FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Matouš Kozma

Procedural Generation of Combat Encounters in Role Playing Video Games

Department of Software and Computer Science Education

Supervisor of the master thesis: Mgr. Jakub Gemrot, Ph.D.

Study programme: Computer Science

Specialization: Computer Graphics and Game Development

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In…..... date..... signature

I would like to thank my thesis supervisor Jakub Gemrot, whose help while writing this thesis was invaluable. I would also like to thank Lukáš Kolek, who helped me design the surveys for the players. I would also like to sincerely thank everyone who took part in the experiment.

Název práce: Generování bojových střetnutí v počítačových hrách na hrdiny

Autor: Matouš Kozma

Katedra / Ústav: Katedra software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Jakub Gemrot, Ph.D.

Abstrakt: Procedurální generování je přítomno v mnoha dnešních hrách. Málo prací se ovšem zabývá generováním bojových střetnutí v počítačových hrách na hrdiny (RPG). V těchto hrách se hráčovi bojové schopnosti liší vyrazně člověk od člověka a je v nich veliké množství nepřátel, se kterými může hráč bojovat. Díky tomu je generování bojových střetnutí velice obtížné. V této práci vytvoříme malou hru, ve které implementujeme nový algoritmus na generování bojových střetnutí. Tuto hru pak distribujeme mezi veřejnost. Ze získaných dat dojdeme k tomu, že tento algoritmus je alespoň tak dobrý v generování soubojů jako autor práce a že se jedná o dobrý základ k dalšímu výzkumu.

Klíčová slova: procedurální generování, počítačové hry, hry na hrdiny

Title: Procedural Generation of Combat Encounters in Role Playing Video Games

Author: Matouš Kozma

Department / Institute: Computer Graphics and Game Development

Supervisor of the master thesis: Mgr. Jakub Gemrot, Ph.D.

Abstract: Procedural content generation is present in many games today. However, little has been written about generating combat encounters in role playing video games (RPG). In these games the player's combat ability varies greatly from person to person and there are many different enemies that could be spawned for the player to fight. These factors make generation of combat encounters difficult. In this thesis we create a small game in which we implement a new algorithm for generating combat encounters. We then distribute this game to the general public. From the data we gather we conclude that this algorithm is at least as good at generating combat encounters as the author of the thesis and that it is a good starting point for further research.

Keywords: procedural generation, video games, role playing games

Table of Contents

1 Introduction

Many games today implement some form of procedural content generation[1, 2], where some parts of the game are generated by an algorithm. This can greatly reduce the work required from designers or artists. It can also improve the replay value of the game, as the levels can be generated at runtime whenever the player starts the game. The procedural content generation problem can be split into many subproblems – we can separately study the best ways to generate many areas of a game, e.g., environments or dungeons. In this thesis we will study the procedural generation of combat encounters, i.e., how to best select a group of monsters that the player should face at once.

The problem of generating combat encounters is complicated by the many criteria we want to consider when generating encounters, some of which cannot be easily computed. The encounters need to be varied in tactics needed to approach them. They should also be varied in the exact enemies the player fights in the encounter. But above all, the encounter must be exactly as difficult as the designer intends. Too many easy encounters in a row might lead to the player getting bored, but too many difficult encounters might result in the player being frustrated, neither of which is desirable.

In this thesis we will focus on procedural generation of combat encounters in role playing video games (RPG). In RPGs, the player controls one or more heroes as they travel through the game world, fighting enemies and experiencing the game story. The central element of RPG games is leveling, where the player characters get stronger as they defeat enemies and progress through the story. The player usually has a lot of freedom in how she develops her characters. Because of this, the players' combat power differs greatly from player to player. This makes it increasingly difficult to design combat encounters as the game progresses, as two players might have completely different groups of characters. Procedural generation of combat encounters could solve this problem, as the algorithm could generate encounters at runtime. These encounters could be tailored to the specific player. Yet it is hard to implement properly, mainly because there are many possible configurations of player characters the player might have. Some players could also be much better than other players at playing the game.

Players can also have completely different play styles. Some players might focus on the roleplaying aspect and choose equipment and abilities that will match the personality of their characters. Other players spend a lot of time on figuring out the best possible combinations of abilities and items to create the most powerful party possible. And other players might not think about these things at all. Ideally, combat encounters should be engaging for all different kind of players or, in other words, RPG games should support different play styles.

We have not found any existing research regarding this specific problem. In this thesis we will introduce a new algorithm for solving the problem of generating combat encounters in RPGs and we will evaluate it. However, as this problem is quite complex, it is not our goal to create a solution that would be ready to use in commercial games. Instead, we formulated a general algorithm with loose ends (as even RPG games are hard to define formally) and implemented it in a short and simple game we developed for the purpose of algorithm's preliminary validation. Furthermore, the algorithm itself is composed of several parts, each solving a certain subproblem, which are hard to solve completely by themselves[1]. For these subproblems we have chosen the simplest possible solution, in order to test whether the approach in general is promising at all.

The rest of this work is structured as follows: The rest of this chapter describes the problem in more detail and further describes RPG games in general. In chapter 2 we discuss some related problems, how are they different from our problem and what, if anything, is related to this thesis. In chapter 3 we further analyze the problem, listing all the things we need to consider while designing the algorithm. In chapter 4 we first discuss whether we should study this problem in a custom game or if we should try to adapt it to an existing game. After that we introduce our custom game, listing all the design elements the reader needs to know about in order to understand further chapters. In chapter 5 we propose a new algorithm as a solution for this problem and we explain how we implemented it in the game from chapter 4. In chapter 6 we introduce our testing methodology, as well as all the hypotheses we want to confirm in our tests. In

[1] For example, we need to determine how similar in terms of combat difficulty would two completely different encounters be for the player's party and play style.

chapter 7 we list the results of our experiment. In chapter 8 we discuss the results of the experiments and suggest possible future work based on this thesis.

1.1 Introducing RPG games

In order to describe the problem in more detail, we must first describe its domain – role playing games (RPGs). In RPG games, the player controls one or more heroes as they travel through the game world. As they travel, they earn *experience points* by defeating enemies and completing quests. When they get enough experience points, they can level up. When a hero levels up, she gets stronger in some ways. Often this is numerical, e.g., level up might mean simply improving the character's damage or health. In some games the player can choose a new skill for the character to gain. This skill can then be used in combat.

Experience points are not the only reward the player gets while playing the game. She can also find new weapons and armor to equip her heroes with. These can modify their strength even further. Some skills and equipment complement each other, therefore correct gear and skill selection might greatly alter the character's strength.

In games where the player controls a party of heroes, it is also common to have more heroes in the game than the player can control at once. The player can then select which of these available heroes she wants to have in her current party.

An example of such a game is Dragon Age: Origins[3]:

Figure 1: Combat in Dragon Age: Origins

Source: Custom screenshot from the game made by Electronic Arts

The above screenshot shows a combat encounter in the game. The characters with yellow circles are the player characters and characters with red circles under them are enemies. In the bottom part of the screenshot we can see a list of skills the currently selected character can use. Each of the player characters has 6 different attributes that influence e.g., damage with specific weapons, hit points or defense. Every time a character levels up he can increase some of his attributes and gains a new skill. The level up bonuses cannot be easily changed. In addition, each character can equip different items, e.g., armor, weapons or magic rings. These can be changed whenever the player desires. Also, while the player cannot change his main player character, the player also controls 3 additional companion characters. The player can select these characters from a pool of up to 9 possible companions he can meet during the game.

As Dragon Age: Origins is a successful commercial RPG game with many combat encounters, we will use it in thesis repeatedly as an example when we wish to explain some concept of RPG games. While many RPG games exist, we believe that Dragon Age: Origins is complex enough to serve as a model RPG for the purpose of this thesis.

1.2 Problem overview

In this section we will define the problem of procedural generation of encounters in RPGs and examine the complexity of the problem.

1.2.1 Definition

First, we must formally define several terms commonly used in RPG games which we will use throughout this thesis (defined terms and their labels are in italics).

- An *enemy* is a single opponent the player can fight. For our purposes we will define it as a tuple of all the attributes that define the enemy in the game world, e.g., its size, durability, damage, skills, or AI. These attributes are game specific. We will label the set of all enemies in the game E.
- An *enemy group* is a multiset formed of elements from E and represents a group of enemies the player might meet.
- A *player character* is a single actor the player controls in the game world. We will define it in the same generic way as an enemy, i.e., as a tuple of all attributes that define the player character.
- A *party* is a set of all player characters the player currently controls.
- *Combat area* represents the part of the game world where the combat takes place. In some games this could be a grid, in some games this could be a more complex 3D map. We will define it as a set of all possible positions where a party member or an enemy can stand. We will label this set L.
- *Combat encounter* represents a situation where a group of player characters must defeat a group of enemies in the game, we label the set of all encounters C. We call them combat encounters because games might have different types of encounter, e.g., puzzle encounters where the player is expected to solve a puzzle. However, as no other kinds of encounters are relevant to the thesis, from now on we will refer to them as *encounters* for the sake of brevity. An encounter is defined as a tuple of some of these elements:

- It must always contain the enemy group the player will fight in this encounter. We will label this enemy group E_c.
- It must contain the party that the player will control in the encounter. We will label it P.
- In some games, enemy placement is important. For these, the encounter must also contain an injective mapping s_e, assigning each element from E_c a starting location l from L.
- Player starting positions can also be defined in the same way, as an injective mapping s_p from P to L. It should also not map any element to locations already used by the mapping s_e.
- And there can be many more game specific elements. For example, in Dragon Age 2 [4], many encounters are divided into several waves, where a next wave appears only when the current one is almost defeated. Therefore, the encounter definition would contain multiple groups of enemies that would appear throughout the encounter. As these elements are specific to each game, we consider them to be out of scope for this thesis.

Furthermore, for every encounter we must be able to measure its quality, which would take into the account both the player's engagement in the encounter as well as the designer's intention. Therefore, we need to define a function $f:C \rightarrow \mathrm{R}$ that will assign each encounter a score, where 0 means that the encounter is perfect and the higher the number, the worse the encounter. We will call this function *encounter evaluation function*.

With these definitions, we can now define procedural generation of combat encounters as the following optimization problem:

Given a set of all possible combat encounters C for a given party P and combat area L, and an encounter evaluation function f, find c_{min} from C such that for all c in C $f(c_{min}) \leq f(c)$.

We will refer to this problem as *encounter generation problem* from now on.

1.2.2 Complexity

The main difficulty lies in limiting *C* and defining *f*. This will always be greatly dependent on the specific game, which the algorithm should be designed for.

First, let us discuss the set of all possible combat encounters. In the simplest case the encounter is a set of enemies. The enemies do not have to be unique, so the number of possible encounters with k enemies is $|E|^k$. We can expect that there will be some maximum number of enemies that can be spawned called n. Therefore, the number of possible will be $\sum_{k=1}^{n} |E|^k = \frac{|E|(|E|^n-1)}{|E|-1}$. This can be reasonable in games with a limited number of different enemies and/or smaller number of enemies in combat. However, in most commercial games there will be dozens of different enemy types, each with several possible variations. They also tend to feature larger battles from time to time. To get an idea how large these numbers can be in commercial games, we will examine encounters in Dragon Age: Origins[3].

The Dragon Age wiki[2] lists 140 creatures that appear in the game. Actual number of enemies in the game is harder to determine, as some of these monsters are boss monsters that appear only once per game and probably would not be included in the normal encounter generation. On the other hand, most of these creatures exist in 3 different strength levels (Normal, Elite, Boss). And they also have a numeric level between 1-20 which further define their strength. Taking all these factors into account, we can safely assume that there are at least 100 different enemies that can appear in encounters.

As for the number enemies that can appear in the one combat, we will look at one specific combat, the boss fight near the end of the game (see below). In this encounter the player faces 14 enemies at once. So even with very conservative estimates of $|E|=100$ and $n=10$, we get approximately 10^{20} different encounters.

[2] https://dragonage.fandom.com/wiki/Category:Dragon_Age:_Origins_creatures

Figure 2: Boss encounter in Dragon Age: Origins

Source: Custom screenshot from the game made by Electronic Arts

However, *C* can be even larger if position is relevant. It can be easily seen that the number of possible encounters would increase to $\sum_{k=1}^{n} |E|^k * \binom{|L|}{k}$ Even if we assume a very small combat area, a 5x5 grid, it still has 25 possible starting locations, which results in approximately 10^{26} encounters. And usually *L* will be much larger than that.

From this we can see that especially if locations are important, any algorithm solving the encounter generation problem will need to heavily prune the search space of possible combat encounters, as enumerating them all would not be possible in real time.

While the space of all encounters is huge yet well defined, encounter evaluation function cannot be easily defined, as many of the factors it must consider are subjective, e.g., measuring player skill. The following is a non-exhaustive list of factors the algorithm must consider when evaluating an encounter.

- Party configuration – which heroes the player controls and how powerful they are.
- Player skill – how skilled is the player at playing this game.

- Desired encounter difficulty – how difficult should the combat encounter be. Ideally this should consider both the designer's directive and the player's previous encounters. For example, if the last encounter was supposed to be easy and was instead hard, this encounter's target difficulty might be lower than usual, which should keep the player from being frustrated.
- Fun factor – we need to keep the player entertained and some fights are just not as fun as others.
- Designer constraints – the designer might choose to put some artificial constraints to make the experience better. For example, let us say that the encounter is a fight against an enemy commander. The designer would probably want the encounter to feature as many enemies as possible, to make the player feel like he is actually fighting a commander of an army. Therefore, the evaluation function might favor encounters with more enemies.

Most of these factors cannot be objectively converted into a number, e.g., there is no single number that could describe the player's skill in an objective way. Furthermore, all these factors are likely to change as the game progresses, making it more difficult to precompute the possible values for the evaluation function (a favorite PCG trick used for algorithms that cannot be used during play time of the game).

Note that in this definition we concern ourselves only with generating a single encounter. However, the ideal algorithm for generating encounters would also consider the encounters generated in the past, because the encounters should be different from one another, both in terms of enemies generated and in terms of the tactics necessary to defeat those enemies. In this thesis we assume that this will be done by removing encounters deemed to be too similar from C in some preprocessing step before solving the optimization problem. However, it is possible that future research will have to extend this definition by letting the evaluation function also consider the encounters generated in the past.

1.3 Generating combat encounters in RPG games

The player has a great deal of freedom when choosing how to develop and equip her player characters and possibly which player characters to bring to an encounter.

And as most of these things can be changed between encounters, should the player decide to do so, it would affect all future encounters, as suddenly the party configuration would be completely different.

There is also a great variance in the player's skill. The player's party might have some skills which work great together. However, if the player never uses them like that, they cease to be powerful.

Some RPG games also employ some specific weaknesses and strengths of some monsters. For example, a fire golem might be resistant to fire and slashing attacks and weak to ice and bludgeoning attacks. So, while this monster might not be difficult for an average party, it might be almost undefeatable for a party that uses only fire and slashing attacks.

Positioning is also important in many of these games, mainly because ranged characters tend to have high attack while being easy to defeat if an opponent gets close. In such a game, three enemy archers and a melee fighter might normally be normally an easy encounter, but might be much more difficult if the archers are standing on top of a wall and the only way up the wall is blocked by the fighter.

RPG games also tend to be focused on the story and the player's immersion in the game world. Any algorithm for generating monsters must take that into account. For example, if in the game world goblins and orcs are enemies, they should not appear together and fight side by side.

And these are just some of the things any procedural combat generator must consider while generating enemies for the player in RPGs. Huge variability of player strength, large number of possible monsters and complex combat rules make it quite difficult to accurately measure the combat's difficulty, while the setting and the story might put constraints on what monsters can be generated.

However, as this is the first work dealing with this issue, we do not aim to create an algorithm that would excel in all these areas. Instead, we will focus only on a small part of the problem and we will try to make an algorithm that could eventually be extended to take all these features into account.

2 Related works

While procedural content generation in games is a well-studied field, it seems that existing research does not focus on generating encounters. The works we found about content generation in RPGs focus either on generating the dungeon levels without focus on enemies[5, 6], or they focus on generating quests in the game[7, 8]. But we found no research that would focus on generating encounters in RPGs.

We tried looking at other genres for inspiration, yet we were unable to find any relevant research. It seems that even in genres such as platformers[9] or roguelikes[10] the main focus is on generating the level, not on generating monsters.

2.1 Dynamic difficulty

Even though we found no existing research about generating encounters, we were able to find several algorithms that try to adjust the difficulty of a game to the player's ability. And even though there are many criteria to evaluating an encounter's quality, it needs to be appropriately difficult. Therefore, techniques for runtime adjustment of difficulty might be relevant for us.

Xue et al.[11] describe the results of adding a dynamic difficulty to a match-three game. While match-three games are quite different from RPG games, two points from the paper could be relevant to us. First, adding dynamic difficulty raised engagement significantly. And as engagement is one of our main priorities, we should focus on providing appropriate difficulty. Second, the algorithm did not have a fixed target difficulty for each level. Instead they used a large database of player behavior without dynamic difficulty adjustments to compute how difficult should the current level be to maximize engagement. While we cannot use the exact algorithm proposed in the paper, as engagement in RPG games is more complex than in match-three games, it is important to note that our algorithm should probably be able to match any difficulty, so it can be used with a difficulty curve, whether the curve is provided by some other algorithm or a designer.

Missura and Gärtner[12] introduced an algorithm that could split the players into groups based on their playstyle and assign players to these groups. While this algorithm is probably not relevant for this thesis, as it requires a lot of data about player behavior that is unavailable to us, we should keep in mind that the possibility of

assigning a playstyle to the player exists. It could allow us to make use of offline preprocessing for each of these playstyles separately. And during the game we could use the correct offline data based on the player's group.

2.2 Algorithm evaluation

The algorithm we design in this thesis will have to be evaluated in some way. As the goal of the algorithm will be generating encounters suitably difficult for the players, any evaluation must involve real people playing the encounters generated by the algorithm. And we will need to find some way of evaluating their experiences. We have chosen to measure the player's feeling of flow[13] as the measure of quality for the algorithm.

In psychology, flow is a state where the person is fully concentrating on some task. It is also known as being "in the zone". A person in a flow state loses track of time and she focuses completely on the activity she is doing. To enter the flow state, the person must feel competent at the task and the task must be appropriately difficult for her.

And creating appropriately difficult challenges is the goal of this thesis. Therefore, if the player is in the flow, we can assume that the algorithm is correctly adjusting the difficulty. So, we need to find a way to determine how in the flow the player is feeling. We found a survey called Flow Short Scale[14, 15]. This survey measures the flow on a 7-point scale and a person's perceived difficulty of the task related to his other tasks on a 9-point scale. We will use this survey to evaluate player's feeling of flow, which will allow us to evaluate the algorithm.

3 Analysis

As there are no existing solutions to this problem, we will have to create a new algorithm from scratch. Before doing that, we will need to properly define our priorities and limit the scope of the broad problem in order to create an algorithm that could serve as a starting point for further research.

3.1 Priorities

From our analysis, it seems that the algorithm must be able to fulfill these conditions to be usable in commercial RPG games:

1. The algorithm should work online and adjust itself to player's party configuration and their skill level.
    - This should help us improve engagement. However, the algorithm can depend on precomputed data. This is a common practice in game development. For example, it is common to precompute navigation meshes which specify all the areas where the player can walk. At runtime, the pathfinding algorithm can work with these navigation meshes instead of working with the complex 3D world geometry.
2. The designer should be able to influence the encounters generated by the algorithm as much as possible.
    - This is necessary if the algorithm should ever be used in story heavy games. That is because in these games, there might be constraints on which enemies can appear when and which enemies can appear together. Control over the algorithm is also necessary in commercial games, were the designers need to be able to tweak the algorithm if it outputs bad results in some specific cases.
3. The algorithm should be able to output encounters in whatever difficulty the designer desires.
4. The algorithm should be generic enough to allow it to be used in many commercial RPG games.
    - However, we expect that the algorithm will have to be heavily modified to fit any specific game. Generating good encounters is likely to be tied heavily to the mechanics of the specific game.

5. The combat encounters generated should try to vary the enemies generated as much as possible.
    - RPG games tend to have many different enemies for the player to fight, so the algorithm should try to vary the enemies as much as the designer allows. Also, usually different groups of enemies require different tactics to defeat. The algorithm should generate combat encounters in such a way that would force the player to switch tactics often. We will call both conditions together *encounter variance*.

3.2 Scope

As we are creating a completely new algorithm from scratch, it seems likely that the algorithm itself will not yet be the perfect algorithm for this problem. Instead it will likely be a starting point. Because of that we will try to put some limitations in place. These should help us keep the algorithm in the scope of a master thesis, yet it should also be easy to overcome them with future research.

Ideally the algorithm should be implemented and tested in multiple games with different mechanics to verify whether it can truly be used universally in RPG games. Unfortunately, that would be well beyond the scope of this thesis. Instead we will select one specific game which we will use to verify whether the algorithm is at all promising. We will also try to limit the complexity of the algorithm as much as possible. If we design a simple algorithm and our experiment shows it does not work at all, we will know that we should select a different approach in the future. If the algorithm were too complex and it did not work, we would not know whether we should select a different approach or just tweak it to get better results, e.g., by selecting better heuristics.

Related to the first point, from our priorities it seems clear that there are many priorities for the algorithm, which we listed in the previous section. However, designing an algorithm that would excel in all these areas would be quite difficult in the scope of single thesis. Therefore, our focus will be on creating and evaluating an algorithm that can create encounters of a specific difficulty. Yet the algorithm should also have the potential to be extended in order to overcome this limitation.

4 Testing environment

From the analysis, it seems clear that any encounter generation algorithm will have to be tailored to the specific game it will be implemented in. Therefore, it is imperative to choose the game in which we will evaluate the algorithm well. In the first section of this chapter we will explain why we chose to create our own game in which to evaluate the algorithm instead of implementing the algorithm in an existing game. In the second section we will list the basic requirements for the game to be at least similar in its basic principles to commercial games. And in the last section we will explain the game and its mechanics in detail.

4.1 Existing game vs. custom game

Early on we had to choose whether we should try implementing this algorithm in one of the existing commercial games or if we should create a new game from scratch. There are many commercial RPG games and some of which provide tools to extend them, e.g., Dragon Age: Origins[3] or Neverwinter Nights 2[16]. We will now list the factors we considered when making this decision and explain why we chose to create our own game in the end.

- When developers release a toolset to their game, the reason behind it is usually to allow players to create custom levels, modules, and other minor modifications. Therefore, we could not be sure whether these toolsets would give us sufficient control over the game and its world to implement the algorithm. Such uncertainty does not exist with a custom game.
- Commercial RPGs take at least tens of hours to finish and the game design reflects that. They tend to have many rules and features and it takes a long time for the player to learn all of them. Therefore, it would be difficult to craft a game experience that could be short enough that the test subjects could finish it some reasonable timeframe and still learn it well enough that they could get into flow. One of the main requirements for entering the flow state is that the subject must understand what he is doing and feel competent in the area. And an unexperienced player

would not feel competent at playing e.g. Dragon Age: Origins after 30 minutes of playing.

- Both Dragon Age: Origins and Neverwinter Nights were popular in the past. Therefore, it would be quite likely that some players would be much better at the game than others. If we also consider that we will most likely have very few test subjects, it would make it much harder to do any experiment, as our experiment groups would have to be fragmented further based on their experience with the game.
- RPGs tend to have a lot of skills, items, and other ways to customize their character. If we implemented our algorithm in a commercial game and it would work, it would be great, as we could show that the basic idea of the algorithm is solid and can be extended to complex games. However, if it did not work, there would be no way to verify whether the problem was in the idea behind the algorithm or in our implementation, as we likely do not understand the complex mechanics of these games well enough.
- On a related note, as we mentioned in the previous chapter our focus is on simplicity. A custom game can be much simpler than a commercial project.
- Creating a custom game would be a lot more work. Therefore, if we created a custom game, we would have to spend less time on fine tuning the algorithm.
- As we have little experience with RPG game design, it was quite likely that the custom game might have some design issues that might prevent players from enjoying the game. Or they might not be able to understand it.
- Our inexperience with RPG design made the prospect of creating a custom game appealing, as we could learn a lot of new skills while doing that.
- Extending a commercial game would remove the option of sending the game to a wide audience to test, as we do not have the license to freely distribute these games. Which means that the people who would want to try out our implementation of the algorithm themselves would have to

purchase the game, which would further limit the number of people we could reach.

After careful consideration of all these factors, we decided to create a custom game tailored to this thesis. Mainly because it would be easier to design an experiment and because it would be easier to keep algorithm in the scope of this thesis. We decided to name the game we created Thesis Quest.

4.2 Thesis Quest requirements

The game needs to be a simplification of RPG games, yet it must not be so oversimplified as to make the algorithm irrelevant to commercial games. So, we defined this small set of requirements which should keep the game from getting too simple. To show that these requirements are present in commercial games, we will show how the game Dragon Age: Origins[3], or DAO for short, fulfills them:

- The player must control several characters at once. These characters must be in some ways different from one another.
    - In DAO, the player controls up to 4 party members. Each party member has a profession, which is either a fighter, a mage, or a thief. The player is likely to develop these characters further to fulfill some specific roles, like healing, damaging enemies, preventing enemies from fighting etc.
- The party should grow in power over the course of the game.
    - As mentioned in the introduction chapter, on every level up the player can increase character's attributes and they can also learn new skills. The player is also likely to frequently find better equipment.
- The player should be able to make choices that would change how the party grows in power. Some of these choices should be better than others.
    - There are many choices the player must make in DAO. As an example, the characters can learn only a limited number of skills. Some of these skills make encounters much easier, e.g. a sleep spell can put many enemies to sleep, which can greatly simplify large encounters, as the player can defeat the sleeping enemies

one by one. Players who learn this spell might find these large encounters to be much easier.

- There should be a variety of enemies in the game.
    - As mentioned in the introduction chapter, there are 141 different creatures listed on the game wiki.
- The enemies should differ both in their strength and in their approach to combat.
    - Each enemy in DAO has a rank and level described in the introduction chapter which greatly affect the strength of the enemy. Enemies also have different AI scripts and have different skills. For example, enemy dogs can pin characters to the ground and damage them over time, enemy mages disable player characters and archers stand in the back and deal heavy damage to their targets.
- The player should be able to make tactical decisions within an encounter.
    - The player must make many decisions. For example, she chooses which enemies to attack first, which to put to sleep or stun in some other ways, which skills to use against which enemies, how to position her heroes etc.
- There should be some resource that can be depleted in an encounter. This resource can take many forms – in some games the player can gather items he can use only once to give himself a temporary boon, other games have spells and skills usable once a day and yet others have long term injuries from combat which cannot be easily healed.
    - First, DAO has single use items like healing potions or mana potions. Also, whenever a character falls in battle, he is resurrected after the battle with a random injury which gives a long-term disadvantage to the character. For example, a cracked skull reduces the character's cunning attribute. These injuries can be healed with single use items or by resting, which cannot be done inside a dungeon.

We believe that a game fulfilling all these requirements would be a reasonable simplification of RPG games. We also believe that the game described in the next

section fits these criteria. Note that the goal of this game is to allow us to run experiments with the encounter generator. Therefore, we did not focus too much on game design, as that would be complex enough to warrant its own thesis. Instead we created a short game which fulfills these requirements, and which can be completed in a reasonable time by inexperienced players.

4.3 Thesis Quest description

Thesis Quest is a rogue like RPG in which the player controls a group of three heroes. The player must guide them through several floors (levels) of a dungeon. Each dungeon floor is comprised of several rooms. And apart from the first room in every floor, all rooms contain an encounter. The encounter is generated only when the player opens a room, so the encounter should always match the player's ability. When the player opens a door leading to an unexplored room, her heroes enter it and they cannot leave the room until the encounter is resolved.

The game is real-time, but the player can pause the game whenever she wishes. While the game is paused, she can still issue commands to her heroes.

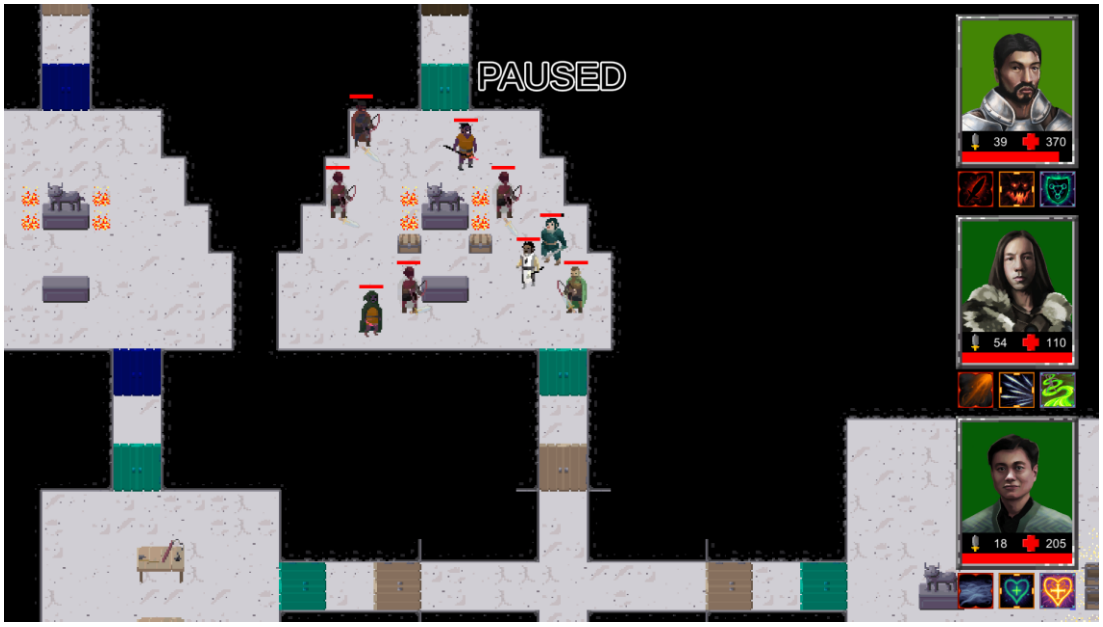

Figure 3: Screenshot from Thesis Quest

4.3.1 Heroes

The player controls three heroes. Each of these heroes has three abilities that will be described below. These abilities differ in who they can target. One ability can target

only the hero casting it, one can target only friendly heroes and one can target only enemies. Whenever the hero uses one of his abilities, all his abilities are unusable for some time. This should force the player to think tactically about which abilities she will use and when.

These are the three heroes the player controls and their abilities:

- Knight – The knight has high health and moderate damage. His main purpose is to be the target of attacks of other enemies. This way he can protect other heroes from harm. His skills are:
    - Enemy skill: Power Strike. The knight attacks with a swift blow that deals double damage to an enemy and forces that enemy to attack the knight for a short time. The enemy will also be knocked back and stunned for a second.
    - Personal skill: Taunt. This skill will force all enemies to attack the knight. The skill has an exceptionally long cooldown.
    - Friendly skill: To the Rescue. The knight rushes to an ally and forces all nearby enemies to attack him.
- Ranger – The ranger has extremely low health and high damage. His main purpose is to defeat the enemies from a distance. However, his low health makes him vulnerable should he be attacked. He has the following skills:
    - Enemy skill: Sniper Shot. The ranger fires an arrow that does quadruple damage, dispatching many enemies in one hit.
    - Personal skill: Rapid Stance. This skill doubles the ranger's rate of fire, although it slightly reduces his damage per shot. Overall, his damage per second is increased by 50%.
    - Friendly skill: Poison Cloud. A poison cloud appears around an ally, poisoning nearby enemies, halving their attack and movement speed.
- Cleric – The cleric has moderate health and low damage. His main role is to keep his allies alive.
    - Enemy skill: Sleep. The target of this spell is put to sleep and is unable to act for a while. The target will wake up if he takes damage.

- Personal skill: Healing Aura. The cleric starts healing nearby allies and himself for the duration of this spell.
- Friendly skill: Heal Other. The targeted hero is completely healed.

4.3.2 Health

As is common in RPG games, every enemy and hero has hit points (HP), a number that specifies how much damage he can survive. When a hero takes damage from his enemies, his hit points (HP) decrease. However, unlike most RPGs, when a hero's HP reach zero, he does not die. Instead, the damage taken from then on will decrease the hero's maximum HP limit. Only when his maximum HP reach 0 will he die. This applies only to heroes, when an enemy's HP reaches 0, he dies.

Heroes are automatically revived after every encounter with low maximum HP. There is no way to revive them during an encounter. Some abilities can allow heroes to heal each other. However, these abilities cannot heal lost maximum HP, they can only restore lost HP up to the current maximum HP.

The heroes can restore lost maximum HP in two ways. First, throughout the dungeon the player can find potions that can restore lost maximum HP. Second, maximum HP are completely restored when the player finishes a level.

When the player's entire party dies, the game is over. The player can then choose to try again. If she does so, she will start at the beginning of the current floor.

4.3.3 Enemies

To allow the players to make meaningful and tactical decisions about which enemies to target first, there needs to be a meaningful difference between them, both in terms of their power and in their tactics. While designing this system, we were inspired by the tabletop roleplaying game Dungeons & Dragons[17]. In the fourth edition of this game, the enemies are defined by two criteria. First, each enemy has a rank which defines how powerful the monster is. Second, each enemy has a role that defines what kind of skills the monster has, its attributes and how it should behave. While we cannot directly use these roles and ranks, mainly because our game is much simpler, we were inspired by this and chose the following ranks and roles for our enemies:

- Ranks:
    - Minion – a weak enemy. He can be defeated in one hit and deals low damage. Minions should appear in large numbers and always together with other monsters.
    - Regular – an average enemy. He has no skills and should be of little threat on his own, though they can be dangerous in large numbers.
    - Elite – should be twice as strong as regular enemies. He will start using skills when his health drops under 50%.
    - Boss – should be twice as strong as elites. He uses skills whenever possible. Even a single boss can be a challenge for a beginner player with a party without upgrades.
- Roles – here we will list the different roles in the game and individual monsters that have these roles. Unless otherwise stated, the monsters exist in three ranks – regular, elite, boss:
    - Minion: Minions have extremely low health and damage. They can only have the minion rank. They attack closest enemies and they have no skills.
        - Goblin (minion), Imp (minion)
    - Brute – brutes have high health and low damage. They wield only melee weapons and attack always the closest enemies. Their skill is to become enraged, increasing their attack and movement speed.
        - Skeleton (regular, elite), Minotaur, Orc Fighter
    - Lurker – lurkers have low health but extremely high damage. They attack in melee and try to target and kill the ranger as fast as possible. If the ranger is dead, they target the cleric. Their skill is to attack their target several times in quick succession. Boss lurker is capable of instantly killing weaker characters with this attack. Boss lurker is also capable of teleporting directly to his target.
        - Orc Thief
    - Sniper – snipers have moderate health and moderate damage. They attack from a distance. Regular snipers choose their targets

randomly with every attack. Elite and boss rangers also choose targets randomly, but after the first attack they will lock onto that target and shoot at him until he dies. Their skill is to increase their attack speed, the same as ranger's Rapid Stance.

- Evil Ranger, Skeleton Mage

- Leader – leaders only exist in elite and boss variants. There can be only one of them in any single encounter. Leaders have moderate health and low damage. They will always go after the ranger first, cleric second and knight third. But most importantly, if the ranger is alive, the leader will order all other creatures in combat to attack the ranger. Leaders also have a skill they use when the ranger is dead or if the leader is the last enemy alive. Leaders can activate a healing aura just like player's cleric, continuously healing himself and nearby allies.
  - Evil Cleric (elite, boss)

4.3.4 Combat rewards

In most rooms the player will find not only monsters, but also treasure chests. The treasure chests are locked during an encounter, but the player can open them once the encounter is over. Each chest can contain one of the following items that give bonuses to the hero who picks it up:

- Health potion – the potion restores 50% of maximum HP lost to the character who picks it up.
- Damage upgrade – a permanent upgrade to hero's damage output. The upgrade will work differently based on the hero who picks it up. Ranger will have his damage increased by a large amount, knight by a moderate amount and cleric by a low amount.
- Maximum HP upgraded – this upgrade will permanently increase the hero's maximum HP limit. As with the damage upgrade, the actual bonus will depend on the hero who picks it up. The knight will get a large increase to his health, cleric will get a moderate increase and ranger will get a minor increase.

The bonus will always be the same whenever the same character picks the item up. This is a change from the original design. We first wanted the items to multiply the relevant attribute by 1.2. However, this worked poorly for two reasons. First, the initial powerups felt almost worthless. Second, if the player had a single character pick up every single powerup, soon the game lost all its tactical depth, as that character could easily defeat almost all enemies in one or two hits and was almost invincible himself. Therefore, we opted for the option of constant attribute increase. These make initial powerups feel powerful without ruining the later parts of the game.

To summarize, we will show how this game fulfills the requirements from the previous section.

- The player must control several characters at once. These characters must be in some ways different from one another.
    - The player controls three characters, each with different skills, attributes, and roles in combat.
- The party should grow in power over the course of the game.
    - The attribute upgrades the player finds in chests increase stats, which increase characters power significantly. For example, if the ranger picks up all upgrades, by the end of the game he can kill in one shot all normal enemies, many of them even if the Rapid Stance is active. And he can kill even boss level enemies with the Sniper Shot.
- The player should be able to make choices that would change how the party grows in power. Some of these choices would be better than others.
    - The player chooses how to distribute the stats. Some of these options are better than others. For example, giving damage upgrades to the ranger is always better than giving them to the cleric.
- There should be a variety of enemies in the game.
    - There are 9 enemies in the game, 21 if we count all elite and boss variants of these enemies.
- The enemies should differ both in their strength and in their approach to combat.

    - Each combat role defines a different approach to combat. The monsters exist in regular, elite, and boss variants to create the difference in strength.
- The player should be able to make tactical decisions within an encounter.
    - The player can decide on the order in which she will defeat her enemies. She must also carefully think about which skills to use when.
- There should be some resource that can be depleted in an encounter.
    - This resource are the maximum hit points which can be reduced during an encounter.

5 Algorithm description

In this chapter we will introduce the algorithm we designed. First, we will explain the high-level concept of the algorithm. In the second section we will split the problem into several smaller problems. We will explain what each subproblem is about and how we decided to solve in Thesis Quest. In the last section we will then put all these components together to present the algorithm in detail.

5.1 Main idea

The main goal of this algorithm is to create encounters of an appropriate difficulty. It uses a precomputed matrix in which the results of many encounters are stored. This algorithm can then estimate the difficulty of an encounter by searching the matrix for the encounters which are the most similar to the one being evaluated. And once we can evaluate a difficulty, it is easy to generate an encounter. We can start with a random encounter and then use the hill climbing algorithm to get to the correct difficulty. We evaluate the difficulty of the current encounter, remove or add an enemy, and repeat this process until the difficulty is satisfactory.

Once an encounter is over, we can update the matrix, as we just got another encounter result. Also, we know exactly how off our original estimate for the difficulty was. We can now traverse the entire matrix, find encounters similar to the one which just ended and adjust their estimated difficulty appropriately. Over the course of the game the matrix should be changing to fit the abilities of the player playing the game.

5.2 Subproblems

In this section we will describe several subproblems the algorithm must solve. These subproblems are most likely game dependent. Therefore, when adapting the algorithm to other games, new solutions will probably have to be found for each of these subproblems.

For Thesis Quest we solved these subproblems with the simplest solutions available, because as we mentioned in the analysis, simplicity is one of our goals when designing this algorithm.

5.2.1 Measuring difficulty

Our algorithm requires a way to measure the difficulty of an encounter that just finished. Ideally, the meaning of this number should remain the same throughout the game. E.g., if 2 means that an encounter was of a medium difficulty, it should mean that both at the start and at the end of the game.

In Thesis Quest, we decided to measure difficulty by adding together the percentage of the maximum HP the heroes lost during the fight. This means that when the combat starts, we will log the current maximum HP of every character. And once the combat ends, we will check how many percent of the starting value each character lost, and we will add it together. This will give us a number between 0 and 3, where 0 means that no maximum HP was lost and 3 means that everyone in the party was killed.

This is indeed a measure of difficulty and has two advantages. First, the designer can now set the goal for the algorithm in a very understandable manner. Second, it is quite easy to check how close was the algorithm to the target difficulty when evaluating the algorithm.

However, this approach is also not without its flaws. Mainly we cannot differentiate between difficulties of extremely easy or difficult combat encounters. Therefore, the designer cannot ask for encounters in which no maximum HP would be lost, as the algorithm could quite conceivably generate an encounter with a single minion. Similarly, on the difficult side, if the party is killed, there is no difference between an encounter that was extremely close and the player might win if she tries hard enough, and an encounter where the player did not stand a chance.

5.2.2 Difficulty matrix

The difficulty matrix is used to store information about which party configurations are strong against which enemy groups. In this matrix, the columns are indexed by party configurations and the rows are indexed by enemy groups. And for matrix M, enemy group e and party configuration p, $M\ [e,\ p]$ should contain the estimated difficulty of this encounter. The matrix can be sparse, but it should be as populated as possible. It should be initialized by values that will estimate a behavior of a beginner player as much as possible. During runtime it will be often updated to match the player's actual performance.

While it is clear how this matrix should work, it is not evident how to precompute the initial matrix. In Thesis Quest, we created a combat simulator to generate it. The simulator went through many different combinations of party configurations and enemy groups and spawned them both in the game. It let them fight, stored the results and prepared the next fight. This process is described below in detail. After several days, the matrix was ready to be used by the algorithm.

The simulator generated the party to fight and the enemy group independently. The combat was randomly generated with a given number of monsters to spawn. First 4000 times it would spawn 2 random monsters, next 4000 times 4 monsters etc. In each encounter there was a 50% probability a that a single leader would be in the enemy group, otherwise no leader would be present. The rest of the monsters were added randomly to the enemy group until the limit was reached.

We specified several ways how the player might distribute powerups as he picks them up, e.g., give all powerups to the knight, distribute them evenly, randomly etc. We then used these powerup distribution strategies to create party configurations the player would have after picking up some specific number of power ups. These are the parties that were fighting against the enemy groups.

The AI for heroes used in the simulator is extremely simple. When it has nothing to do it will either with 66% probability attack the nearest enemy. If it does not decide to attack, it will pick a random skill and use it at a random target. If the skills are not usable yet, it will always attack.

We originally wanted to use a more sophisticated AI. This AI made tactical decisions about which enemies to target and when to use skills. However, this AI created a matrix that was not representative of how an average or a beginner player would play the game. It could identify the most dangerous enemies and quickly eliminate them, marking extremely dangerous encounters with difficulty 0.

5.2.3 Enemy group difference function

This function is used to estimate similarity of encounters, which we need when updating the matrix and determining the difficulty of encounters. It should accept two enemy groups as parameters and return either 0 if those two enemy groups are the same, or a positive number describing how different the enemy groups are from each

other. Where the lower the number, the more similar the enemy groups are. The function should describe the difference as much as possible for the game. As an example, if the game has enemies with elemental strengths and weaknesses, two enemy groups of the same approximate difficulty but with monsters of different elements should not be considered similar. Similarly, if two enemy groups require vastly different tactical approach to defeat, they should not be considered similar. In short, the lower the result of the function, the higher the likelihood that the enemy groups would be similarly difficult for any possible party in the game.

In Thesis Quest, we decided not to care about specific enemies in an enemy group. While there are, e.g., several different brutes with slightly different attributes, we decided that as far as the difficulty is concerned, we will consider all monsters of the same rank and role to be equivalent.

For each monster we determined a single number, a weight, describing how dangerous the specific monster is. With these we could calculate the enemy group rating by adding together the weights of every enemy in the encounter. And when calculating the difference between two enemy groups, we returned the difference between their ratings.

Originally, we wanted to create a better approximation and tried to create a linear model on the initial matrix that would predict the results of the encounter based on which enemies were in it. Unfortunately, this gave extremely incorrect results. However, we still used the coefficients suggested in the linear model as weights for individual enemies. While these coefficients did not manage to accurately predict the difficulty of an encounter, they did approximately capture how dangerous the monsters are in relation to each other. This made them suitable as weights.

5.2.4 Party difference function

Like the enemy group difference function, this function is used to determine similarity of two encounters. For any two party configurations, this function should return how different the two parties are. Like with the encounter difference function, the lower the returned positive value, the higher the similarity. And zero means the party configurations are the same. And just like with the encounters, the lower the returned value, the higher the likelihood that the two parties will perform just as well against any possible encounter in the game.

In Thesis Quest we calculate the strength of every party member by multiplying his damage by his current maximum HP. And we determine the strength of a party by adding together the strength values of every party member. Difference between parties is the difference between their strengths.

This approach is simple and properly captures party's progress, but it does not perfectly capture the complexity of the party. For example, ranger's low health means that his damage upgrades do not change party strength as much as upgrades for the knight. However, we deemed this approach to be good enough.

5.2.5 Enemy group adjustment function

This function is used to adjust an encounter during the hill climbing algorithm. It takes two arguments, an enemy group to be modified, and a number specifying how much should the function try to shift the difficulty. If the number is positive, the encounter is too easy and should be made more difficult. If it is negative, it should make the encounter easier. The function should then return the modified encounter.

In Thesis Quest, this is the function that is responsible for the variance in tactics and monsters. At the start of each combat it must be configured by doing the following:

- First the algorithm must determine which enemies can be spawned in this encounter. The designer must provide the algorithm with one or more lists of allowed enemies. The algorithm picks one of them. It uses weighted selection – once an enemy list is selected, the unselected candidates become more likely to be selected in the future.
- The algorithm goes through this list of monsters and extracts a list of all rank and role pairs the monsters in the list have. This gives us a list of roles and ranks that can spawn in the encounter.
- Then the algorithm must select the appropriate encounter type. We identified three main points that change how the encounter must be approached. By combining these we defined multiple encounter types. The algorithm selects those for which it has enemies available and picks one of them at random, again preferring those it has not chosen recently. The attributes that define an encounter are:
    - Whether there is a boss or not in the enemy group.

- Whether there is a leader or not in the enemy group.
- How attack oriented are the enemies in this enemy group. For each monster role we defined a constant value specifying how much is the monster attack oriented or defense oriented. For example, a lurker is attack oriented, as he has high damage and tries to kill the weakest party members first. A brute is defense oriented because he has high hit points and low damage.

When an encounter should be made harder, the function will try to find an enemy that would make the encounter closer to the encounter type requested by the configuration, e.g., if the enemy group is too defense oriented, it might add a lurker or a sniper to the encounter.

Similarly, when an encounter should be made easier, this function will remove an enemy. This function will try not to remove a boss or a leader if they are specified in the encounter type.

5.2.6 Initial enemy group generator

This function creates the initial solution for the hill climbing algorithm which is then improved. It should take 2 arguments, the difficulty requested by the designer for this encounter and the current party. It should then return an encounter that should be somewhat in the area of that difficulty. But the algorithms here are not expected to be accurate in any way, it can use as simple method for this as possible. It can even return just an empty encounter, as our algorithm will eventually fit whatever initial encounter to the desired difficulty.

In Thesis Quest the generator first finds any encounter in the matrix with a similar party and difficulty. From this it calculates the target enemy group rating of the enemy group it should return. Then it creates an empty enemy group and starts adding enemies to it using the enemy group adjustment function described above until the rating of the enemy group exceeds the target rating. This is the initial candidate.

5.2.7 Difficulty adjustment function

After the player finishes an encounter, this function will be called for every element in the matrix to adjust it to better fit the player's skill. It will take seven arguments:

- Element difficulty – the current value in the matrix for some encounter.
- Difficulty difference – what was the difference between the estimated difficulty of the just finished encounter and the actual difficulty.
- Element party – the party that this matrix element represents.
- Current encounter party – the party that was fighting in the last encounter.
- Element enemy group – the enemy group that this matrix element represents.
- Current enemy group – the enemy group in the encounter that has just ended.
- Learning factor – A number between 0 and 1 specifying how aggressively should the function adjust the matrix. 0 means no adjustments should be made and the function should return the current encounter difficulty argument, 1 means that the algorithm should change the difficulty as much as possible.

The function should return the new estimate of the difficulty for some encounter and some party configuration.

We will demonstrate how we implemented this function in Thesis Quest via the pseoudocode below. These are the arguments for this function: *elementDifficulty, difficultyDifference, elementParty, currentEncounterParty, elementEnemyGroup, currentEnemyGroup., learningFactor*.

```
partyStrength1 = GetPartyStrength(elementParty)
partyStrength2 = GetPartyStrength(currentEncounterParty)
partyDifference = |partyStrength1 – partyStrength2|
enemyGroupRating1 = GetEnemyGroupRating(elementEnemyGroup)
enemyGroupRating2 = GetEnemyGroupRating(currentEnemyGroup)
enemyGroupDifference = |enemyGroupRating1 – enemyGroupRating2|
partyDifferenceRatio = partyDifference / max(partyStrength1,
partyStrength2)
enemyGroupDifferenceRatio = enemyGroupDifference /
max(enemyGroupRating1, enemyGroupRating2)
similarity = 1 – partyDifferenceRatio –
enemyGroupDifferenceRatio
similarity = similarity < 0 ? 0 : similarity
```

```
return elementDifficulty + similarity * learningFactor *
difficultyDifference
```

In short, we calculate a number between 0 and 1 called similarity and then we modify the original difficulty by adding the error multiplied by similarity and learning factor.

When calculating similarity, we use a ratio between the difference and the larger of the relevant enemy group ratings or party strengths. We do this because a difference between party powers 2000 and 6000 is enormous, while difference between party powers 42000 and 46000 is obviously much lower. This ratio should be a universal way to calculate how similar enemy groups or parties are.

5.3 Algorithm

With all these elements in place, we can now explain in detail how the algorithm works and why.

5.3.1 Difficulty estimation

First, we will explain how to estimate a difficulty of an encounter and a party if we have the difficulty matrix. If the matrix contains an element for that party and that encounter, we can just return the value. If it does not, we do have functions that can measure distance between encounters and parties. Therefore, we can take n closest neighbors of the requested encounter and party and return their weighted average, where the closer the encounter and party to the requested ones, the higher the weight.

5.3.2 Encounter generator

Whenever the game requests an encounter, the following algorithm will be called. Let M be the difficulty matrix, p the current party, *requestedDifficulty* the difficulty requested by the game and Δ be the allowed difficulty error. Then:

```
enemyGroup = InitialEnemyGroupGenerator(requestedDifficulty,
p)
difficulty = M.getDifficulty(enemyGroup , p)
difference = difficulty – requestedDifficulty
while (|difference| > Δ) {
enemyGroup = EnemyGroupAdjustmentFunction(enemyGroup,
difference)
difficulty = M.getDifficulty(encounter, p)
difference = difficulty – requestedDifficulty
}
```

5.3.3 Matrix adjustment

Anytime an encounter is generated, we store in memory the following values: the enemy group e_{fought} and the party p_{fought} from the generated encounter, as well as the original difficulty estimate *difficultyEstimate*. After every encounter we will then measure the actual difficulty of the encounter, *difficultyReal*. The designer should also set learning factor α, which defines how much should the matrix change after an encounter. With these values we can now present an algorithm that will adjust the matrix after every encounter.

```
difficultyError = difficultyEstimate – difficultyReal
foreach (enemyGroup, party, difficulty) in M {
newDifficulty = DifficultyAdjustmentFunction(difficulty,
difficultyError, party, pfought, enemyGroup, efought, α)
M[encounter, party] = newDifficulty
}
M[efought, pfought] = difficultyReal
```

In short, we will adjust the entire matrix based on data and add the fight result as a new element in the matrix. Or replace the existing one if this fight was already added.

5.4 Algorithm analysis

Having described both the algorithm and its individual components, we can now further discuss its properties in further details.

5.4.1 Difficulty targeting

This algorithm focuses mainly on providing a way to measure a difficulty of an encounter. Unfortunately, we think it is impossible to prove mathematically that this algorithm will yield acceptable results.

The problem lies in the fact that game difficulty is not a well-defined variable. There is no mathematical definition so we cannot prove that the algorithm would yield acceptable results. And difficulty will always be a subjective variable. Therefore, it seems clear that real players will be necessary to evaluate the algorithm.

Also, even if we assume that the individual subproblems were solved perfectly, the hill climbing algorithm might not yield the best results, as it might get stuck on local maxima. This is an expected result of our focus on simple solutions described in the Analysis chapter. It would be easy to replace the algorithm with a much more sophisticated solution if we came to conclusion that this solution is unsatisfactory.

5.4.2 Priorities

We can now verify whether the algorithm fits the criteria we outlined in the chapter Analysis. For this section we assume that the algorithm works, i.e., that the after some time the matrix will represent the player's current skill level.

1. The algorithm should work online and adjust itself to player's party configuration and their skill level.
    - We can trivially see that the algorithm works online and that it adjusts itself after every encounter.
2. The designer should be able to influence the encounters generated by the algorithm as much as possible.
    - The main way how the designer can influence the algorithm is by setting a different target difficulty. They can also have great control over concrete enemies being spawned by setting the allowed enemy lists. The programmers could easily extend the algorithm to allow the designer to select specific encounter types as well. And if there are some enemy groups that are too powerful and should never be generated, the programmers could easily create a blacklist of disallowed enemy combinations.

3. The algorithm should be able to output encounters in whatever difficulty the designer desires.
    - This was described in detail in the previous subsection.
4. The algorithm should be generic enough to allow it to be used in many commercial RPG games.
    - The solutions listed in the subproblems section would have to be solved again for every game. Yet we think that the general approach, i.e., having a difficulty matrix and modifying it, could be reused across multiple games.
5. The combat encounters generated should try to vary the enemies generated as much as possible.
    - The enemy group adjustment function is responsible for the variation. And if required, this function is not dependent on anything else and could easily be modified to improve the encounter variance.

6 Testing Methodology

In this chapter we will describe in detail how we wish to evaluate algorithm presented in the previous chapters. First, we will list the goals for the experiment, i.e., what do we wish to learn. After that we will describe our testing scenario in detail. The last section will list the hypotheses which we wish to confirm by the experiment.

6.1 Experiment goals

The experiment should evaluate how the algorithm performs. While we mentioned many different criteria which RPG encounter generation algorithms should fulfill, the focus of this experiment is only on the difficulty. Which means that we will not focus on evaluating enemy variance. Instead we will want to show that:

- As the game progresses, the algorithm should get better at approaching the target difficulty.
- The player should recognize whether an encounter is meant to be difficult or not.
- The player should enjoy the generated encounters.
- The challenge should be appropriate to the player, making him feel in the flow[13].

6.2 Experiment design

6.2.1 Control group

We have introduced a new algorithm for generating enemies and we need to compare it with some other approach. However, as this algorithm is completely new, we cannot compare our solution to some other version of the same algorithm. We have also chosen to create a completely new game, which means that we cannot compare our encounters against those created by the original game designers. Considering all this, we have decided that the best control group available to us are static encounters created by the author.

6.2.2 Experiment phases

Each player playing the game will play the game in following three phases:

- Tutorial phase: A short and easy level created by us. The level exists for two reasons. First, it should teach the player the basic mechanics of the game. Second, while the algorithm is not generating monsters in this level, it is still estimating the difficulty of these static encounters. And when they are over it adjusts the matrix as usual. Therefore, after the tutorial level the matrix should match the player skill more closely.
- Static phase: Two levels with encounters placed in the level manually by the author. Unlike the tutorial phase, the difficulty matrix is not being modified during this phase.
- Generated phase: Two levels with encounters generated at runtime by the algorithm.

The players will be randomly assigned into one of two groups, G-First and S-First. Group S-First will play these in the order they were introduced, group G-First will play the Tutorial phase first, then the generated phase and finally the static phase. Once the player in any group finishes her second phase, the attributes of her heroes will be reset to the same values they had when they finished the tutorial, ensuring that the experiences are as similar as possible.

6.2.3 Door difficulties and colors

To test whether the players can recognize different difficulties, each of the rooms in the experiment has an assigned difficulty. And all doors leading to a room have a different color based on what the difficulty of that room is. These colors change for the third part of the experiment, so the player must figure them out once more. The difficulties are:

	Target difficulty	**Phase 1 and 2 color**	**Phase 3 color**
Easy	0.25	Blue	Pink
Medium	1	Turquoise	Orange
Hard	2	Brown	Yellow

Table 1: Door difficultes

In generated levels, the room difficulty defines the target difficulty for the encounter. For static encounters, we had no concrete method of creating encounters of appropriate difficulty. Therefore, these encounters were based only on the author's understanding of the game design.

6.2.4 Data gathering

We decided to use two methods for gathering data. First, whenever a combat ends, we send a message to our server. The message contains information about the encounter, i.e., which enemies were in the encounter, what was the state of the party, and how did the encounter end. The data is sufficient to recreate the matrix after every single combat encounter, allowing us to see how the matrix was evolving for every player.

Second, after phases 2 and 3, the player is asked to fill out a survey. The survey after the second phase asks the player to rate the difficulty of encounters behind specific doors, to rate the game so far and fill out the Flow Short Scale[14, 15]. After the third phase the player is presented with the same questions. She is also asked to fill out some demographic information and give feedback on the game. If a player fails to finish the experiment, she is presented with another survey, asking her for a reason for quitting. These surveys can be found in the appendix A.

6.2.5 Experiment distribution

The experiment was made into a game that could be distributed with little additional information. It was then distributed to the general public, primarily over social media, to gain as many respondents as possible. To improve engagement the game also included a short humorous story. This story was told between every two levels, as well as at the start and end of the experiment.

6.3 Hypotheses

We have formed these hypotheses which the experiment should be able to prove and which should help us to better understand the behavior of the algorithm:

1. The players should be able to correctly order the doors based on their difficulties in generated levels better than in static levels.
2. The players should rate the dynamic phase as better than the static phase.

3. In the dynamic phase the players should be more in the flow than in the static phase.
4. The perceived difficulty of the game should be closer to 5 (average difficulty) for generated levels than for the static levels.
5. The absolute value of the error in difficulty estimation should be getting lower as the game progresses.
    - As this is a research of a new algorithm, we do not have any other solution to compare these values with. Therefore, we will explore different statistics and see if they suggest the error is lowering over time.

7 Results

In this chapter we will present the results of the experiment. We will show data related to our hypotheses as well as other interesting information gained from the experiment.

7.1 First experiment run

When we ran the experiment for the first time, 31 people in total finished the experiment. However, when analyzing the data, we noticed a major bug in the algorithm implementation. Due to a typo, the algorithm was creating much more difficult encounters than it was supposed to create and did not properly adjust the matrix as it was supposed to. The first version also had an issue where in some rare instances no monsters were generated at all.

Due to these issues we have declared the data from this experiment run as unusable in regards to the evaluation of the effectiveness of the algorithm. The only useful data from this experiment run are the open questions from which we can gain feedback on the game and the experiment. These could be useful if we decide to expand upon this thesis in the future.

7.2 Second experiment run

After we fixed all the issues from first run, we repeated the experiment. 11 people in total finished the experiment. 6 of them were in the G-First group, 5 of them were in the S-First group.

7.3 Result Analysis

In this section we will present the results of the experiment. First we will present raw results, then we will show several different summaries of the data.

7.3.1 Experiment results

Group	Generated levels rating	Generated levels flow	Generated levels difficulty	Generated levels – did order doors correctly?	Static levels rating	Static levels flow	Static levels difficulty	Static levels – did order doors correctly?
G-First	5	5.50	5	1	4	5.40	6	1
G-First	4	5.50	6	1	5	6.30	5	0
G-First	5	5.10	4	0	4	5.30	3	0
G-First	5	5.70	3	1	5	4.70	3	1
G-First	4	3.60	5	1	5	4.10	4	1
G-First	5	6.20	6	1	5	5.10	8	1
S-First	5	5.80	6	0	4	4.90	6	1
S-First	5	6.10	5	1	5	5.90	5	0
S-First	5	5.90	5	1	5	5.70	5	1
S-First	1	3.00	7	0	3	3.20	4	1
S-First	5	6.50	6	1	5	6.20	5	1

Table 2: Result analysis – Experiment results

Here we can see the raw results of the individual players who successfully completed the experiment. First half of the table shows how the users experienced the generated levels. The second half shows their experience with static levels. Values in the column „Did order doors correctly? “are 1 if the player’s rated easy doors as just as difficult or easier than the medium doors and medium doors as easier or the same as hard doors. Otherwise the value in the column is 0.

7.3.2 Comparing experiment groups

Variable	Generated levels rating	Generated levels flow	Generated levels difficulty	Generated levels – did order doors correctly?	Static levels rating	Static levels flow	Static levels difficulty	Static levels – did order doors correctly?
G-First Average	4.67	5.27	4.83	0.83	4.67	5.15	4.83	0.67
S-First Average	4.20	5.46	5.80	0.60	4.40	5.18	5.00	0.80
Total Average	4.45	5.35	5.27	0.73	4.55	5.16	4.91	0.73
G-First St. Dev	0.47	0.81	1.07	0.37	0.47	0.67	1.77	0.47
S-First St. Dev	1.60	1.25	0.75	0.49	0.80	1.08	0.63	0.40
Total St. Dev	1.16	1.04	1.05	0.45	0.66	0.88	1.38	0.45
Pooled St. Dev	1.18	1.06	0.92	0.44	0.66	0.90	1.33	0.44
Cohen's d	-0.39566	0.182984	1.048840027	-0.53609	-0.40614	0.033353	0.125294	0.304997

Table 3: Resuts analysis – Comparing experiment groups

This table shows the average and standard deviation of all the variables from the previous table for the entire experiment and for each group separately. For all these values we then calculate Cohen's d to measure the effect size. Note the large difference in perceived difficulty of levels between groups. For some reason, the G-First group found the generated levels to be much easier.

7.3.3 Comparing second and third phase

	Rating	Flow	Difficulty	Did order doors correctly?
Second phase Mean	4.55	5.23	4.91	0.82
Third phase Mean	4.45	5.29	5.27	0.64
Second phase St. Dev	0.66	0.94	0.90	0.39
Third phase St. Dev	1.16	0.99	1.48	0.48
Pooled St. Dev	0.940371	0.968854	1.226431	0.435985
Cohen's d	-0.10	0.07	0.30	-0.42

Table 4: Result analysis - Comparing second and third phase

Next, we decided to compare how the users experienced the second phase of the experiment compared to the third phase. We wanted to see if there was a significant difference, e.g., if the users found the third phase to be easier, as they probably got better at the game. However, we saw no significant difference here.

7.3.4 Comparing generated and static levels

	Rating	Flow	Difficulty	Did order doors correctly?
Generated levels Mean	4.45	5.35	5.27	0.73
Static levels Mean	4.55	5.16	4.91	0.73
Generated levels St. Dev	1.16	1.04	1.05	0.45
Static levels St. Dev	0.66	0.88	1.38	0.45
Pooled St. Dev	0.94	0.96	1.23	0.45
Cohen's d	0.10	-0.20	-0.30	0.00

Table 5: Result analysis – Comparing generated and static levels

The focus of the experiment was on evaluating the algorithm, therefore the comparison between the generated and static levels is the most important one. It seems that there was not a significant difference between the player's experience in generated and static levels.

7.4 Hypotheses testing

In this section we will test the hypotheses that we explained in the previous section.

1. The players should be able to correctly order the doors based on their difficulties in generated levels better than in static levels.
    - H_0: Generated levels – did order doors correctly? = Static levels – did order doors correctly?
    - H_A: Generated levels – did order doors correctly? > Static levels – did order doors correctly?
    - The p-value for the paired sample t-test for our data is 0.5, therefore we have failed to reject the null hypothesis.
2. The players should rate the dynamic phase as better than the static phase.
    - H_0: Generated levels rating = Static levels rating
    - H_A: Generated levels rating > Static levels rating
    - The p-value for the paired sample t-test for our data is 0.62, therefore we have failed to reject the null hypothesis.
3. In the dynamic phase the players should be more in the flow than in the static phase.
    - H_0: Generated levels flow = Static levels flow
    - H_A: Generated levels flow > Static levels flow
    - The p-value for the paired sample t-test for our data is 0.16, therefore we have failed to reject the null hypothesis.
4. The perceived difficulty of the game should be closer to 5 (average difficulty) for generated levels than for the static levels.
    - H_0: |Generated levels difficulty – 5| = |Static levels difficulty – 5|
    - H_A: : |Generated levels difficulty – 5| < |Static levels difficulty – 5|
    - The p-value for the paired sample t-test for our data is 0.28, therefore we have failed to reject the null hypothesis.
5. The absolute value of the error in difficulty estimation should be getting lower as the game progresses.
    - As we do not know how to properly evaluate this hypothesis, in the next subsection we will show several interesting statistics about it.

7.4.1 Evaluating the difficulty error decreasing over time hypothesis

First, to get an intuitive understanding of how the difficulty error behaved, we decided to plot the difficulty error change over time for several specific players. For all these graphs, positive error means the combat was easier than estimated, negative error means it was more difficult.

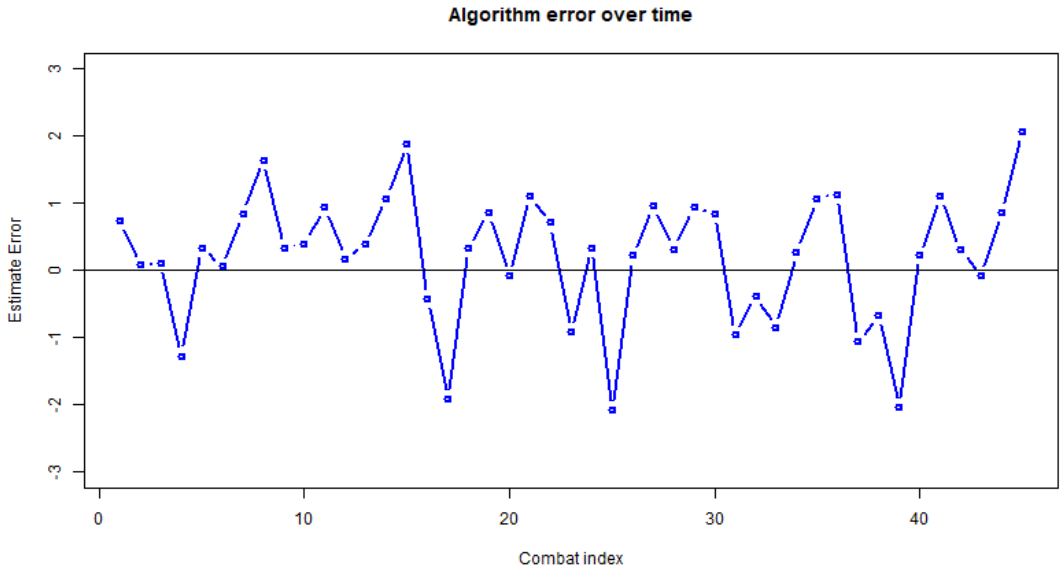

Figure 4: Difficulty error over time – highest difficulty

This graph shows the difficulty error development of player c898d197-387d-45bc-bc10-3f0675438637, whose perceived difficulty of generated levels was the highest of all players, 7. We can see that for this player the average error was constantly changing, therefore for this player the algorithm seems to have failed.

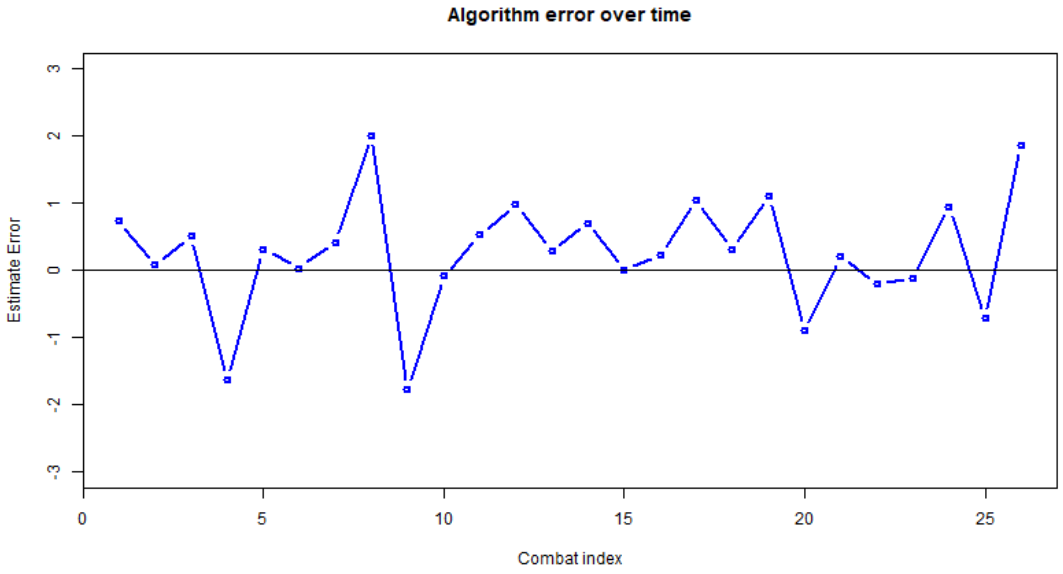

Figure 5: Difficulty error over time - lowest difficulty

This graph represents the playthrough of player 2ab2cade-4c66-49ac-86d7-531f14b6731c, whose difficulty rating was the lowest of all players, 3. Apart from few large spikes in the first 9 encounters, the algorithm seemed to be quite close when estimating difficulty and when there was a large spike near the end, it was because the encounter was too easy, which probably explains the low difficulty rating.

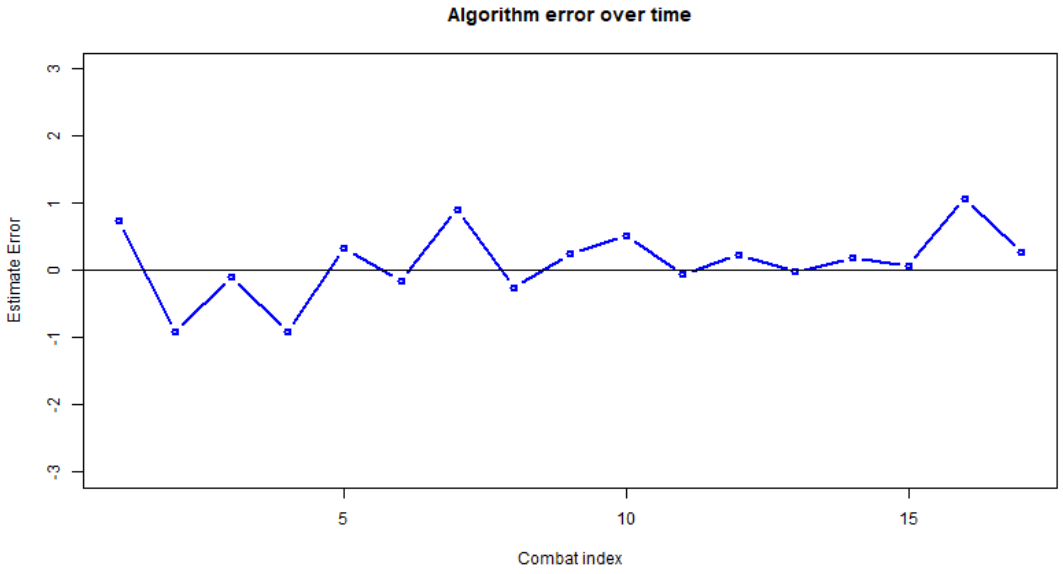

Figure 6: Difficulty error over time - average difficulty

This is a graph of player 598b1b49-73d1-4757-be05-b9c70b9a085b, who rated the difficulty as 5, i.e., average. For this player, the algorithm seemed to perform exactly as expected – after first five encounters it adjusted itself to the player's ability almost perfectly with only very minor errors.

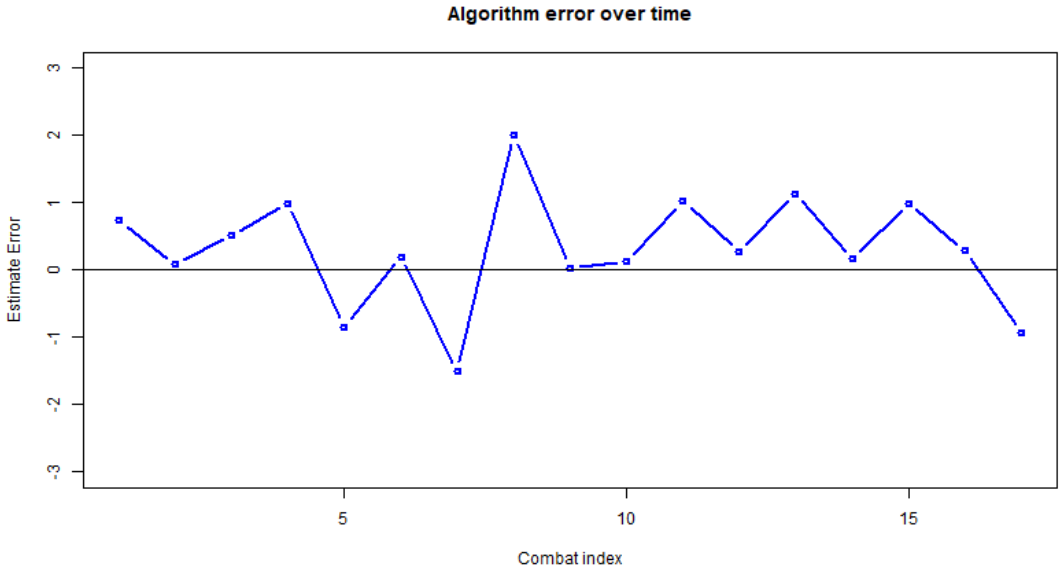

Figure 7:Difficulty error over time – unfinished experiment player

This is a graph of player a543bd42-5e2a-47fd-965a-9eddf33ccccd, who did not finish the experiment. Most of the time the error seems to lie between 0 and 1. And once more we can see that the error is not decreasing.

To compare the data of all successful players, for each player we split all their encounter results into halves and quarters and calculated the average error in these parts. Ideally, the average error should be decreasing in later halves and quarters. In this table we considered only the absolute value of the error, so it does not matter whether the algorithm overestimated or underestimated the difficulty.

Group	Generated levels difficulty	Avg. Error	Avg. Error Half 1	Avg. Error Half 2	Avg. Error Quarter 1	Avg. Error Quarter 2	Avg. Error Quarter 3	Avg. Error Quarter 4
G-First	5	0.56	0.43	0.71	0.39	0.48	0.67	0.75
G-First	6	0.64	0.57	0.70	0.48	0.71	0.51	0.86
G-First	4	0.68	0.62	0.74	0.50	0.76	0.60	0.88
G-First	3	0.68	0.72	0.64	0.53	0.91	0.60	0.68
G-First	5	0.41	0.51	0.30	0.60	0.39	0.21	0.40
G-First	6	0.41	0.45	0.37	0.44	0.46	0.58	0.16
S-First	6	0.71	0.66	0.76	0.70	0.61	0.64	0.88
S-First	5	0.58	0.54	0.63	0.52	0.55	0.92	0.35
S-First	5	0.68	0.57	0.81	0.45	0.71	0.75	0.87
S-First	7	0.79	0.72	0.85	0.58	0.88	0.74	0.96
S-First	6	0.76	0.71	0.82	0.61	0.84	0.83	0.81

Table 6: Algorithm average error

However, it seems this was not the case. For all but three players the average error in the first half was lower than the average error in the second half. And there are only two players whose average error in the last quarter of the encounters was lower than in all the other quarters.

7.5 Additional results

These are all the results that could be easily summarized with tables and graphs. However, there are also data which could not be easily shown in this thesis and are provided on the attached DVD. Of note are the answers to the open question of all our respondents which contain valuable information.

We have also visualized the development of the difficulty matrix for all players over time. The visualization looks like this:

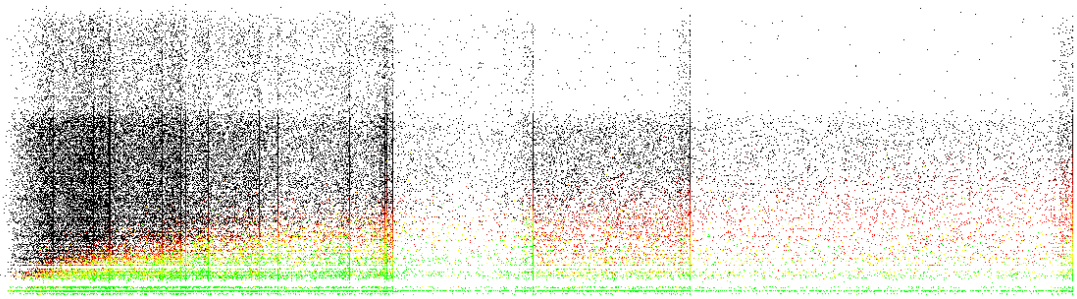

Figure 8: Initial matrix visualization

This is a representation of the matrix when the game starts. Each element of the matrix is placed here and colored based on its difficulty. The x coordinate of the element is the party power, the y coordinate is the combined strength of the monsters. Green is difficulty 0, yellow is difficulty 1, red is 2 and black is 3.

For each player we have generated these visualizations of the matrix after every single encounter. The folders with the visualizations also contain a log file showing which encounter each of these visualizations represent. The matrix visualizations with the log file together present a detailed account of the player's experience in the generated encounters and how the algorithm adjusted to the player. Unfortunately, these are not summarized easily, so they can only be found on the DVD.

8 Discussion

In this chapter we will discuss the results of the experiment, as well as how could this thesis be expanded upon in the future.

8.1 Algorithm evaluation

We could not prove any of our hypotheses regarding the algorithm's performance. Therefore, we can only evaluate the algorithm by comparing Cohen's d effect size and the means of the different variables.

When ignoring the experiment groups and looking only at the comparison between static levels and generated levels, the data suggest that there is not a significant difference between the two. The largest difference is that the generated levels were slightly more difficult than static levels. The difference between means for this variable was 0.36. This seems to suggest that there was not a major difference between the static and generated levels.

But we should also consider that there were 37 people who did not finish the game, 10 of whom listed as their reason for quitting that the game was too difficult. Unfortunately, as we do not have the user ids of the players who failed to finish the game, we cannot be certain about their reasons and whether they quit in the generated or the static levels. Also, without pairing the survey with the gameplay data we cannot determine which survey results belong to which test run. Therefore, we cannot tell how many of the users who quit played the first version of the game, in which the algorithm was bugged.

Another interesting point is that the people who played the generated levels first found them to be easier that the group which started with the static levels. We think the most likely explanation is that the users who started with the static levels were more used to the game by the point they got to the generated levels. Therefore, they made fewer mistakes, built their characters better and the algorithm presented them with more difficult encounters. However, this is only a hypothesis and we have not gathered any data which could support this hypothesis.

Without being able to properly analyze the results of the users who did not finish the game, the only thing we can say is that the results suggest that the algorithm can generate encounters similar in quality to the encounters created by the author.

8.2 Experiment evaluation

In retrospect we have made several mistakes in the experiment design.

First, the game was not tested extensively. Apart from the author, only two people played the game before the release, the thesis supervisor and the experiment consultant. Both of whom were involved with the experiment before they played the game. As we expected a small number of players, we did want to disqualify some of them from the experiment by having them playtest the game. However, this led to several critical bugs not being detected before the experiment launch. And because of these bugs we had to restart the experiment, losing a lot more test subjects.

Second, we have not provided sufficient introduction to the players. From the open questions we can see that many players failed to learn the basics of the game. Namely:

- How health works. Many players thought that they found a bug when they could not heal their lost Max HP.
- That they must use the knight's abilities to taunt monsters. Some players failed to defeat the first boss because they did not taunt him, and they could not progress.
- That when the enemy cleric orders his allies to target the ranger, it overrides the Taunt skill used by the knight. Because of this, many players said that the taunt seems to randomly not work and were frustrated by it.

All these issues could have been be easily fixed had we known about them, either by UI fixes or by hints on the game over screen. But, as we did not play test properly, we did not know about them.

Third, we have gathered our players on social media. Therefore, most of our players had known the author of the thesis personally before playing. Which means that we cannot be sure whether their rating and feedback is truly objective.

Fourth, the hypothesis about the players correctly ordering the difficulty of the doors might have been flawed. The first room in every floor was always an easy room and the last room was always a difficult room, a boss fight. The boss fight was also visibly different from the rest of the rooms, as it featured a different music and had a unique room layout. It is conceivable that some players could have ordered the room difficulties based on the level layout and the conventions of the genre instead by evaluating their actual difficulty.

Fifth, we used the same door colors in the tutorial section as in the second phase. This could have made it easier for the players in the second phase to estimate the door colors, as almost all rooms in the tutorial phase were easy.

And last, we were comparing our algorithm with a static level created by the author. While we did not have a better option available, right now we cannot say much about the quality of the algorithm.

However, for all its problems the experiment had its strengths. Players rated the game generally favorably. We have also gained a lot of data about the algorithm and its performance. The data can be used by further researchers to improve the algorithm.

Also, we got a lot of feedback on how to improve the game. Should another researcher expand upon it, she could use it to make a game that would be more accessible to the general public. This would then help her get a lot more test subjects who would play the game.

8.3 Future work

We think that the algorithm seems to be promising. While it cannot be used in any real game in its current form, we think that the experiment results suggest that it has a potential to be improved and used in a commercial product. We think so because overall the encounters were of the same quality as the static ones and there was at least one person for whom the algorithm worked as intended. In general, we see two ways how future researchers could expand upon this work – they could implement the algorithm in a commercial game, or they could improve the algorithm in Thesis Quest.

8.3.1 Commercial game

There are two main ways how this algorithm could be implemented in a commercial product – by adding it to an existing game or by implementing it in a new game being produced. However, as this algorithm was not yet proven to give great results, we think it is quite unlikely that a game studio would risk their game's success on a new algorithm. Therefore, the more likely future work in this direction would be adapting the algorithm to an existing game.

The researcher going in this direction could use the general idea behind this algorithm. However, he would have to start from scratch with most of the game specific components of the algorithm. He would have to create abstractions to represent the enemies and the party, as there would be much greater variety of characters than in this small game. He would also need to figure out how to update the matrix, and to do that, he would need to come up with a function that could compare vastly different parties and enemy groups.

While this would be a great amount of work, in the end this could prove that the algorithm is useful in commercial games. Also, it would be possible to compare the generated encounters with encounters created by the original designers. That could show whether the algorithm can create encounters that would be better than those of professional designers.

8.3.2 Small game

Should the researcher choose to work on a smaller project, she would have two approaches available. She could either improve Thesis Quest, or she could create another game entirely. We will focus only on extending Thesis Quest, as we see that as a more efficient approach.

While there are many gameplay elements that could be improved, we will not focus on those. Instead we direct the reader to the attached DVD where they can find the survey responses of our players. Many possible adjustments can be seen in the feedback of our players, e.g., they would like a better and shorter tutorial, improved UI or controls.

Instead we will focus on the algorithm. And the potential for improvements in this area is clear – for all the subproblems the algorithm solves we selected the simplest

solution available. Therefore, further research could build upon these and find better solutions. For example, when comparing the encounters, the algorithm right now compares only the sum of the monster strengths. However, when a leader is present and the player does not know how to deal with him targeting the ranger, the leader can increase the difficulty of the battle significantly, especially when combined with multiple sniper enemies. Therefore, the algorithm should take composition of enemies and how they complement each other into account.

An algorithm with these modifications could then be compared to the algorithm without them. Furthermore, these modifications could be turned off and on easily, which could help the researcher find out which changes are improvements, and which do not help.

8.3.3 Inclusion in larger systems

The algorithm could also be improved in other ways both in a commercial game and in Thesis Quest. In general, encounter generation is only a small part of the game and must work together with all the other game systems.

For example, instead of a designer specifying the parameters of the generator, the parameters could be set by another system that would try to control the player experience. An example of this system would be the AI Director system in the game Left 4 Dead 2 [18]. The AI director tries to control different aspects of the game in order to create unique experiences that keep the player engaged. If such a system worked together with this algorithm, it would try to generate encounters that would sometimes be easy, sometimes challenging, whatever the AI Director would feel is necessary. Furthermore, whenever the algorithm would make a significant error the AI Director could fix the issue – if the encounter were too difficult, there could be more rewards from it and the next encounter might be easier.

Or the algorithm could work together with a system that could determine a person's playstyle and preferences, like the algorithm described by Missura and Gärtner[12]. The algorithm could then work differently based on the player's preferences. For example, it might use a different initial matrix more representative of the player's strategies. Or it might change the learning factor, changing how responsive the matrix is to the player's results.

But even if the game does not use these complex systems, it would need some way to adjust the difficulty. Some players want to breeze through the game while others want to struggle through every encounter. Most commonly this is solved by letting the player choose a difficulty setting. Which means that the algorithm must have some way of responding to the difficulty. In the game we presented, this could be accomplished by simply changing the target difficulty of the individual rooms. However, the algorithm might also use a different initial matrix for higher difficulties. By using a different matrix, e.g., one generated with heroes controlled by a smarter AI, the matrix could then match the player's skill much more quickly.

Conclusion

In this thesis we explored and formalized the problem of generating combat encounters in RPG games. We created a small game which approximated RPG games while greatly reducing their complexity. In this game we implemented a new algorithm for generating encounters. We tried to implement this algorithm in the simplest possible way, so we could evaluate the general idea behind the algorithm. We then distributed the game to the general public to evaluate our solution. The data we gathered from this experiment failed to prove any of our hypotheses, however they also suggest that the encounters generated by the algorithm are just as good as the encounters manually defined by the author of this thesis. While the algorithm in its current form is probably not ready to be used in new commercial games, we believe that this thesis is a good starting point into further research extending this algorithm, either by improving it in the context of our game or by implementing it in an older commercial game.

Bibliography

[1] SHAKER, Noor, Julian TOGELIUS and Mark J. NELSON. *Procedural Content Generation in Games* [online]. Cham: Springer International Publishing, 2016 [accessed. 2020-06-16]. Computational Synthesis and Creative Systems. ISBN 978-3-319-42716-4. Available at: doi:10.1007/978-3-319-42716-4

[2] TOGELIUS, Julian, Georgios N. YANNAKAKIS, Kenneth O. STANLEY and Cameron BROWNE. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2011, **3**(3), 172–186. ISSN 1943-068X. Available at: doi:10.1109/TCIAIG.2011.2148116

[3] *Dragon Age™: Origins* [online]. [accessed. 2020-06-17]. Available at: https://www.ea.com/games/dragon-age/dragon-age-origins

[4] *Dragon Age 2* [online]. [accessed. 2020-07-08]. Available at: https://www.ea.com/games/dragon-age/dragon-age-2

[5] VAN DER LINDEN, Roland, Ricardo LOPES and Rafael BIDARRA. Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2014, **6**(1), 78–89. ISSN 1943-0698. Available at: doi:10.1109/TCIAIG.2013.2290371

[6] NEPOŽITEK, Ondřej. Procedural 2D Map Generation for Computer Games. no date.

[7] VANHATUPA, Juha-Matti. Guidelines for personalizing the player experience in computer role-playing games. In: *Proceedings of the 6th International Conference on Foundations of Digital Games* [online]. Bordeaux, France: Association for Computing Machinery, 2011, p. 46–52 [accessed. 2020-06-15]. FDG '11. ISBN 978-1-4503-0804-5. Available at: doi:10.1145/2159365.2159372

[8] DORAN, Jonathon and Ian PARBERRY. Towards procedural quest generation: A structural analysis of RPG quests. *Dept. Comput. Sci. Eng., Univ. North Texas, Tech. Rep. LARC-2010-02*. 2010.

[9] SMITH, Gillian, Mike TREANOR, Jim WHITEHEAD and Michael MATEAS. Rhythm-based level generation for 2D platformers. In: *Proceedings of the 4th International Conference on Foundations of Digital Games* [online]. Orlando, Florida: Association for Computing Machinery, 2009, p. 175–182 [accessed. 2020-06-16]. FDG '09. ISBN 978-1-60558-437-9. Available at: doi:10.1145/1536513.1536548

[10] SHAKER, Noor, Antonios LIAPIS, Julian TOGELIUS, Ricardo LOPES and Rafael BIDARRA. Constructive generation methods for dungeons and levels. In: Noor SHAKER, Julian TOGELIUS and Mark J. NELSON, eds. *Procedural Content Generation in Games* [online]. Cham: Springer International Publishing, 2016 [accessed. 2020-06-16], Computational Synthesis and Creative Systems, p. 31–55. ISBN 978-3-319-42716-4. Available at: doi:10.1007/978-3-319-42716-4_3

[11] XUE, Su, Meng WU, John KOLEN, Navid AGHDAIE and Kazi A. ZAMAN. Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games. In: *Proceedings of the 26th International Conference on World Wide Web Companion* [online]. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, p. 465–471 [accessed. 2020-06-15]. WWW '17 Companion. ISBN 978-1-4503-4914-7. Available at: doi:10.1145/3041021.3054170

[12] MISSURA, Olana and Thomas GÄRTNER. Player Modeling for Intelligent Difficulty Adjustment. In: João GAMA, Vítor Santos COSTA, Alípio Mário JORGE and Pavel B. BRAZDIL, eds. *Discovery Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, p. 197–211. ISBN 978-3-642-04747-3.

[13] *Optimal experience: Psychological studies of flow in consciousness*. New York, NY, US: Cambridge University Press, 1988. Optimal experience: Psychological studies of flow in consciousness. ISBN 978-0-521-34288-9.

[14] ENGESER, Stefan and Falko RHEINBERG. Flow, performance and moderators of challenge-skill balance. *Motivation and Emotion* [online]. 2008, **32**(3), 158–172. ISSN 0146-7239, 1573-6644. Available at: doi:10.1007/s11031-008-9102-4

[15] RHEINBERG, Falko, Regina VOLLMEYER and Stefan ENGESER. Die Erfassung des Flow-Erlebens. no date, 21.

[16] *Games | Obsidian Entertainment* [online]. [accessed. 2020-06-17]. Available at: https://www.obsidian.net/games#nwn2

[17] WYATT, James. *Dungeon & Dragons: Dungeon Master's Guide - Roleplaying Game Core Rules, 4th Edition*. B.m.: Wizards of the Coast, 2008. ISBN 978-0-7869-4880-2.

[18] *Left 4 Dead 2 - GAME* [online]. [accessed. 2020-07-20]. Available at: https://www.l4d.com/game.html

List of tables and figures

Appendix A – Survey Questions

In this section we will list all three surveys the players had to fill out during the experiment.

Phase two survey

This survey was displayed after the player finished the phase two of the experiment.

Survey title – Halfway there!

First page

Please fill out a survey about the first half of the game

1. User ID (meant to pair responses with the game data)
2. How did you enjoy the first half of the game?
    - 1-5 stars
3. How would you rate the difficulty of combat encounters in rooms with CYAN doors? (picture of the doors from the game)
    - 1 (very easy) – 10 (very difficult)
4. How would you rate the difficulty of combat encounters in rooms with BROWN doors? (picture of the doors from the game)
    - 1 (very easy) – 10 (very difficult)
5. How would you rate the difficulty of combat encounters in rooms with BLUE doors? (picture of the doors from the game)
    - 1 (very easy) – 10 (very difficult)

Second page – Flow[14, 15]

Please check how much you agree with several statements about how much in the flow you were during the first half of the game.

6. I feel just the right amount of challenge.
    - 1 (not at all) – 7 (very much)
7. My thoughts/activities run fluidly and smoothly.
    - 1 (not at all) – 7 (very much)
8. I don’t notice the time passing.

- 1 (not at all) – 7 (very much)

9. I have no difficulty concentrating.
    - 1 (not at all) – 7 (very much)
10. My mind is completely clear.
    - 1 (not at all) – 7 (very much)
11. I am totally absorbed in what I am doing.
    - 1 (not at all) – 7 (very much)
12. The right thoughts/movements occur of their own accord.
    - 1 (not at all) – 7 (very much)
13. I know what I have to do each step of the way.
    - 1 (not at all) – 7 (very much)
14. I feel that I have everything under control.
    - 1 (not at all) – 7 (very much)
15. I am completely lost in thought.
    - 1 (not at all) – 7 (very much)

Third page – Overall difficulty

Please answer some general questions regarding the overall difficulty of the first half of the game.

16. Compared to all other activities which I partake in, this one is …
    - 1 (easy) – 9 (difficult)
17. I think my competence in this area is …
    - 1 (low) – 9 (high)
18. For me personally, the current demands are …
    - 1 (too low) – 9 (too high)

Phase three survey

First 18 questions are the same as from the phase two, only asking about the second half of the game and about different colors, yellow, pink and orange. After that, the following pages are displayed:

Survey title – Almost done!

Fourth page – Demographic Questions

19. What is your gender?
    - Male
    - Female
    - Prefer not to say.
20. What is your age?
    - Under 18
    - 18-24
    - 25-34
    - 35-44
    - 45-54
    - 55-64
    - Age 65 and older
21. What is your highest level of education you have completed?
    - None
    - Elementary School
    - High School
    - Bachelor's Degree
    - Master's Degree
    - Postgraduate
22. Which of the following games did you play?
    - Multiple choices allowed:
        - Baldur's Gate 1 or 2
        - Icewind Dale 1 or 2
        - Planescape Torment
        - Neverwinter Nights 1 or 2
        - The Temple of Elemental Evil
        - Dragon Age 1, 2 or Inquisition
        - Divinity: Original Sin 1 or 2
        - Fallout 1, 2 or Brotherhood
        - Fallout 3, 4 or New Vegas
        - The Elder Scrolls – any game in the series
        - The Witcher 1, 2 or 3
23. On average, how many hours per week do you spend playing video games?
    - Less than one hour

- 1 – 2 hours
- 2 – 5 hours
- 5 – 10 hours
- 10 – 20 hours
- 20 – 30 hours
- 30 – 40 hours
- More than 40 hours

24. Please list the last three games you have played.
    - Open question
25. Please list your favourite video game genres.
    - Open question

Fifth page – Feedback

26. Would you like to see more of this game, see it extended, with better level design, more levels and such?
    - Sure, and I might be even willing to pay for it!
    - If it were free, sure.
    - No, I do not think this game would interest me even if it were free.
    - Other – please specify.
27. What do you feel should be improved in the full game? Which area did you find the most lacking?
    - Open question.
28. Do you have anything more to say to us? You can write anything here, this is an open question for the developer to get feedback on the game, the experiment or anything else you might want to write in here.
    - Open question.

Premature exit survey

This survey appears when the play closes the game during an experiment for any reason other than revoking her privacy agreement.

Survey title – So sad to see you go!

Survey subtitle – Hey there! It seems you've exited the game before finishing the experiment. Would you mind filling a short survey for us, so we can understand why you chose to do that?

1. What was the main reason why you shut down the game?
    - I closed it accidentally.
    - I ran out of time.
    - I do not enjoy playing this kind of games.
    - I did not enjoy playing this game in particular.
    - The game was too difficult.
    - I only wanted to try out the game, I did not plan to finish it.
    - Other – please specify.
2. If you want to, please elaborate on your decision to quit the game.
    - Open question.
3. What is your gender?
    - Female
    - Male
    - Prefer not to say
4. What is your age?
    - Under 18
    - 18 – 24
    - 25 – 34
    - 35 – 44
    - 45 – 54
    - 55 – 64
    - Age 65 and older
5. What is your highest level of education you have completed?
    - None
    - Elementary School
    - High School
    - Bachelor's Degree
    - Master's Degree
    - Postgraduate

Appendix B – Technical documentation

In this appendix we will explain the technical details about the Thesis Quest game. In the first section of this chapter we will explain how to run the game, how to set up the development environment and how to run the analysis of the data. In the second section we will briefly introduce the architecture of the project, which should help reader understand the source code.

Throughout this chapter we will assume that the reader has basic familiarity with the Unity game engine. If the reader needs to, there are many great resources online that can help the reader learn the basics[3].

Installation Instructions

Thesis Quest installation instructions

It is not necessary to install Thesis Quest in any way. In the folder Binaries/Thesis Quest you will find Windows, macOS and Linux versions of the game which were distributed during the experiment. Select the appropriate zip file for your system, extract it, and play the game.

If you wish to see how the matrix generation worked, you can find the combat simulator in the folder Binaries/Encounter Simulator. For this simulator we provided only the Windows build, as we could not test it on any other platform. The encounters will start playing one by one and a file TestResults.csv will be generated next to the executable with the results of the experiment. The simulator was done for development purposes, so it has no UI and can be only exited by closing the application, e.g., by pressing ALT+F4 during the game.

Setting up the development environment

To run the Unity project used to build these executables, you will need to do the following steps. If any of the applications listed in the guide are already installed, you do not need to reinstall them.

[3] https://learn.unity.com/course/getting-started-with-unity

1. Copy all the files from the DVD to some location on your hard drive.
2. Install Visual Studio 2019[4], any edition.
3. Install Unity 2019.2.12. This version can now be only installed from the download archives[5].
4. Download the LITE - Fantasy - Headstrong Archer Character Voice Sound Pack[6] and extract it to the folder Source/Assets/Sounds/Voices/Ranger. There should already be a folder with the same name present in this location.
    - For every file in the sound pack, a file with the same name and the .meta extension will already exist in that existing folder. Make sure that the extracted files are in the same folder as the corresponding .meta file.
5. Open Unity
6. Open the project located in the Source folder.
7. Select Window -> TextMeshPro -> Import TMP Essential Resources
8. Download the following assets from the Unity Asset Store. These are optional and some of these are paid. If the reader does not download them, the game will work, but relevant sound effects and music will not play:
    - Action RPG Music Free[7]
    - Medieval Combat Sounds[8]
    - Magic Spells Sound Effects[9]
        - Unfortunately, since we implemented these sound effects in the game, the distributor of this pack released a 2.0 version of this pack with a different file structure. Therefore, spell effect sounds will not work. The only way to solve this issue is to ask the developer for the old version of the assets. And we are unsure whether the developer will agree to providing this older version.

---

[4] https://visualstudio.microsoft.com/cs/vs/

[5] https://unity3d.com/get-unity/download/archive

[6] https://bobfeeservo.itch.io/lite-fantasy-headstrong-archer-character-voice-sound-pack

[7] https://assetstore.unity.com/packages/audio/music/action-rpg-music-free-85434

[8] https://assetstore.unity.com/packages/audio/sound-fx/weapons/medieval-combat-sounds-149395

[9] https://assetstore.unity.com/packages/audio/sound-fx/magic-spells-sound-effects-114628

Running results analysis

This process works only on Windows. If you wish to run the scripts for analyzing the results of the thesis, you need to set up the development environment as described in the previous section. After that you need to do the following steps:

1. Download and install R[10].
2. Add the folder where R executable is located to your PATH variable.
3. Copy the entire Results folder from the DVD to the source folder of your project.
4. If you want to see how the files are created from scratch, delete the folder Source/Results/Processed.
5. Open the Unity project and open the scene Scenes/AnalysisHelperScene.
6. Start play mode.
7. Wait until all the objects delete themselves and the only object in the scene is Main Camera. After that you can stop the game.
8. Run the PowerShell script DrawAllAlgorithmErrorGraphs.ps1 located in the source folder.
    - If the script fails to run, you might need to change your execution policy. Open PowerShell as administrator and run the following command:
      Set-ExecutionPolicy unrestricted
9. Open the generated results file Source/Results/Processed/Summary/SuccessfulUsersSummary.csv. This should be the same file as the one used for the analysis in the folder Results/SummaryAnalysis/SuccessfulUsersSummary.csv.

Note that the entire result analysis was done with no focus on user friendliness and maintainability, as the process was always meant to be single use only. Therefore, some of these steps might take a long time and freeze the UI while they do their work.

[10] https://www.r-project.org/

Project documentation

In this section we will provide a high-level summary of the project. However, we will not go into concrete implementation details that could be easily understood from the source code. Before trying to understand the code, we recommend playing the game, as being familiar with how the game works should help the reader understand what is going on under the hood.

Terminology

There are several terms we use throughout the code which the reader should familiarize herself with.

- Combatant – This is the umbrella term for all enemies and player characters.
- Hero/Player Character – These are used interchangeably in code. They represent a single character the player controls.
- Monster/Enemy – These are also used interchangeably and represent a single enemy that can appear in the game.
- Encounter – throughout the code, we use encounter in the same context we used enemy group in the thesis, i.e., the list of all enemies appearing in an encounter.
- Cutscene – A sequence of events in the game in which the characters do something, and the player has no control over it. In Thesis Quest this is a situation where characters enter a room when it is opened.

Scenes overview

In the folder Source/Assets/Scenes you can find 5 different scenes:

- MainMenu – This is the entry point of the game. It shows the GDPR consent screen and the main menu. It also shows the intro text if the user chooses to start a new game.
- DungeonScene – This is the main scene of the game. When it is loaded, the Map Loader object tries to find a Level Loader Object. This is an object that is not destroyed between scenes and provider information about which map should be loaded. Map Loader generates the appropriate dungeon and spawns the player. Once the player reaches the end of the level, the Level Loader shows

the next story segment, asks the player to fill out a survey if necessary, and loads the next level. That level will be either another DungeonScene or the Credits Scene.

- Credits – This scene shows the end credits of the game. Once the credits are over, the session is reset, new User ID is generated, and the game goes back to the game menu.
- CombatSimulator – Development only scene. This scene is similar to the DungeonScene. It is used to generate a difficulty matrix. When it is loaded, the CombatSimulator object will spawn some enemies and heroes controlled by AI. Once the encounter is over, the object logs the result, removes all enemies and heroes from the game and spawns new ones. This is repeated indefinitely, the user is expected to close the game by force when there are enough results. As this was always meant to be development only, parametrization of the combat simulator can only be done directly in code
- AnalysisHelper – Another development only scene. It exists only to analyze results. This entire process was done in the simplest way possible with no focus on reusability or maintainability, as we do not expect to repeat the analysis in the future.

These scenes use a decentralized architecture. Whenever a component requires some dependency, it searches the entire scene for it, throwing an exception if the dependency could not be found. Not finding a dependency leads to undefined behavior. The functionality of most of these components should be clear from their name, source code, and the reference documentation.

Some of these scenes might be inefficient when used during development. For example, the DungeonScene loads the entire difficulty matrix into memory when the scene launches, which freezes the engine. However, this is not a problem during the actual gameplay, as the matrix is preloaded on a different thread in the main menu before it is needed for the first time.

When trying to get an understanding of these scenes, note that the Generated dungeon object also contains several important components, namely components handling pathfinding, information about room layout and the component responsible

for spawning heroes and enemies. This Generated dungeon object does not exist before runtime and can only be seen in the play mode.

Level generation

For level generation we use the dungeon generator library created by Nepožitek[6]. The documentation for this library is available online[11]. We use an older, v1 version of the library than the one available right now, so some details might be different.

This library requires as its input a graph describing the rooms that should appear and which rooms should be connected to each other. Example of such a graph is the file Source/Scenes/LevelGraphs/TestLevelGraph, which can be only opened directly in Unity. The designer must specify game objects that specify rooms that can appear in the game and corridors that can connect these rooms. We have extended the level graph to allow the designer to set parameters for individual rooms. These parameters specify what kind of an encounter should happen in the room the player enters and what should the rewards be.

This level graph is than passed to the Dungeon Generator object in the DungeonScene. This object contains a pipeline of steps that will be executed in sequence when a dungeon should be generated. We extended this pipeline with custom steps. These are the steps with the "EG" prefix. They do game specific tasks like adding custom objects to the map or storing information about the level layout to be used at runtime.

Note that the architecture of the original library requires us to go through several objects to get to the implementation of each pipeline step. If you double-click on some step in the pipeline, it will open a scriptable object in the project. If you double click on the Script property of the object, a source code will appear, e.g., SpawnObjectConfig. However, this still represents only the data this pipeline step needs. To get to the implementation of this pipeline step, you need to find the matching

[11] https://ondrejnepozitek.github.io/Edgar-Unity/docs/introduction/

task for this config object, e.g., SpawnObjectTask. It is always located in the same folder as the Config class, in the Source/Assets/Scripts/DungeonGenerator folder.

To specify which levels should appear, we created one more scriptable object, LevelDefinition. This object contains additional data about a level, e.g., which story text should appear, a link to the survey that should appear before this level, the graph of the level to generate etc. The Level Loader prefab contains the ordered list of these LevelDefinition objects and opens the next level when it should appear.

Prefabs

Whenever a game object is complex enough or is reused between scenes, we place it in a prefab to improve maintainability. These are stored in the folder Scenes/Assets/Prefabs. Whenever you wish to know how some specific object works in the game, e.g., some specific enemy, try to find the appropriate prefab in this folder and examine it.

Appendix C – DVD contents

In this section we will describe the structure of the DVD that is attached to this thesis.

- Binaries – Contains the executable files relevant to this thesis.
    - Encounter Simulator – The application which generates the initial matrix.
    - Thesis Quest – The binaries of the game itself which were distributed to the players. Each folder contains a zip which must be extracted before playing. Do not try to extract the macOS and Linux builds on Windows, they can contain symlinks which will break on Windows.
        - Linux – The Linux build of the game.
        - Mac – The macOS build of the game.
        - Windows – The Windows build of the game.
- Documentation – The generated reference documentation of the project.
    - index.html – To read the documentation, open this file in a web browser. The generator also created files that would enable search functionality if the site were hosted online. However, we do not officially support that functionality, the documentation is meant to be viewed locally without being hosted anywhere.
- Results – Both raw and processed results of the experiment.
    - Processed – The raw data processed into a more user-friendly form.
        - Summary – Contains only a single file, SuccessfulUsersSummary.csv, that served as the base for the analyzed CSV files in the SummaryAnalysis folder described below.
        - IndividualTests – Contains the results of each separate experiment session. They are split into many subfolders to help the user with finding the relevant data. First the data are split by the version of the experiment (v1 is the first experiment run with bugs, v2 is the fixed version). These folders are split by the experiment group. In addition to the G-First (GeneratedFirst) and S-First (StaticFirst), there are also groups for users who did

not finish the tutorial (TutorialOnly) and for users who experienced the bug which caused the enemies to stop appearing (InvalidValues). The groups are further split into subfolders based on how many levels did the player complete. These subfolders then contain a folder for each test run, with the name of the folder corresponding to the user ID of the player who played the test. Each folder contains some of the following:

- errorGraph.png – The graph describing how the algorithm error developed over time.
- log.txt – User friendly description of the individual encounters in this experiment. Lists only those that affected the matrix.
- rawdata.csv – Contains a subset of data from data.csv described below, specifically, only those rows relevant to the current experiment session. This file is still not human friendly, as it mixes data of different kinds in a single file.
- HalfSurvey.csv – The survey the user filled out in the middle of the experiment.
- Endsurvey.csv – The survey the user filled out at the end of the experiment.
- Unfinished.csv – The survey the user filled out when he did not finish the experiment. We are not sure if these are properly assigned. For each experiment session we found the time of the last event sent to the server and tried to match it with a survey with similar start time. However, this is not 100% reliable.
- VisualizationX.png – X is the index of the encounter. Contains the visualization of the matrix after the Xth encounter. The encounter details can be found in the log.txt file.

- Raw – The files with unprocessed results of the experiment.

        - data.csv – All the data that were uploaded to our server, containing results of every single encounter of every player.
        - GeneratedFirstComplete.csv – Survey results for the G-First group at the end of the experiment.
        - GeneratedFirstHalf.csv – Survey results for the G-First group in the middle of the experiment.
        - StaticFirstComplete.csv – Survey results for the S-First group at the end of the experiment.
        - StaticFirstHalf.csv – Survey results for the S-First group in the middle of the experiment.
        - Unfinished.csv – Survey results of the players who did not finish the experiment.
    - SummaryAnalysis – Analysis of the SuccessfulUsersSummary.csv file described above.
        - AllPlayers_GeneratedVsStaticEncounters.csv – This file compares how all successful players viewed static encounters when compared with the generated encounters.
        - DifficultyHypothesis.csv – This file has details about our testing of the fourth hypothesis described in the section 6.3.
        - DoorOrderHypothesis.csv – This file has details about our testing of the first hypothesis described in the section 6.3.
        - FlowHypothesis.csv – This file has details about our testing of the third hypothesis described in the section 6.3.
        - GeneratedVsStaticFirst.csv – This file compares the experiences if the G-First group with experiences of the S-First group.
        - GFirst_GeneratedVsStaticEncounters.csv – This file compares how players from the G-First group viewed static encounters when compared with the generated encounters.
        - RatingHypothesis.csv – This file has details about our testing of the second hypothesis described in the section 6.3.
        - SecondPhaseVsThirdPhase.csv – This file compares how all successful players viewed the second phase with how they viewed the third phase of the experiment.

        - SFirst_GeneratedVsStaticEncounters.csv – This file compares how players from the S-First group viewed static encounters when compared with the generated encounters.
        - SuccessfulUsersSummary.csv – The source data for this analysis.
- Source – The source files for the game. Described in detail in Appendix B.