Tackling Generation of Combat Encounters in Role-playing Digital Games

Matouš Kozma[1], Vojtěch Černý[1], and Jakub Gemrot[1]

Faculty of Mathematics and Physics, Charles University
Ke Karlovu 3, Praha 2, 121 16, Czech Republic
mattkozma@hotmail.com, {cerny,gemrot}@gamedev.cuni.cz

Abstract: Procedural content generation (PCG) has been used in digital games since the early 1980s. Here we focus on a new problem of generating personalized combat encounters in role playing video games (RPG). A game should provide a player with combat encounters of adequate difficulties, which ideally should be matching the player's performance in order for a game to provide adequate challenge to the player. In this paper, we describe our own reinforcement learning algorithm that estimates difficulties of combat encounters during game runtime, which can be them used to find next suitable combat encounter of desired difficulty in a stochastic hill-climbing manner. After a player finishes the encounter, its result is propagated through the matrix to update the estimations of not only the presented combat encounter, but also similar ones. To test our solution, we conducted a preliminary study with human players on a simplified RPG game we have developed. The data collected suggests our algorithm can adapt the matrix to the player performance fast from little amounts of data, even though not precisely.

Keywords: Procedural generation, Video games, Role-playing games, Encounter generation, Difficulty adaptation

1 Introduction

Many games today implement some form of procedural content generation (PCG) [1, 2], where some parts of a game are generated by an algorithm. PCG is a colloquial term referring to creation of digital content through rules instead by hand. In games, it has been used for generating, e.g. dungeon level layouts (Rogue by Michael Toy and Glenn Wichman, 1980) to entire galaxies (Elite Dangerous by Frontier Developments, 2015). This can greatly reduce the work required from designers or artists. It can improve the replay value of the game, as, e.g., the levels, dungeons, characters, or even dialogues can be generated during playtime to vary a player's game-play experience every run. This paper deals with the PCG of combat encounters for role-playing games (RPGs). Combat encounter in RPG refers to any situation in which a player, who is in control of virtual characters usually called heroes, faces an opposition of enemy non-player characters (NPC) usually called monsters, which result in a combat between characters. The problem of combat encounter generation can be described as how to select a group of monsters that a player should face while keeping the game engaging for the player.

One of the current widely accepted psychological models of engagement in games is flow [3, 4]. Csikszentmihalyi defines flow as:*"A state in which people are so involved in an activity that nothing else seems to matter; the experience is so enjoyable that people will continue to do it even at great cost, for the sheer sake of doing it."* [5]. It is thought that a player can enter the flow state only if a game supplies a challenge that is adequate to the player's ability [6]. Wrt. player skill, if the challenge is too high, a player may experience anxiety, fear of failure or frustration, while if the challenge is too low, a player may experience routine, boredom or loss of interest (Fig. 1).

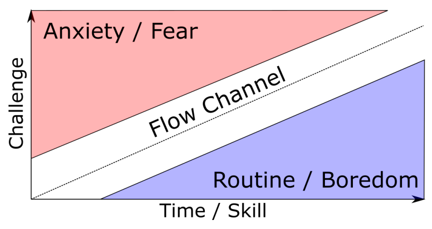

Figure 1: Relation of the player's skill to the challenge provided by the game, depicting the flow channel area.

Focusing on RPG combat encounters, the challenge of their design is how to compose a group of monsters players should face in order for them to stay in the flow channel (Fig. 1). This is problematic as the player's ability is not known when the game starts. Whereas we can fathom how difficult a given combat encounter is, e.g., by statistics over monsters combat values (total health, damage per second, etc.), we do not know how proficient a player is at first or how fast they can improve in playing the game. We propose to solve this problem using reinforcement learning, i.e., to create a system that given the observation of player's performance in the game can generate next combat encounters of appropriate difficulty.

The rest of the paper is structured as follows. In Section 2, we examine work related to ours. In Section 3, we formulate the combat encounter generation problem (CEGP) and requirements on its solutions, namely the ability to adapt to concrete game instances and the fact the solution must be able to adapt to the player fast. In Section 4, we detail our approach tackling CEGP. In Section 5, we present an implementation of our approach for an example

RPG game we have created to test it. In Section 6, we describe and discuss the results of an experiment with human subjects that was conducted to test our approach. Finally in Section 7, we conclude the paper with notes on future work.

2 Related Works

While procedural content generation in games is a well-studied field, it seems that existing research does not focus on generating combat encounters. The works we found about content generation in RPGs focus either on generating levels [7, 8], or quests [9, 10].

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[11] or roguelikes[12] the main focus is on generating the level, not on generating monsters or combat encounters in general.

2.1 Dynamic Difficulty

The encounters we generate must be appropriately difficult for the player. Therefore, techniques for runtime adjustment of difficulty might be relevant to us.

Xue et al.[13] describe the results of adding dynamic difficulty to a match-three game. For our purposes, the most important conclusion is that we should attempt to maximize engagement throughout the entire game, not focus on creating encounters of some specific difficulty.

Missura and Gärtner[14] introduced an algorithm that could split players into groups and then generate challenges specifically for that group. However, this approach required a lot of data about many players playing the game, which we do not have, so this approach could not be used.

2.2 Algorithm Evaluation

As the goal of the algorithm will be generating encounters suitably difficult for the players, its evaluation must involve real people playing the encounters generated by the algorithm. We have chosen to also measure the player's feeling of flow [15] 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.

To measure the player's feeling of flow, we use the Flow Short Scale [16, 17]. This survey measures the flow on a 7-point scale and a person's perceived difficulty of the task. However, these data were not conclusive and we do not report them and focus on difficulty estimation of encounters only.

3 The Problem

Here we define several terms commonly used in RPG games, which we will use throughout the paper; defined terms are in *italics*. Some terms are formulated in an open-ended manner as different games will differ in details.

For the purpose of this paper, we will understand a *game* as a double (W,S) of a *game world* W and a *game state* S.

A *game world* as a double (L,P) of game locations L and actor prototypes P. A game state consists of spawned actors only.

Game locations L is a set of all possible positions in the game world, which actors may occupy. In some games locations could be a grid of discrete tiles, whereas in others it could be a more complex 3D structure. The number of these atomic locations can be in thousands per game level for grid-based games, or infinite in case of 3D worlds where location is represented by floating point numbers.

An *actor prototype*, $p \in P$, is a prototype of a character that can be spawned as an actor into the game. An actor prototype is a tuple of all the attribute values that describe the class of an actor, e.g., its health, damage per second, size, skills, etc. These attributes are game specific, but we assume health and damage per second to be present at least to allow for combat between actors.

Actor, $a = (p_a, s_a, l_a, c_a)$, is a spawned actor prototype that can be present within the game and a part of the game state. It consists of actor prototype p_a definition, its current state s_a of p_a attributes (initial values set according to p_a), location $l_a \in L$ an actor occupies in the world and a controller c_a, which is either a *human* or an *AI*. We label the set of all possible actors as A.

A $hero(r) ::= \{r \in A \mid c_r = human\}$ is an actor a player currently controls. A $party(R) ::= P(\{r \in R \mid hero(r)\})$ is a set of heroes the player may control. All such parties are denoted as ρ.

An $enemy(e) ::= \{e \in A \mid c_e = AI\}$ is an actor a player does not control and the party has to fight. $enemies(E) ::= P(\{e \mid enemy(e)\})$ is then a group of enemies a player may face in the game, all such enemy groups are denoted as ε.

A *combat area*, $O = \{l \in L \mid \forall l_1, l_2 \in O : path(l_1, l_2) \subseteq O\}$, represents some continuous area in the game world where some combat encounter may take place.

A *combat encounter* is a triple, $c = (E_c, R_c, O_c)$, that represents a situation where a party must defeat some enemies. It consists of enemies E_c the party R_c has to fight inside some combat area O_c. The set of all possible combat encounters is labelled as C. In particular, actor prototypes are not the only part of c as also their runtime states are included. As such, C can be thought of as game states in the space of combat; as actors are exchanging damage, this space is traversed until either side is eliminated (their total health reaches zero).

We model the player $y = (s_y)$, as a single number $s_y \in [0;1]$ denoting the skill of the player. 0 means no skill, which can be thought of as not issuing any actions to heroes during any combat encounter. 1 refers to an optimal play, i.e., player can achieve the best outcome possible regardless the encounter given. We denote set of all kinds of player abilities as Y.

We define the *difficulty* of an encounter $c \in C$ as a function $D : C \times Y \to <0; 1>$, where its value states how many percent of party health will be lost during encounter as played through by the player of given skill. 0 means the party R_c will defeat enemies E_c at O_c without any health losses and 1 means the party R_c will certainly be eliminated by the enemies E_c regardless player actions. Importantly, there can be encounters that cannot be won even by optimal players (enemies are too strong for the party to defeat).

The *combat encounter generation problem* (CEGP) is than an optimization problem: given a set of all possible combat encounters C for a given party R, a player y, and a desired difficulty d, find $c_{min} \in C$ such as $\forall c \in C : |d - D(c_{min}, y)| \leq |d - D(c, y)|$.

3.1 Problem Complexity

The CEGP consists of two sub-problems: 1) how to evaluate the difficulty D; 2) even if D is known, how to search for appropriate $c \in C$ fast. The challenge of solving them stems from the problem formulation itself (e.g., the size of C) and from the video game context, which poses additional requirements on the CEGP solution.

Difficulty evaluation. The skill of a player is not known beforehand. As the result, we cannot compute precise values of D even via simulation. Moreover, player's skill can change throughout the game as the player improves in playing the game. Therefore, once observed difficulty of a concrete c may become irrelevant in future as the player skill changes. Finally, if the game is played in real-time, the skill may fluctuate as player's reflexes improve.

Problem size. Considering solving the CEGP for the game Dragon Age: Origins (Bioware, 2009) featuring 140 enemy actor prototypes (according to https://dragonage.fandom.com) trying to generate an optimal encounter that consists of about 7 enemies in the game, we would need to go through 140^7 possible groups of enemies. Moreover, every enemy group can be spawned into the game to a different locations, which could affect difficulty of the encounter greatly. E.g., if a group of archers is spawned behind a strong knight that is blocking the path towards them, it would pose an extra challenge to the player than if the group of archers could be easily reached. The size of C is thus enormous.

Training data. The only direct way to model skill of a player playing an instance of the game is through observation of his combat encounters. Such observations can be seen as training data, provided during the gameplay. However, it is not realistic to expect a player to play even ten tutorial encounters just to calibrate the model of the player, which is still very few still given the size of C.

Video game context. RPG games are (typically) played in real-time and there are usually several to tens of combat encountered prepared by game designers in each game level. Therefore, the algorithm solving the CEGP must compute fast in order not to interrupt the gameplay and it should not consume an unnecessary amount of memory.

4 Methodology

Our current approach to solving the CEGP is to use reinforcement learning for estimating the difficulty function D, while heuristically reducing the size of the combat encounters set C. If we can contract the set, it would make both the estimation of D and the search for suitable $c \in C$ much easier. For looking up an encounter of some required difficulty d, we are softening the requirement of optimality and use stochastic hill-climbing to look for a combat encounter c of difficulty that differs up-to δ from d, i.e., $|d - D(c, y)| < \delta$.

4.1 Estimating Encounter Difficulties

Looking at a combat encounter definition $c = (E_c, R_c, O_c)$, we disregard O_c (simplification) and maintain a sparse *difficulty matrix M* indexed by enemies and a party. The matrix then holds estimated difficulties of a combat encounters M_{er} given enemies E_e and the party R_r for the player y that is currently playing the game.

The size of the matrix is huge. Adapting only single matrix element after each encounter would not result in fast adaptation to the player. Therefore, we propose to adapt multiple values from the matrix given a single observation using some difficulty adjustment function.

A *difficulty adjustment function*, which we denote by $\Delta(d_m, E_m, R_m, d_o, E_o, R_o, \alpha)$ should return a new difficulty value for an encounter between enemies E_m and the party R_m, which was thought to be of d_m difficulty, given the observed difficulty d_o from just finished encounter between enemies E_o and the party R_o; α is a learning factor.

Given the observation of d_o for some encounter $c(E_o, R_o, O_o)$, we update each matrix element with $M_{er} = \Delta(M_{er}, E_e, R_r, d_o, E_o, R_o, \alpha)$.

To be able to construct Δ, we define two heuristic functions, which computes similarities of enemies and parties. *Enemies similarity function* is defined as $S_E : \varepsilon \times \varepsilon \to [0; 1]$ and *party similarity function* is defined as $S_R : \rho \times \rho \to [0; 1]$. 0 value means total difference, while 1 means equality of elements.

We then compute $\Delta(d_m, E_m, R_m, d_o, E_o, R_o, \alpha) = d_m + (d_o - d_m) * max\{0, 1 - S_E(E_m, E_o) - S_R(R_m, R_o)\} * \alpha$. The

```
1: procedure UpdateMatrix($M, d_o, E_o, R_o, \alpha$)
2:     for each $(E_m, R_m) \in M$ do
3:         $d_m = M[E_m, E_m]$
4:         $M[E_m, E_m] = \Delta(d_m, E_m, R_m, d_o, E_o, R_o, \alpha)$
5:     end for
6: end procedure
```

Figure 2: Pseudocode of the method updating difficulty matrix M after each combat encounter result. Inputs are: M - difficulty matrix, d_o - observed difficulty of some encounter that just finished, E_o - initial enemies of the encounter, R_o - party that entered the encounter, α - learning rate.

function updates the estimation of its error scaled according to the similarity of enemies and the party of the matrix element to real enemies and party participating in combat that was just observed (and multiplied by the learning rate α).

Updating matrix is then done by updating all of its elements (Fig. 2) using Δ function. This is the place where similarity functions are used the most as they allow us to update multiple elements of the matrix.

4.2 Difficulty Matrix Initialization

Ideally, initialization of M should be based on real-data by letting players play the game to obtain observations. However, the amount of required observations is usually so high that it is probably unreasonable to require it even from big video game making companies. Instead, we propose to implement a simple reactive player that can play through combat encounters and use it to obtain first difficulty estimations for the matrix. We do this only for a selected subset of enemies and parties.

4.3 Generating an Encounter

Having matrix M estimating D for player y, enemies and party similarity functions S_E, and S_R we can implement stochastic hill-climbing algorithm for finding a combat encounter to present as a challenge to the player.

The algorithm (Fig. 3) is getting the party R, the combat area O and required encounter difficulty d as an input. It is supposed to find a suitable set of enemies E in order to return an encounter $c(E,R,O)$, whose difficulty differs from a required d by an amount, which is smaller or equal to δ. It works in a stochastic hill-climbing manner. It keeps adding/removing random enemies to/from the E trying to match required d. For estimation of D, it queries passed difficulty matrix M. If particular element $M[E,R]$ is not present within M, it uses similarity functions S_E, S_R to pick the most similar element of M. For practical real-time constraints, the algorithm does not perform more then $iter_{max}$ iterations before terminating.

```
1: procedure GenEncounter($R, O, d, M, \delta, iter_{max}$)
2:     $i = 0$
3:     $E_{best} = E_{curr} = \{pick\ random\ enemy\ for\ O\}$
4:     $\delta_{best} = \delta_{curr} = |M[E_{curr}, R] - d|$
5:     while $\delta \leq \delta_{best}$ & $i < iter_{max}$ do
6:         if $M[E_{curr}, R] - d \leq 0$ then
7:             remove random enemy from $E_{curr}$
8:         else
9:             add random enemy for $O$ into $E_{curr}$
10:        end if
11:        $\delta_{curr} = |M[E_{curr}, R] - d|$
12:        if $\delta_{curr} \leq \delta_{best}$ then
13:            $E_{best} = E_{curr}$
14:            $\delta_{best} = \delta_{curr}$
15:        end if
16:        $i++$
17:    end while
18:    return $c(E_{best}, R, O)$
19: end procedure
```

Figure 3: Pseudocode of the method for generating an encounter. Inputs are: R - current party, O - combat area, for which we generate the encounter, d - required difficulty, M - the difficulty matrix, δ - difficulty tolerance, $iter_{max}$ - maximum number of search iterations

5 Implementation

In order to test our approach, we implemented a simple PC game of the RPG genre, which is mouse-controlled. Its design is inspired by gameplay mechanics of well-known, if a bit dated, RPG games such as Baldur's gate (BioWare, 1998) and Planescape Torment (Black Isle Studios, 1999). A player is leading a party of three preset heroes through a dungeon, which is a procedurally generated maze consisting of rooms, which constitute combat areas where encounters with enemies take place. We tried to design the game to be fun and to have a certain gameplay depth. Heroes represent three RPG archetypes: knight (high health, melee attack only), ranger (low health, strong ranged attack) and cleric (healer and area controller). Each hero, apart form the attack action, has three different skills they can use to affect the fight. A player can level heroes between encounters, i.e., improve their health and dealt damage per second. For enemies, there are 4 archetypes. Some of them having multiple instances for variety. Each archetype is designed to behave differently in the combat in order to prolong the learning curve for the player. Additional information can be found in a previous master's thesis [18]. Due to the time and budget constraints, we used some free and some paid sound and graphical assets of the pixel-art style (Fig. 4).

The combat. Every time the party enters a new room, i.e., opens the door, an encounter is generated inside given the difficulty assigned by the designer (us) to the room. Then, the party enters, the door closes behind it, and the party has to fight spawned enemies until one of the sides

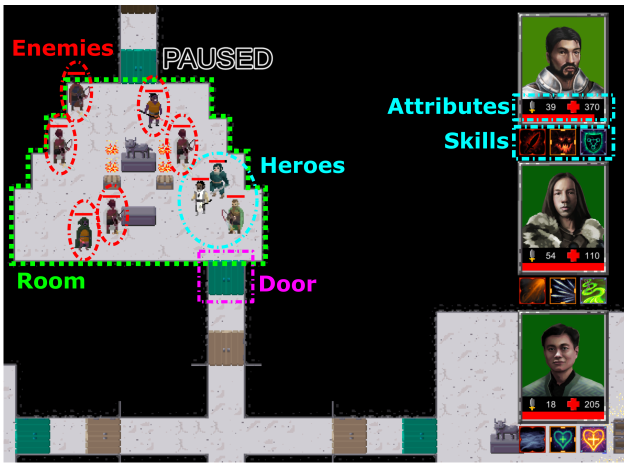

Figure 4: Cropped game screenshot with labels overlay depicting important game elements.

is eliminated. The combat happens in real-time, though the player is able to pause it anytime to analyze the situation and reconsider their heroes' actions. Enemies are driven by a simple reactive AI, that chooses their attack according to a script associated with the enemy archetype. Heroes also attack automatically if player does not intervene, i.e., orders them to use some skill, changes their target or moves them around the room.

The difficulty range. As the party consist of 3 heroes, we modified the range of difficulty to be [0;3]; a difficulty was then the summed portions of health points lost during the encounter.

Difficulty matrix. To generate the initial sparse matrix, we generated 83572 scenarios, in each of which a random party was fighting a random group of monsters. In these scenarios the party was controlled by a very simple script doing random actions. This proved to be a better approximation of player behavior than a script making tactical decisions, which was too strong and produced estimations suitable for a skillful player rather then a beginner. The results of these scenarios were recorded and used as the initial matrix to be modified at runtime. We set the learning rate $\alpha = 0.75$ for UpdateMatrix method (Fig. 2).

Visualization of the initial difficulty matrix is provided in Fig. 5.

6 Experiment

To test our combat encounter generator we conducted a pilot experiment with human subjects playing the game.

6.1 Hypothesis

As there is no prior work to compare our results to, we designed the experiment to be exploratory only. We were interested to see if the algorithm can adapt initial estimation of combat difficulties to the player and improve the initial estimation of encounter difficulties. Our only hypothesis therefore was that the algorithm would be able to adapt initial difficulty matrix to the ability of players, i.e., it would provide better difficulty estimates.

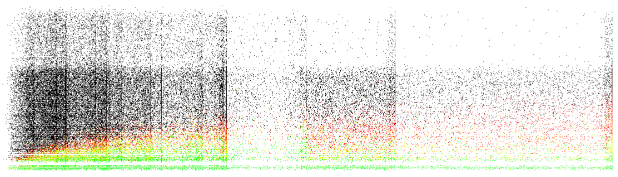

Figure 5: Visualization of initial values of difficulty matrix of the game. The x-axis is the party ordered according to their estimated strength, the y-axis represents enemies ordered according to their estimated strength. Colors depict different difficulty categories: green is [0;0.5), yellow is [0.5;1.5), red is [1.5;2.5), black is [2.5;3].

6.2 Design

Each participant was given the build of the game to play. The data about encounters was collected by the game itself and stored within online database. The game consist of three levels.

1. Tutorial level. A short and easy level with 4 combat encounters, which were fixed and created by us. The level existed for two reasons. First, it was designed to teach the player the basic mechanics of the game. Second, while the algorithm was not generating enemies in this level, it was still estimating the difficulty of these static encounters and was updating the matrix. Therefore, after the tutorial level the matrix could have matched the player skill more closely.

2. Static levels. Two levels with encounters fixed and created by us. Unlike the tutorial level, the difficulty matrix was not being modified during this phase.

3. Generated levels. Two levels with encounters being generated according to the matrix. The matrix got updated after every combat encounter.

Participants were split into two even groups A and B randomly. Group A was called *static-first* as their members played the game in the sequence 1-2-3. Group B was called *generated-first* as their members played the game in the sequence 1-3-2. The difference wrt. to difficulty estimation was that for static-first group, there was a gap in the matrix adaptation between levels 1 and 3. Here we were interested to see if there is going to be difference between groups as participants of the static-group should be more experienced when they enter the third level.

6.3 Data Collected

We collected both subjective categorical data from participants (both quantitative and qualitative) and objective numerical data concerning combat encounters, collected automatically by the game. Originally, we were also interested how would the players rate the game and collected

Table 1: Participants profile.

Group	Size	Male	Female
A	5	4	1
B	6	5	1

data on flow via the Flow Short Scale [16, 17] calibrated questionnaire [18], however these data was not conclusive.

Regarding the encounters, for each encounter in levels 1 and 3, the game recorded the enemies spawned, estimated difficulty by the initial version of the matrix and the current version of matrix and the result of the encounter, which is the final observed difficulty. We also generated a visualization of the difficulty matrix state after every matrix update for every participant.

6.4 Participants

The experiment was carried out during COVID-19 outbreak in May, 2020. Therefore, all experiments happened online in an uncontrolled environment, typically participants' homes. We had 22 participants out of which a total of 11 participants were not able to finish the level 3 (either because of a bug in a game, lack of time or not being able to understand all game mechanics) and thus we decided to discard their data from the analysis. In total, we analysed data from 11 participants (Table 1); all but two participants reported they play games more than 2 hours per week (majority more then 10), but these two reported that they played games regularly in the past. We consider both groups to consist of experienced game players only. Participants were split into groups randomly, they were of similar age (20-25).

6.5 Results and Discussions

Even though all the participants finished level 3, they played a different amount of combat encounters. This was the result of the difficulty we preset for some rooms, where some participants have their parties eliminated and were thus forced to replay the level.

Together, we assembled 260 combat encounter observations. The first 11 observations are irrelevant as they come from the first combat in level 1 where the matrix has been in the initial setting. In total, we analyse 249 combat encounters for which we can observe the differences between the adapted matrix (referred to as *ADAPTED* in graphs) and its initial version (referred to as *INIT* in graphs). We mainly analysed with estimation errors, i.e., difference between difficulty prediction of a matrix and the real difficulty observed within the game .

To visualize the data, we are using R version 4.0.5 together with the package ggstatsplot v0.8.0 and its *ggstatsplot* for group comparisons.

First, we were interested to see whether the algorithm was able to provide better difficulty estimations. For that, we measured errors of difficulty prediction as the difference between predicted difficulty and observed difficulty (negative value means the encounter was more difficulty then it should have been). Data for groups A and B respectively are summarized by Fig. 6 and Fig. 7.

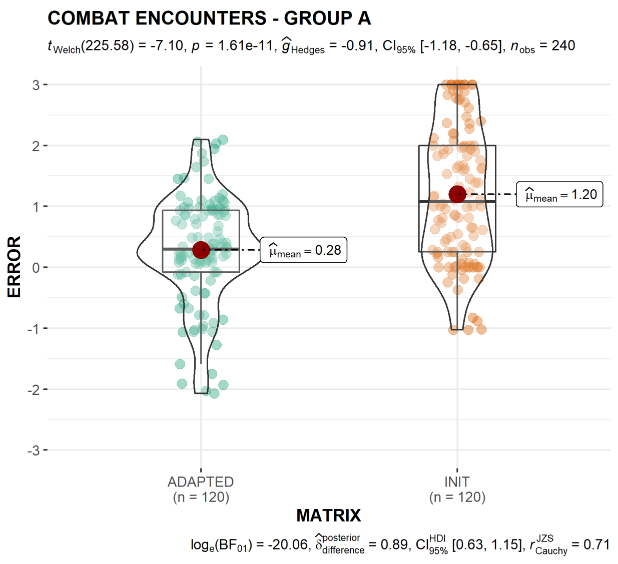

Figure 6: Difficulty prediction errors for group A; left part depicts value for adapted matrix and the right part for the initial state of the matrix.

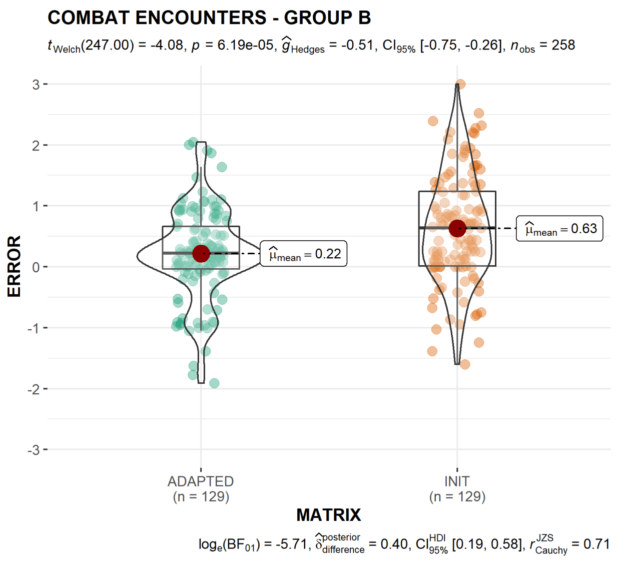

Figure 7: Difficulty prediction errors for group B; left part depicts value for adapted matrix and the right part for the initial state of the matrix.

For both groups, mean values between ADAPTED and INIT differ. Interestingly, mean of INIT matrix errors for group A is higher then in group B. This is probably the effect of "static-first" ordering in group A. Majority of data collected for group A happens only after participants played level 2. As the result, they are more skilled and INIT matrix is more off then for group B.

Then we looked how errors of ADAPTED matrices between groups looked like (Fig. 8).

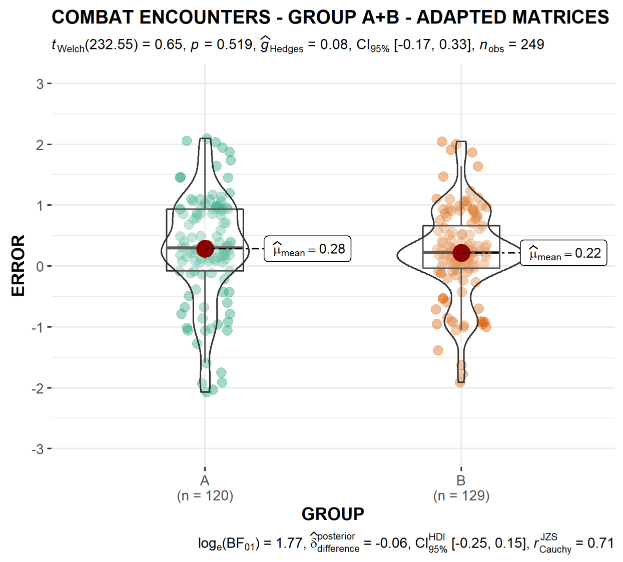

Figure 8: Difficulty prediction errors of ADAPTED matrices between groups.

Interestingly they are not much different, which is a bit surprising. Given the previous observation, that INIT matrix has a larger error for group A, we expected the same would be true for ADAPTED matrices between groups A and B. However, that's not the case. It seems that missing observation from level 2 in group A did not influence the ability of the algorithm to adapt to boosted players' abilities. The only difference, visually speaking, is the data distribution, which is more packed for group B.

Finally, we checked the error differences between ADAPTED and INIT matrices for both groups respectively (Fig. 9).

Interestingly, the analysis shows that the adapted matrix provided correct predictions more frequently (173/249 times in total 69.5%, 86 / 120 times for group A 71.7%, 85 / 129 times for group B 65.9%). Also means are different for both groups. We think this is the influence of statics-first vs. generated-first level again as INIT matrix had larger error for group A due to longer gameplay. Results per participants are summarized in Table 2.

We do not comment on statistical significance of findings as reported by graphs (e.g. Welch test and others) as not all samples are independent because multiple samples (estimation errors) are grouped according to players.

7 Conclusion

In this paper, we discussed a new procedural content generation problem: the Combat Encounter Generation Problem. We proposed an open-ended formalization of RPG games, in which CEGP arises. Moreover, if the game is not solving the CEGP during runtime, it must be solved by designers by hand during design time. We proposed an approach for solving CEGP consisting of two parts: a) an estimator of combat encounters difficulties, which needs to be able to adapt to players' ability online, and b) a generator that can propose next encounters given the estimation of their difficulties, which need to generate encounters fast. Our first implementation is making use of a) a custom reinforcement learning algorithm, which attempts to estimate difficulties of combat encounters despite the

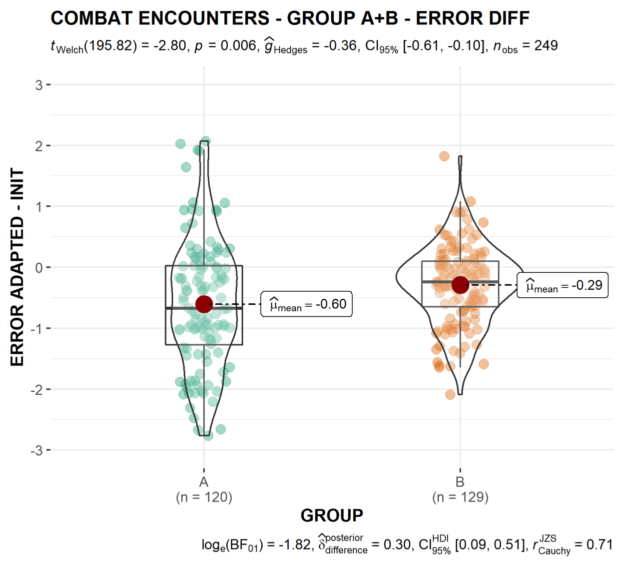

Figure 9: Difficulty prediction differences between ADAPTED and INIT matrices for both groups respectively. Positive number means the INIT matrix provided better prediction whereas negative number means ADAPTED matrix provided better prediction.

Table 2: Errors by participant. Columns: participant group and ID, number of encounters observed, mean and std. dev. of ADAPTED matrix errors, mean and std. dev. of INIT matrix errors, how many times was ADAPTED matrix prediction better then INIT matrix prediction.

Group, ID	#Enc	Err. ADAPTED	Err. INIT	Err. A<I
A-1	16	0.38 ± 0.89	0.82 ± 0.98	8
A-2	16	0.14 ± 0.77	0.56 ± 0.78	10
A-3	20	0.14 ± 0.22	0.61 ± 0.92	13
A-4	44	0.21 ± 0.96	1.46 ± 1.09	35
A-5	24	0.51 ± 0.81	1.89 ± 1.05	20
B-1	16	0.22 ± 0.61	0.89 ± 0.79	12
B-2	33	0.19 ± 0.78	0.72 ± 0.84	20
B-3	20	0.40 ± 0.76	1.23 ± 0.83	17
B-4	25	0.25 ± 0.89	0.72 ± 0.94	16
B-5	16	0.08 ± 0.53	0.13 ± 0.58	11
B-6	19	0.08 ± 0.01	-0.08 ± 0.59	11

limited number of observations, and b) utilize stochastic hill-climbing for finding a combat encounter of required difficulty. We presented and discussed preliminary results of the pilot study with human subjects. Even though the number of players is small, which is limiting the results, the number of combat encounters observed is moderate (249). The data shown that the algorithm was indeed able to adapt the difficulty estimation of encounters to players though the prediction was not always better wrt. initial setting of the matrix as computed from synthetic simulations using simple AI as human player surrogates. Given the fact that the number of encounters is huge and our approach is rather simplistic, we deem our results as promising.

7.1 Future Work

Encouraged by the results of our pilot experiment, we see many directions for future work.

First, it would be interesting to employ the algorithm in the context of a different game, ideally existing one, though this is usually very challenging technically. Second, if a (large enough) data set on how human players are playing a given game is available, it might be possible to model human-like play in order to experiment with the algorithm without the need of experiments with human subjects. Finally, proposed algorithm is a skeleton that can be configured at many places (e.g., creating different similarity functions or providing different abstractions for combat encounters), thus it would be interesting to observe how it behaves under different configurations.

Acknowledgement This work was supported by the Charles University grant SVV-260588.

References

[1] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural content generation in games*. Springer, 2016.

[2] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[3] M. Csikszentmihalyi, S. Abuhamdeh, and J. Nakamura, "Flow," in *Flow and the foundations of positive psychology*. Springer, 2014, pp. 227–238.

[4] J. H. Salisbury and P. Tomlinson, "Reconciling csikszentmihalyi's broader flow theory: with meaning and value in digital games," *Transactions of the Digital Games Research Association (ToDIGRA)*, vol. 2, no. 2, pp. 55–77, 2016.

[5] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. Harper Perennial Modern Classics, 2008.

[6] J. Chen, "Flow in games (and everything else)," *Communications of the ACM*, vol. 50, no. 4, pp. 31–34, 2007.

[7] R. Van Der Linden, R. Lopes, and R. Bidarra, "Procedural generation of dungeons," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 78–89, 2013.

[8] O. Nepožitek, "Procedural 2d map generation for computer games," *Master thesis, Univerzita Karlova, Matematicko-fyzikální fakulta*, 2018.

[9] J.-M. Vanhatupa, "Guidelines for personalizing the player experience in computer role-playing games," in *Proceedings of the 6th International Conference on Foundations of Digital Games*, 2011, pp. 46–52.

[10] J. Doran and I. Parberry, "Towards procedural quest generation: A structural analysis of rpg quests," *Dept. Comput. Sci. Eng., Univ. North Texas, Tech. Rep. LARC-2010*, vol. 2, 2010.

[11] G. Smith, M. Treanor, J. Whitehead, and M. Mateas, "Rhythm-based level generation for 2d platformers," in *Proceedings of the 4th international Conference on Foundations of Digital Games*, 2009, pp. 175–182.

[12] N. Shaker, A. Liapis, J. Togelius, R. Lopes, and R. Bidarra, "Constructive generation methods for dungeons and levels," in *Procedural Content Generation in Games*. Springer, 2016, pp. 31–55.

[13] S. Xue, M. Wu, J. Kolen, N. Aghdaie, and K. A. Zaman, "Dynamic difficulty adjustment for maximized engagement in digital games," in *Proceedings of the 26th International Conference on World Wide Web Companion*, 2017, pp. 465–471.

[14] O. Missura and T. Gärtner, "Player modeling for intelligent difficulty adjustment," in *International Conference on Discovery Science*. Springer, 2009, pp. 197–211.

[15] M. Csikszentmihalyi and I. S. Csikszentmihalyi, *Optimal experience: Psychological studies of flow in consciousness*. Cambridge university press, 1992.

[16] S. Engeser and F. Rheinberg, "Flow, performance and moderators of challenge-skill balance," *Motivation and Emotion*, vol. 32, no. 3, pp. 158–172, 2008.

[17] F. Rheinberg, R. Vollmeyer, and S. Engeser, "Die erfassung des flow-erlebens," 2003.

[18] M. Kozma, "Procedural generation of combat encounters in role playing video games," Master's thesis, Charles University, 2020.