means that the data is not normally distributed. In addition, it can be seen from the Q-Q plots of the gathered data for each construct in Figures (11), (12), (13), (14), (15), (16), (17), (18), (19), (20), (21), (22) that not every set of data is normally distributed. Knowing this, I chose to the assumption that the data is not normally distributed. To analyze the answers to the questionnaire I have used the Wilcoxon signed-rank test [12, p.350], because in order to see whether there is a difference between the two versions of the game, answers to the questionnaire for both constructs need to be compared. When using two populations, this test compares their location using two matched sets of data. This test checks for the null hypothesis that the observations $X_i - Y_i$ are symmetric about $\mu = 0$, where X is data gathered from the questionnaire inquiring about the DDA version of the game, Y is the data collected from the questionnaire about the static version and μ is the mean of the differences of the two datasets. For the alternative hypothesis I chose to check for the greater alternative, where the observations $X_i - Y_i$ are symmetric about $\mu > 0$, because if there is a difference between the DDA and static versions of the game, I want to see whether the DDA version was better than the static version. Because I am testing multiple hypotheses, the possibility of a rare event increases which can result in Type I errors, incorrectly rejecting the null hypothesis.

After calculating the p-values for each construct I have also applied the Holm-Bonferroni correction [24], to counteract Type I errors resulting from multiple comparisons. Using this method, each p-value is no longer compared to α, but it is compared to $\frac{\alpha}{m+1-k}$, where m is the amount of hypotheses I have and k is the k-th hypothesis. In addition, all constructs are ordered by their p-value in ascending order when applying this correction.

Furthermore, when measuring the data collected, the statistical test was also applied to subgroups of the data, more specifically to the data collected only from participants with previous experience or participants without any previous experience with turn-based RPGs. The reason was that, even though the collected data is small, I speculate that there is a difference between how an experienced participant experienced the system compared to a non-experienced participant.

After running the Wilcoxon signed-rank test over the whole data for each construct, the following calculated p-values can be seen in Table 6. It can be seen that besides *curiosity*, where the p-value is

Construct	Shapiro-Wilk pvalue	Kolmogorov-Smirnov pvalue
Meaning	0.615	0.00012
Curiosity	0.057	0.00012
Mastery	0.016	1e-05
Immersion	0.811	0.00014
Challenge	0.453	0.00152
Enjoyment	0.357	0.0

Table 4: P-values of each construct when checking for normality the data from the DDA version

Construct	Shapiro-Wilk pvalue	Kolmogorov-Smirnov pvalue
Meaning	0.299	0.00014
Curiosity	0.12	0.00392
Mastery	0.177	0.00014
Immersion	0.134	0.00152
Challenge	0.155	0.01132
Enjoyment	0.159	1e-05

Table 5: P-values of each construct when checking for normality the data from the static version

0.006, for every other construct we fail to reject the null hypothesis at a 95% confidence level, meaning that there is no difference for these constructs between the DDA version of the game and the static version of it. However, this also means that participants might be more curios to see how the DDA version would change over time.

Construct	p-value	α value	α value after correction
Curiosity	0.007	0.05	0.008
Challenge	0.068	0.05	0.01
Immersion	0.074	0.05	0.012
Enjoyment	0.092	0.05	0.016
Meaning	0.525	0.05	0.025
Mastery	0.775	0.05	0.05

Table 6: P-Values for each construct using all participants' data

After splitting the data into two categories, participants that have previous experience with turn-based RPGs and participants with no previous experience, we have the following p-values in Tables 7 and 8. Here it can be seen that when the participant has no previous experience, for all constructs we fail to reject the null hypothesis at a 95% confidence level. This shows that the DDA version was not more engaging than the static one. However, the exact Wilcoxon signed-rank test was shown to only work if the sample size is greater or equal to six [17], so for a sample size of five, the resulting p-values have an increased likelihood of creating Type II errors.

For experienced players there is a difference when it comes to *curiosity*, *immersion* and *enjoyment*. Here, these constructs have a *p-value* smaller than 0.05, which means that at a 95% confidence level the null hypothesis is rejected, which means that the DDA version might be more immersing, make the player more curious and be more enjoyable than the static version of it. However, after applying correction, we can see that only the p-value for *curiosity* is smaller than the corrected value of α. Nonetheless, because for the rest of the constructs we failed to reject the null hypothesis, it cannot be concluded that the DDA version is more engaging than the static one.

6.1.2 Player observations

Beside collecting participants' questionnaire answers, I have observed how the player would act during the game and I would ask them about their experience after the play test session, as mentioned in Section (3). Overall, the participants felt that the DDA version was more challenging, but for two of the players that had no previous experience with turn-based RPGs it proved to be frustrating.

Construct	p-value	α value	α value after correction
Curiosity	0.006	0.05	0.008
Enjoyment	0.016	0.05	0.01
Immersion	0.043	0.05	0.012
Challenge	0.082	0.05	0.016
Meaning	0.338	0.05	0.025
Mastery	0.547	0.05	0.05

Table 7: P-Values for each construct using only experienced participants' data

Construct	p-value	α value	α value after correction
Curiosity	0.209	0.05	0.008
Challenge	0.312	0.05	0.01
Immersion	0.393	0.05	0.012
Meaning	0.828	0.05	0.016
Mastery	0.889	0.05	0.025
Enjoyment	0.892	0.05	0.05

Table 8: P-Values for each construct using only inexperienced participants' data

However, experienced players felt that the added challenge was more stimulating and enjoyed this experience. Moreover, during their play sessions, the majority of the experienced players looked bored compared to when they were playing the dynamic version. This is most likely caused by their need for challenge as a main motivation. The feeling of an increased challenge contradicts the results of the statistical tests, which shows that there is no difference between both versions of the game. However, since the p-value for challenge in Tables (6) and (7) is small this means that contradiction can be caused by the low number of participants.

When it comes to which version was more enjoyable, one participant, with no prior experience, said that the static version was more enjoyable because they now understood the game better. This was caused by the fact that they first tried the DDA version and then the static one. Out of the players with previous experience, six of them felt that the DDA version was more enjoyable, while the rest of them felt that the game was as enjoyable. For the players with no previous experience, 3 of them thought the DDA version was as enjoyable as the static one, while the rest of them thought the DDA version was less enjoyable.

When it comes to unfairness, no player felt that the DDA version was more unfair than the static version. However, during the play test of the static version one the inexperienced participants got stuck for too much time in one level of the static version and wanted to end the testing. Since this was only one case, it can be considered as an edge case, but it shows that a game with static difficulty can put the player in this situation. Whereas, in the DDA version after losing for a couple of times the participant would have been able to finish the level, because the difficulty would have decreased.

Finally, after every loss, the participants would adapt their playstyle, regardless of the version of the game. However, 2 out of 14 participants, 1 with no previous experience and 1 with, claimed that they felt more compelled to change their playstyle even when they would win in the DDA version because of the perceived increase in challenge.

From this data, I conclude that the DDA version has the potential to improve the game by making it more enjoyable and provide the right amount of challenge. However, at the moment there are some contradictions between the qualitative data and quantitative data regarding the challenge of the DDA version compared to the static one and the DDA version was mostly more enjoyable only for the experienced participants.

6.2 Action variety

To answer the second supporting research question, *How does the variety of actions used by a player change during a playthrough?*, player logs had to be collected after each play session. The logs, which differ from the ones used to adapt the difficulty, contain the amount of moves used by each candidate per level, as well as their names, to observe whether the moves that were used also differ over time. The variety of actions over a playthrough can be seen in Figure (7). As it can be seen from the figure, there is a small difference in variety for the DDA and static versions of the game. On average, participants had to use more different actions per level when playing the DDA version, than when playing the static version of the game.

For the DDA version it can be seen that the highest amount of different actions was used on the second level of the game. This coincides with the first generation of enemies, where there is no upper bound on generation so the difficulty increases substantially. This also matches the average increase of difficulty per level in Figure 8. Here it can also be seen that compared to the static version, the DDA version difficulty at the second level increased considerably. However, it can be seen that after the second level the variety of actions decreased a bit and remained constant. This means that the player reached a balanced difficulty and does not struggle to finish the levels. Furthermore, in Figure 8 it can be seen that after the second level there is a slight decrease in the game difficulty and then it remains constant.

For the static version of the game, it can be seen that the difficulty keeps increasing at a fixed rate, but after the second level the average amount of different actions used per level does not go above 3 and it remains constant.

Finally, the results collected match the first scenario expected, where the variety when using the DDA system is higher. From this data I conclude that the DDA system achieved its second objective to make the players adapt their playstyle and change the variety of their actions used.

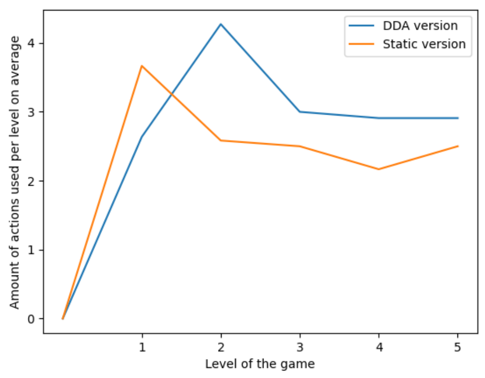


Figure 7: Average variety of actions used by players

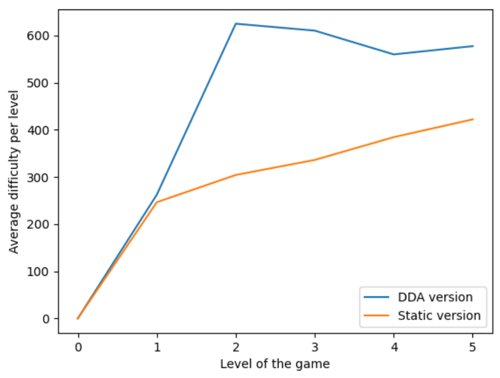


Figure 8: Average difficulty of levels

7 Reflection & Future Improvements

Prior to delving into the conclusion of the research questions, a brief introspection on the created system will be provided, along with suggestions for potential future updates to enhance the system. The significance of these updates will also be discussed.

7.1 Reflection

When I started this thesis, my aim was to develop a system that would make the player adapt to the situation and not always use the same tactics by changing the strengths and weaknesses of the enemy. While this proved successful by bringing an extra layer of challenge, it is far from perfect. During the beginning of it, I have also learned that one of the limitations of this system was the game. While Pokémon is a well known game and was well suited for this thesis, I had to change several mechanics to reduce the scope of the game such that it matches the grammar. In addition, the game had to have a well defined structure of the enemies, which properly defines their strengths and weaknesses. In this case Pokémon proved the perfect candidate, because the stats, moves and affixes of a monster influence how strong or weak one is.

In the process of conducting a literature review, I found that several papers have explored the concept of adapting the difficulty of a game by altering the behaviour of enemy NPCs. For instance, Spronck et al. [47] presented a system that uses dynamic scripting to adapt the difficulty dynamically, while Butler et al. [9] employed generative grammars to create new attack patterns for boss enemies to adapt the difficulty. Although my proposed DDA system shares some similarities with these approaches, such as using generative grammars and adapting enemies to adapt the difficulty, it is different in the sense that it does not change the behaviour of the enemy NPCs. Instead, my system adapts the structure of the enemy through the generation of NPCs attributes, actions, and affixes. This unique approach is what sets my system apart from the previous literature. As highlighted in Section (2), the generative grammar-based approach and the use of enemy adaptation to dynamically adjust difficulty are not new concepts. However, my DDA system enhances these concepts by introducing new augments to the generative grammar. The literature review helped me understand where this thesis stands among existing literature and the amount of possibilities for future improvements, such as adapting the behaviour or game parameters. It also helped with outlining the scope of this thesis and identifying the elements of novelty, the grammar augments and the generation using attributes, actions and affixes.

The use of a generative grammar to create the structure of an NPC proved successful, and in its current state it can be applied to any turn-based RPG that is similar to Pokémon. The rules of the grammar provided the right balance between randomly generating values and generating them in a predictable way. This helped with generating NPCs that adapt to the player, but at the same time do not feel like they are cheating and always counteracting them. Furthermore, the use of attributes helped with setting up interdependencies between attributes, actions and affixes. For example, in rules (15), the weight of the *Reverse Weakness* affix is the attribute *Weakness*, which is influenced by the type the NPC has and the types the player has, which can be seen in rules (10).

Regarding the results, I have expected the differences between the static and dynamic versions of the game to be more apparent among the players that had no previous experience with this game. However, current data showed the contrary. This does not mean that the presented system does not improve the play experience, as it showed an increase in curiosity for players that already have previous experience with the used game.

7.2 Future improvements

In this section I will present two types of possible improvements that can be done for the system. The first ones are the improvements that can be done to the grammar, while the second will tackle the future improvements that can be made for the play testing sessions.

7.2.1 Grammar improvements

There are four main functionalities that can be added to further improve the grammar. The first one is the addition of an upper bound during the generation. At the moment, during the generation the stats of the NPCs should always be at least bigger by one than the weakest enemy from the previous level. This ensures the increase in difficulty for the next stage, but at lower levels this increase might be too steep. Because of this the addition for an upper bound might be needed for future updates. Furthermore, when a player loses the difficulty is decreased by decreasing the total stats of the enemies, but because there is no upper bound then the difficulty can spike again after the player has won the level, meaning that the player might lose the level again until it reaches the wanted difficulty. By adding the upper bound this problem can be avoided.

The second future update is the addition of new augments which can indicate the order in which rules are expanded. One case where this is needed is when the EVs are generated. As it can be seen in rule (13), the order of the EVs is health points, attack, defence, special attack, special defence and finally speed. While this works in practice, we do not always want the HP EV to be the first generated, because it can happen that it receives the highest possible value, and then every other EV will have a lower value. This behaviour is not always wanted because an increase in health points does not translate directly to an NPC being stronger. In competitions for example, players would usually assign 252 EV points to two stats and everything else, 6 points, would be added to a third stat. This means that there is a priority of stats and not all stats should receive EV points. A possible solution for this would be the addition of an augment which takes into consideration the play style of the player and chooses three augments which should receive EV points, 2 main which receive 252 and one which receives the remaining 6.

Another possible update would be to see how the difficulty can be adapted by also changing the amount of enemies the player fights. Right now the player fights two enemies and there is a difference of two levels between them. For this update the grammar would have to determine the level difference between the player and enemies based on how many enemies are generated.

Finally, a minor update which would influence how battles work is to create specific roles for NPCs, by adapting their behaviour. For example, let's assume that the player likes to only attack using physical attacks. Then in the next level we would have 3 enemy NPCs, two which would act as supporting characters and would only defend the third one against the physical attacks of the player by enhancing the defence, while the third one would be the only one fighting. This would further compel the player to change their strategy and would also work together when the number of enemy NPCs is used to adapt the difficulty. This was already proved to work in DDA [47, 26, 15], so it can be a useful future addition.

7.2.2 Play testing improvements

For play testing the main improvement that can be done to gather more accurate results is to increase the number of participants. While having only 14 participants showed that there are some differences between the DDA and static versions of the game. However, a small sample size is not always representative of the group and can lead to a bias. In addition to this, the participants that tested the system were further split into two subgroups: those with previous experience with turn-based RPGs (nine participants) and those without (five participants). An increase in the sample size would be able to remove any possible bias, as well as analyze the answers of the questionnaire at the level of the subgroup. This can be more beneficial because different types of players have different ways of reaching motivation [5, 30, 35]. Moreover, the difference in motivation can also be used to further split the two groups of participants into more groups. For example, even though two participants have the same experience with turn-based RPGs, one of them can be more motivated by a challenging encounter while the other one is less motivated by it. Based on motivation, the players can be split using the BrainHex survey [36], which will ensure that the results of the play test sessions will reduce the amount of bias as much as possible.

8 Conclusion

In the proposed DDA system, the aim was to create a more engaging game experience for players by using a novel approach of adapting the difficulty of levels through the generation of NPCs' attributes, actions, and affixes. To achieve this, a generative grammar was used, to which new augments were added.

The results of the study conducted in Section (6) suggest that the quantitative analysis of the data did not show a significant difference in engagement between the DDA and static versions of the game. On the other hand, the qualitative data analysis indicated that participants perceived the DDA version to be more challenging. This increase in perceived challenge was also accompanied by an increase in the variety of actions used by the players, suggesting that a more challenging experience will naturally lead to an increase in the amount of different actions used by the player.

It is worth noting that participants with previous experience found the DDA system to be more engaging than the static version. This suggests that the DDA system is suitable for players who are looking for a more challenging experience. However, further research is needed to explore the motivations and preferences of different player subgroups, such as those identified in the BrainHex model [36].

In conclusion, the DDA system successfully provided an extra layer of challenge and increased the variety of actions used by players compared to the static version. Further research is needed to explore the preferences of different player subgroups and to investigate the potential of additional updates to the system.

References

[1] V. V. Abeele, K. Spiel, L. Nacke, D. Johnson, and K. Gerling. Development and validation of the player experience inventory: A scale to measure player experiences at the level of functional and psychosocial consequences. *International Journal of Human-Computer Studies*, 135:102370, 2020.

[2] H. Alblas. Introduction to attributed grammars. In *SAGA*, volume 545 of *LNCS*, pages 1–15, CZ, 1991. Springer.

[3] M.-A. Anagnoste. Procedural generation of skill trees in video games using graph grammer, Jan 1970.

[4] L. Bagheri and M. Milyavskaya. Novelty–variety as a candidate basic psychological need: New evidence across three studies - motivation and emotion, Oct 2019.

[5] R. Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. *Journal of MUD research*, 1(1):19, 1996.

[6] S. Bowman. *Immersion and Shared Imagination in Role-Playing Games*, pages 379–394. 04 2018.

[7] C. Browne, M. Stephenson, É. Piette, and D. J. N. J. Soemers. A practical introduction to the ludii general game system. In T. Cazenave, J. van den Herik, A. Saffidine, and I.-C. Wu, editors, *Advances in Computer Games*, pages 167–179, Cham, 2020. Springer International Publishing.

[8] S. Bunian, A. Canossa, R. Colvin, and M. S. El-Nasr. Modeling individual differences in game behavior using hmm.

[9] E. Butler, K. Siu, and A. Zook. Program synthesis as a generative method. In *FDG*, New York, NY, USA, 2017. ACM.

[10] A. Cannizzo and E. Ramírez. Towards procedural map and character generation for the moba game genre, 2015.

[11] N. Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[12] W. J. Conover. *Practical nonparametric statistics.* John Wiley Sons, Inc., 1999.

[13] M. Cook, S. Colton, and J. Gow. The angelina videogame design system—part i. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(2):192–203, 2017.

[14] I. M. Dart, G. De Rossi, and J. Togelius. Speedrock: Procedural rocks through grammars and evolution. In *PCGames*, New York, NY, USA, 2011. ACM.

[15] S. Demediuk, M. Tamassia, W. L. Raffe, F. Zambetta, X. Li, and F. Mueller. Monte carlo tree search based algorithms for dynamic difficulty adjustment, 2017.

[16] D. Dimovska, P. Jarnfelt, S. Selvig, and G. N. Yannakakis. Towards procedural level generation for rehabilitation. In *PCGames*, New York, NY, USA, 2010. ACM.

[17] A. K. Dwivedi, I. Mallawaarachchi, and L. A. Alvarado. Analysis of small sample size studies using nonparametric bootstrap test with pooled resampling method. *Statistics in medicine*, 36(14):2187–2205, 2017.

[18] D. Dziedzic and W. Włodarczyk. Approaches to measuring the difficulty of games in dynamic difficulty adjustment systems. *International Journal of Human–Computer Interaction*, 34(8):707–715, 2018.

[19] H. Fernandez, K. Mikami, and K. Kondo. Perception of difficulty in 2d platformers using graph grammars. *International Journal of Asia Digital Art and Design Association*, 22(2):38–46, 2018.

[20] A. Gellel and P. Sweetser. A hybrid approach to procedural generation of roguelike video game levels, 2020.

[21] R. Gusmão, K. Calixto, and C. Segundo. Dynamic difficulty adjustment through parameter manipulation for space shooter game, 09 2015.

[22] J. Gutierrez and J. Schrum. Generative adversarial network rooms in generative graph grammar dungeons for the legend of zelda. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2020.

[23] A. Hintze, R. Olson, and J. Lehman. Orthogonally evolved ai to improve difficulty adjustment in video games, 03 2016.

[24] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[25] R. Hunicke. The case for dynamic difficulty adjustment in games, 2005.

[26] M. Ishihara, S. Ito, R. Ishii, T. Harada, and R. Thawonmas. Monte-carlo tree search for implementation of dynamic difficulty adjustment fighting game ais having believable behaviors, 2018.

[27] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin. Polymorph: Dynamic difficulty adjustment through level generation, 2010.

[28] Z. Jiang, S. Earle, M. Green, and J. Togelius. Learning controllable 3d level generators. In *FDG*, New York, NY, USA, 2022. ACM.

[29] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius. Pcgrl: Procedural content generation via reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 16(1):95–101, Oct. 2020.

[30] R. Koster. *Theory of Fun for Game Design*. O'Reilly Media, Inc, USA, 2013.

[31] C. Langin. Languages and machines: An introduction to the theory of computer science. *Scalable Comput. Pract. Exp.*, 8, 2007.

[32] L. LeBron. Procedural character generation for narrative games.

[33] B. Li, R. Chen, Y. Xue, R. Wang, W. Li, and M. Guzdial. Ensemble learning for mega man level generation. In *FDG*, New York, NY, USA, 2021. ACM.

[34] R. Lämmel. *Software Languages*. Springer, Switzerland, 2018.

[35] A. Marczewski. *User Types*, pages 65–80. CreateSpace Independent Publishing Platform, South Carolina, USA, 2015.

[36] L. E. Nacke, C. Bateman, and R. L. Mandryk. Brainhex: A neurobiological gamer typology survey. *Entertainment Computing*, 5(1):55–62, 2014.

[37] J. Nakamura and M. Csikszentmihalyi. *The Concept of Flow*. Springer Netherlands, Dordrecht, 2014.

[38] V. Palban. Managing difficulty in games, May 2021.

[39] M. d. Plater, C. H. Hoge, R. K. H. Roberts, D. P. Valerius, R. A. Newton, and K. L. Stephens. Nemesis characters, nemesis forts, social vendettas and followers in computer games, Aug 2021.

[40] A. S. Ruela and F. G. Guimarães. Procedural generation of non-player characters in massively multiplayer online strategy games - soft computing, Jul 2016.

[41] A. Salomaa. Probabilistic and weighted grammars. *Information and Control*, 15(6):529–544, 1969.

[42] S. Satheesh, H. Narasimhan, and P. Senthil. Evolving player-specific content for level based arcade games. In *FDG*, page 329–330, New York, NY, USA, 2009. ACM.

[43] V. T. G. K. H. M. Schattke, Kaspar; Brandstätter. Flow on the rocks : motive-incentive congruence enhances flow in rock climbing, 2014.

[44] N. Shaker, J. Togelius, and M. J. Nelson. *Procedural Content Generation in Games.* Springer Publishing Company, Incorporated, 1st edition, 2016.

[45] S. Snodgrass and S. Ontañón. Learning to generate video game maps using markov models. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(4):410–422, 2017.

[46] C. Snyder and S. Lopez. *Oxford Handbook of Positive Psychology.* Oxford library of psychology. Oxford University Press, 2009.

[47] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma. Adaptive game ai with dynamic scripting - machine learning, Mar 2006.

[48] A. J. Summerville and M. Mateas. Super mario as a string: Platformer level generation via lstms. In *Proceedings of the First International Joint Conference of DiGRA and FDG*, Dundee, Scotland, August 2016. DGRAS.

[49] A. Tychsen, M. Hitchens, and T. Brolund. Motivations for play in computer role-playing games. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, Future Play '08, page 57–64, New York, NY, USA, 2008. Association for Computing Machinery.

[50] K. Vahé. *Building an RPG with unity 2018: Leverage the Power of Unity 2018 to build elements of an RPG.* Packt Publishing, 2018.

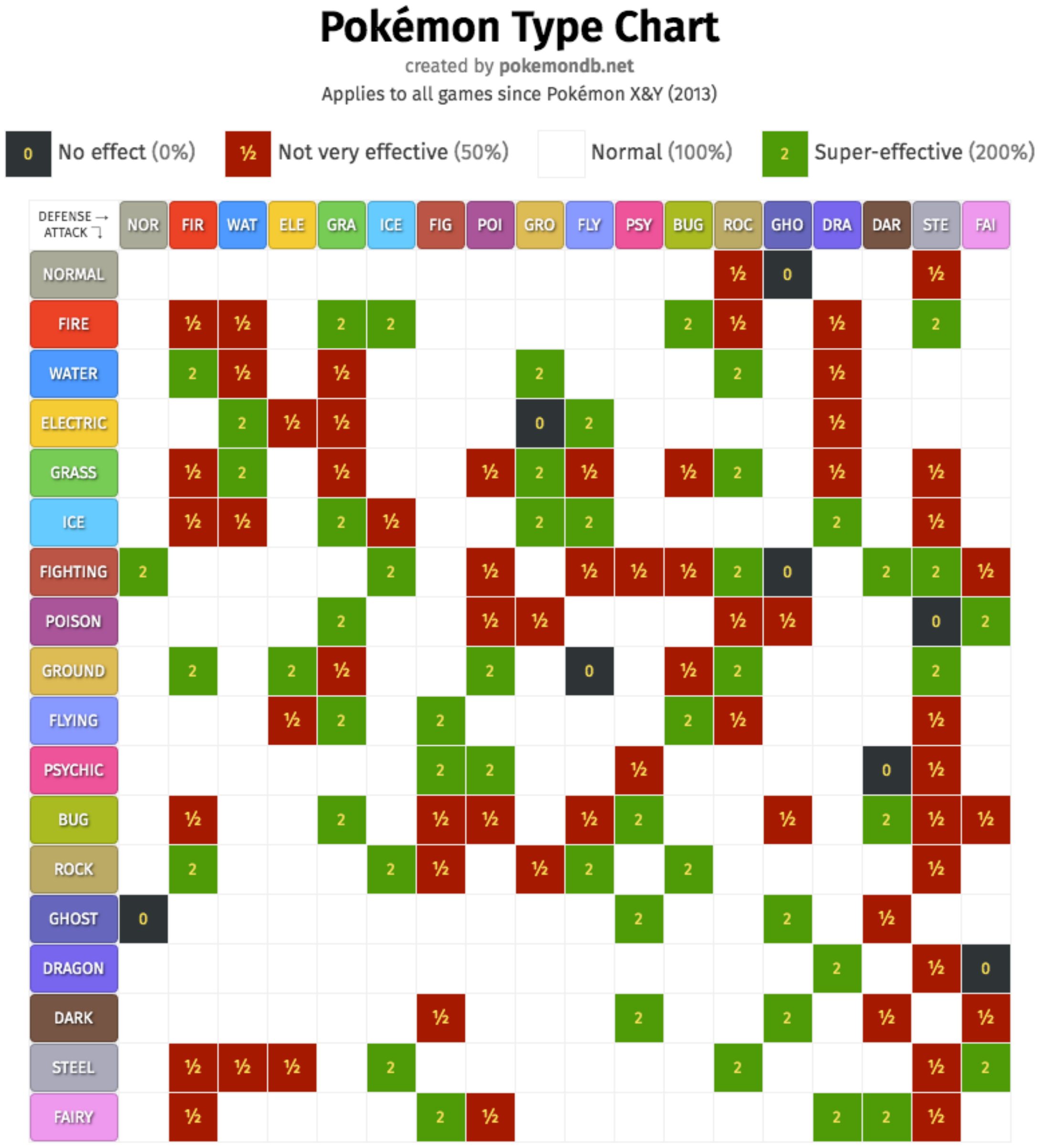
[51] S. Vidman. Dynamic difficulty adjustment & procedural content generation in an endless runner, 2018.

[52] D. Wheat, M. Masek, C. P. Lam, and P. Hingston. Dynamic difficulty adjustment in 2d platformers through agent-based procedural level generation, 2015.

[53] S. Xue, M. Wu, J. Kolen, N. Aghdaie, and K. A. Zaman. Dynamic difficulty adjustment for maximized engagement in digital games. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, page 465–471, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[54] M. Zohaib. Dynamic difficulty adjustment (DDA) in computer games: A review. Hindawi, Nov 2018.

A Pokémon data



Pokémon Type Chart

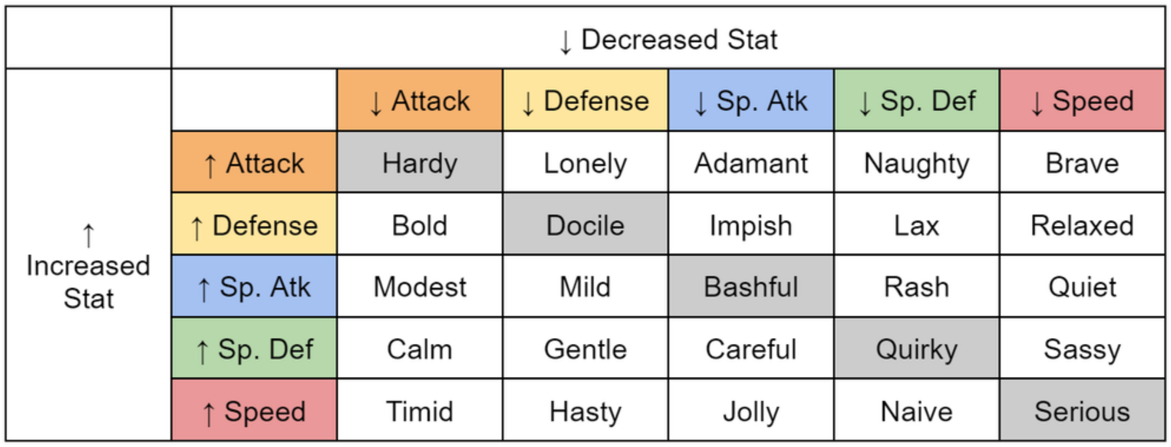
created by **pokemondb.net**

Applies to all games since Pokémon X&Y (2013)

0	No effect (0%)
½	Not very effective (50%)
	Normal (100%)
2	Super-effective (200%)

DEFENSE → ATTACK ↴	NOR	FIR	WAT	ELE	GRA	ICE	FIG	POI	GRO	FLY	PSY	BUG	ROC	GHO	DRA	DAR	STE	FAI
NORMAL													½	0			½	
FIRE		½	½		2	2						2	½		½		2	
WATER		2	½		½				2				2		½			
ELECTRIC			2	½	½				0	2					½			
GRASS		½	2		½			½	2	½		½	2		½		½	
ICE		½	½		2	½			2	2					2		½	
FIGHTING	2					2		½		½	½	½	2	0		2	2	½
POISON					2			½	½				½	½			0	2
GROUND		2		2	½			2		0		½	2				2	
FLYING				½	2		2					2	½				½	
PSYCHIC							2	2			½					0	½	
BUG		½			2		½	½		½	2			½		2	½	½
ROCK		2				2	½		½	2		2					½	
GHOST	0										2			2		½		
DRAGON															2		½	0
DARK							½				2			2		½		½
STEEL		½	½	½		2							2				½	2
FAIRY		½					2	½							2	2	½	

Figure 9: Type effectiveness chart in Pokémon



		↓ Decreased Stat				
		↓ Attack	↓ Defense	↓ Sp. Atk	↓ Sp. Def	↓ Speed
↑ Increased Stat	↑ Attack	Hardy	Lonely	Adamant	Naughty	Brave
	↑ Defense	Bold	Docile	Impish	Lax	Relaxed
	↑ Sp. Atk	Modest	Mild	Bashful	Rash	Quiet
	↑ Sp. Def	Calm	Gentle	Careful	Quirky	Sassy
	↑ Speed	Timid	Hasty	Jolly	Naive	Serious

Figure 10: Pokémon natures

B Q-Q Plot Questionnaire Data

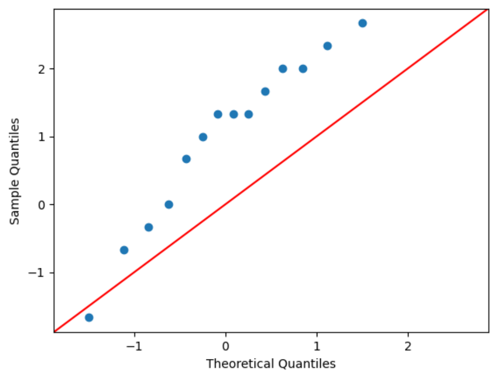


Figure 11: Q-Q Plot of answers for the *Challenge* constructs answers for the DDA version

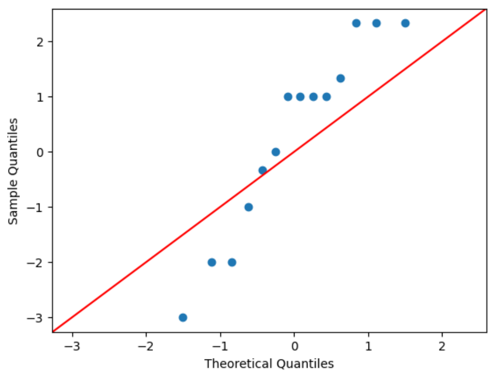


Figure 12: Q-Q Plot of answers for the *Challenge* constructs answers for the static version

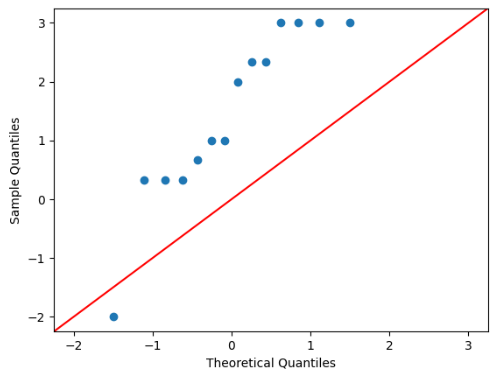


Figure 13: Q-Q Plot of answers for the *Curiosity* constructs answers for the DDA version

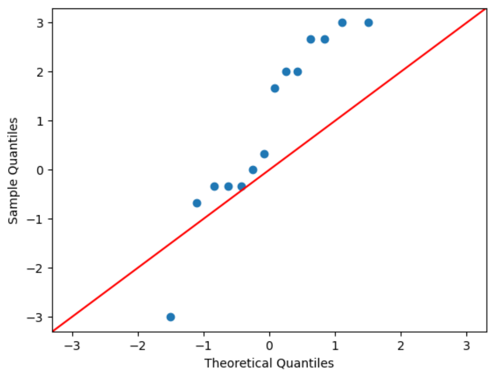


Figure 14: Q-Q Plot of answers for the *Curiosity* constructs answers for the static version

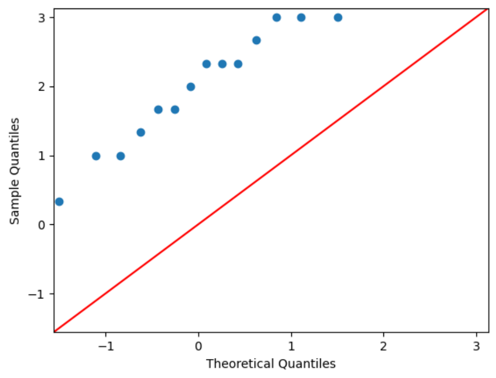


Figure 15: Q-Q Plot of answers for the *Enjoyment* constructs answers for the DDA version

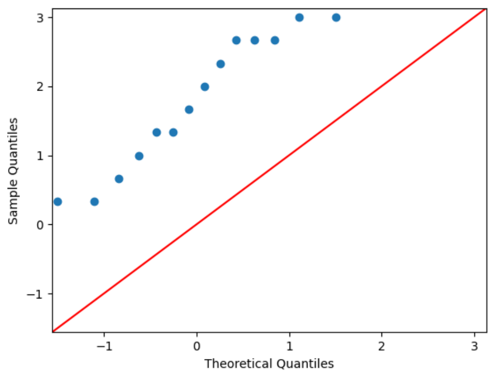


Figure 16: Q-Q Plot of answers for the *Enjoyment* constructs answers for the static version

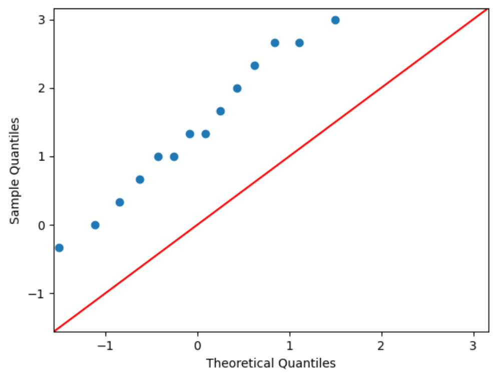


Figure 17: Q-Q Plot of answers for the *Immersion* constructs answers for the DDA version

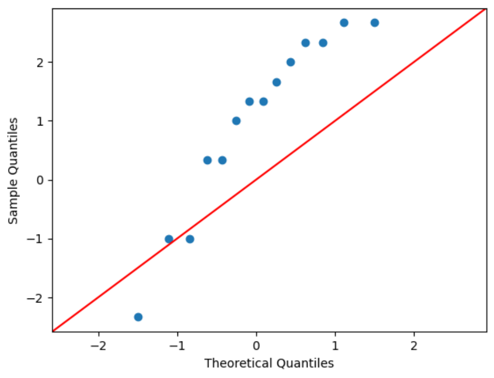


Figure 18: Q-Q Plot of answers for the *Immersion* constructs answers for the static version

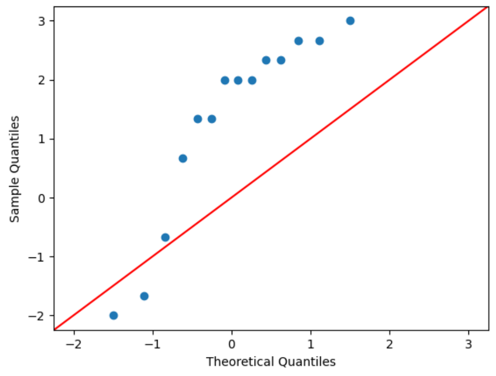


Figure 19: Q-Q Plot of answers for the *Mastery* constructs answers for the DDA version

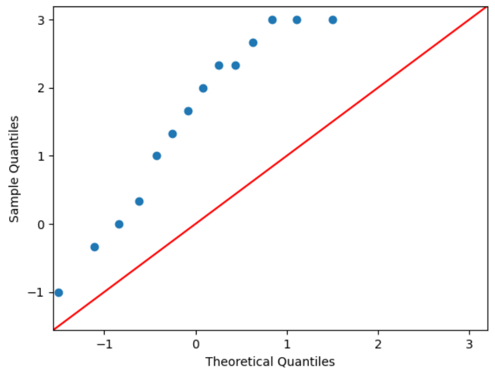


Figure 20: Q-Q Plot of answers for the *Mastery* constructs answers for the static version

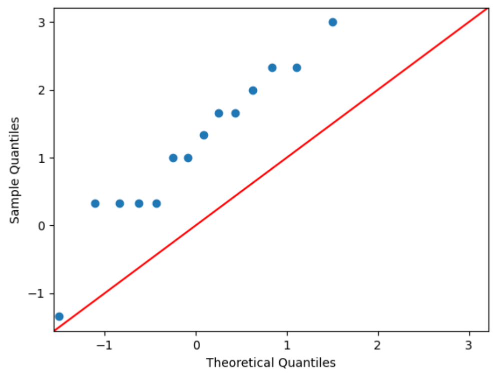


Figure 21: Q-Q Plot of answers for the *Meaning* constructs answers for the DDA version

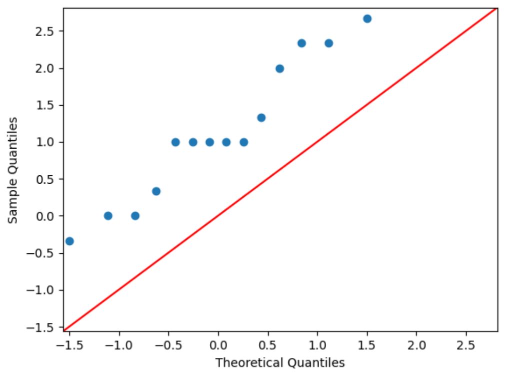


Figure 22: Q-Q Plot of answers for the *Meaning* constructs answers for the static version