Illuminating the Space of Enemies Through MAP-Elites

Breno M. F. Viana
Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
São Carlos, São Paulo, Brazil
bmfviana@gmail.com

Leonardo T. Pereira
Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
São Carlos, São Paulo, Brazil
leonardop@usp.br

Claudio F. M. Toledo
Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
São Carlos, São Paulo, Brazil
claudio@icmc.usp.br

***Abstract*—Action-Adventure games have several challenges to overcome, where the most common are enemies. The enemies' goal is to hinder the players' progression by taking life points, and the way they hinder this progress is distinct for different kinds of enemies. In this context, this paper introduces an extended version of an evolutionary approach for procedurally generating enemies that target the enemy's difficulty as the goal. Our approach advances the enemy generation research by incorporating a MAP-Elites population to generate diverse enemies without losing quality. The computational experiment showed the method converged most enemies in the MAP-Elites in less than a second for most cases. Besides, we experimented with players who played an Action-Adventure game prototype with enemies we generated. This experiment showed that the players enjoyed most levels they played, and we successfully created enemies perceived as easy, medium, or hard to face.**

***Index Terms*—enemy generation, procedural content generation, video game, evolutionary algorithm, map-elites**

I. Introduction

Enemies are crucial features of several game genres, such as Action-Adventure games. They offer challenges to the player since their goal is to hinder the player's progression and may require specific strategies to be defeated [1]. The enemies may provide different gameplay experiences, which is a relevant aspect for PCG methods [2]. However, PCG methods usually approach enemies by spreading them through a level map [3]–[5], instead of concerning more about how to create themselves [6], [7]. The generation of enemies in terms of attributes and visuals design appeared in some games such as Spore [8], No Man's Sky [9], and Creatures [10].

We present in this paper an extension of the enemy generation algorithm introduced by Pereira et al. [7]. They introduced a Parallel Evolutionary Algorithm (PAE) to evolve enemies represented by common features of Action-Adventure games [7]. Our approach advances the evolutionary strategy by applying MAP-Elites population to illuminate enemies' space, and we did not evolve the enemies through parallel evolution. Our MAP-Elites approach discretizes the enemy search space into a two-dimensional matrix mapped in terms of movement and weapon types.

The computational experiment showed that our approach converged most enemies in the MAP-Elites population with a generation process that took less than a second for all cases. We report results with players who enjoyed most of the gameplay, where enemies are successfully created based on easy, medium, or hard-to-face difficulty levels.

This paper is structured as follows. Section II presents the related works, and we highlighted our contributions. Section III describes the representation of our enemies and our evolutionary enemy generation approach. Section IV presents and discusses the results of our experiments. Finally, Section V presents the conclusions and future works.

II. Related Works

The research on procedural enemy generation is a recent research area, [7] and this section describes enemy generation methods from research works and industry games regarding any stage of their generative processes and gameplay progress. Table I summarizes our related work findings with the present paper contributions. In this table, placement refers to the works that perform enemy placement. The amount refers to those works that control the number of enemies to place, while status relates to papers that generated the enemies' attributes. Visuals refer to those developing visual components of enemies, and adaptive refers to works applying adaptive generated for such contents. Finally, MAP-Elites refers to those using such a technique to create enemies.

Balwin et al. [3] developed the Evolutionary Dungeon Designer (EDD) to assist game designers in their creative process. The EDD generates dungeon levels with enemies and rewards through an evolutionary approach. The levels match micro-patterns regarding levels' structure defined by users as input. The same authors later extended their approach to evolve meso-patterns that consider enemies and rewards besides the level structure [4]. However, both algorithms only place enemies in their levels.

Liapis [5] introduced an evolutionary approach that works on two steps. First, the method evolves level sketches composed of walls and different rooms types to place them strategically. Rooms are classified discretely with pre-defined numbers of rewards and enemies, and they may connect with other rooms in one, two, three, or eight directions. After dispersing rooms, the second stage evolves each room in a

TABLE I
ENEMY GENERATION LITERATURE SUMMARIZING AND COMPARISON WITH THIS WORK. 'P' DEFINES THE PARTIALLY GENERATED ENEMY FEATURES.

Work	Placement	Amount	Status	Visuals	Adaptive	MAP-Elites
Baldwin et al. [3]	✓	✓	-	-	-	-
Baldwin et al. [4]	✓	✓	-	-	-	-
Liapis [5]	✓	-	-	-	-	-
Khalifa et al. [6]	✓	✓	P	P	-	✓
Pereira et al. [7]	✓	✓	✓	P	-	-
Spore [8]	-	-	P	P	-	-
No Man's Sky [9]	-	-	P	P	-	-
Creatures [10]	-	-	✓	P	✓	-
Diablo 3 [11]	-	-	P	-	-	-
Middle Earth: Shadow of Mordor [12]	-	-	P	-	-	-
Left 4 Dead 2 [13]	✓	✓	-	-	✓	-
State of Decay 2 [14]	✓	✓	-	-	✓	-
This work	✓	-	✓	P	-	✓

cavern-fashion way by placing walls and enemies strategically around rewards and protecting them from the player. Again, this work only puts enemies at their levels.

So far, the papers described only the placement of enemies instead of generating the enemies that players would face in their games. Khalifa et al. [6] were pioneers in such an area by introducing an evolutionary approach to evolve Bullet Hell games' levels. These games consist of bullets (enemies) with different damage values, speed, and movement patterns. To create the levels, they evolved Talakat scripts, a language proposed to describe their levels, through Constrained MAP-Elites (CME). These scripts have sections for spawners and the boss. The former section defines spawn points to spawn bullets or create new spawners, while the latter describes the boss's health, position, and behavior. The spawners have sets of parameters to determine the bullet it spawns, i.e., its speed, angle, size, and the angle rotation and speed of spawners.

Therefore, they generate enemies that players face besides placing them in their levels. In a single execution, the authors developed a variety of levels without losing quality by using a Quality Diversity approach. This class of algorithms is exciting for PCG purposes [15]. More specifically, Khalifa et al. [6] applied an extension of the MAP-Elites, an Illumination Algorithm that returns a set of the best-found solutions discretized in a map regarding their features [16]. The original MAP-Elites is the one we applied in our work.

Pereira et al. [7] presented a Parallel Evolutionary Algorithm (PEA) that generates enemies for an Action-Adventure game. The authors extracted the most common variables from enemies in different Action-Adventure games to build the enemy's genotype. Their PEA evolves enemies matching their difficulty degrees with the difficulty goal given as input. Their method inspires our approach, but the main differences are the MAP-Elites and the new difficulty function introduced here.

Regarding the industry games, the creation of Non-Playable Characters (NPCs) is present in *Spore* [8] and *No Man's Sky* [9]. These NPCs may be confronted by players, just like enemies. The creatures of these games are created by randomly assigning different body parts. Their algorithms have some constraints for body parts, and they do not put together parts that cannot match another already selected. This strategy ensures the feasibility of procedural animations of the NPCs.

An older game series used an extended version of this concept, *Creatures* [10] generated NPCs by evolving them physically and making them learn. These NPCs could learn about the environment, and the player's actions via a neural network that receives simulated senses using semi-symbolic approximation techniques as input [17].

Regarding the generation of actual enemies, i.e., NPCs that actively look for fighting players, *Diablo 3* [11] and *Shadow of Mordor* [12] generated their enemies by changing some predefined characteristics. This behavior change approach allowed these games to make more diverse and unique challenges. In *Left 4 Dead 2* [13] and *State of Decay 2* [14], the enemies can adapt to the players. However, instead of changing their features, like in the previous two games, *Left 4 Dead 2* [13] and *State of Decay 2* [14] decide how many and where to place enemies. They make such decisions accordingly to the players' performance. If the player is doing well, new enemies are spawned, else the games spawn fewer enemies[1] [2].

III. METHODOLOGY

This section describes our enemy representation and how we evolve them through our MAP-Elites approach.

A. Representation

As mentioned, we propose an evolutionary approach that advances from the method introduced in [7], where our enemy genotypes come from the one presented in the previous work. Such representation extracts common attributes of enemies in Action-Adventure games to build the enemy's genotype. These attributes are the following: health, damage, attack speed, movement speed, active time, rest time, movement type,

[1]The AI Systems of Left 4 Dead (https://steamcdn-a.akamaihd.net/apps/valve/2009/ai_systems_of_l4d_mike_booth.pdf).

[2]Procedurally generating enemies, places, and loot in State of Decay 2 (https://www.gamedeveloper.com/design/procedurally-generating-enemies-places-and-loot-in-i-state-of-decay-2-i-).

TABLE II
LIST OF ATTRIBUTES OF THE ENEMY'S GENOTYPE. THE LINE BETWEEN ATTRIBUTES REPRESENTS THE CROSSOVER POINT. ADAPTED FROM [7].

Attribute	Type	Range	Details (the attribute defines...)
Health	Integer	1-5	How many hits an enemy endures.
Damage	Integer	1-4	How many life points an enemy takes from the player.
Attack Speed	Float	0.75-4.0	How frequent an enemy shoots a projectile (1/Attack Speed).
Movement Type	Nominal	-	How the enemy moves during gameplay.
Movement Speed	Float	0.8-2.8	How faster the enemy moves.
Active Time	Float	1.5-10.0	The time in seconds that the enemy moves before resting.
Rest Time	Float	0.3-1.5	The time in seconds that the enemy rests before moving.
Weapon Type	Nominal	-	The weapon gameplay properties.
Projectile Speed	Float	1.0-4.0	How faster the projectile moves towards the player.

weapon type, and projectile speed. Table II describes these attributes and shows their respective ranges of values. Our only change in the numerical values was the max movement speed value; we decreased it slightly because the max value was too fast.

Again in Table II, movement type and weapon type are nominal attributes representing more complex behaviors and objects that enemies may have. Following, we list the types of movements:

- **None** the enemy stays still.
- **Random** the enemy's movement is defined by a random direction 2D vector.
- **Random 1D** the enemy's movement is determined by a random direction 1D vector (i.e., horizontal or vertical).
- **Flee** the enemy's movement is calculated by the opposite of the player's direction vector.
- **Flee 1D** the enemy's movement is calculated by the opposite of a single axe of the player's direction vector (i.e., horizontal or vertical).
- **Follow** the enemy's movement is determined by the direction vector that points towards the player.
- **Follow 1D** the enemy's movement is defined by a single axe of the direction vector that points towards the player (i.e., horizontal or vertical).

All these movements occur during the active time. Regarding weapon types, we list their types and describe how they work:

- **Barehand (None)** deals damage on contact.
- **Sword** deals damage on contact with a higher reach regarding the barehand.
- **Bow** shoots bullets towards the player, and they deal damage when hit the player.
- **Bomb-Thrower** shoots a bomb towards the player; they explode in 2 seconds and deal damage in a limited area.
- **Shield** protects the enemy from frontal attacks.
- **Cure Spell** cures one health point of all enemies in a circular area.

We add the cure spell to generate healer enemies, and our melee enemies use the following weapons: barehand, sword, and shield. Furthermore, our ranged enemies use a bow and bomb-thrower. We discarded the weights of the movement and weapon types and dealt with these attributes in dedicated equations to calculate the enemies' difficulty.

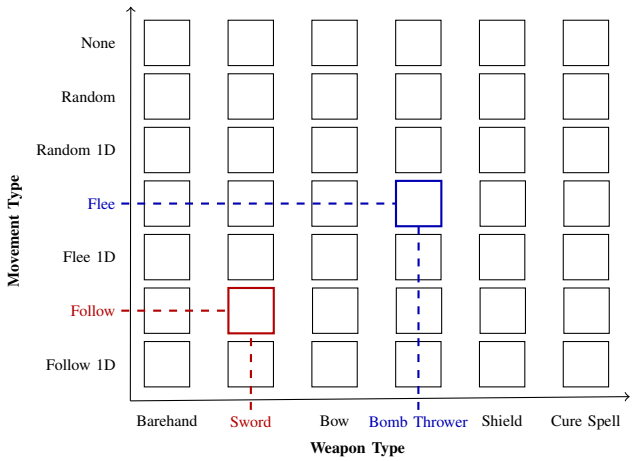


Fig. 1. The map of MAP-Elites population. The red cell represents a melee enemy that follows the player to hit with a sword. The blue cell represents a ranged enemy that flees from the player while throwing bombs towards them.

B. Generation Process

The input for the generation process is only the goal difficulty of enemies. The fitness function measures the distance between aimed difficulty and the difficulty encoded in the enemy stated as an individual (representation of solution) of the evolutionary algorithm. Therefore, our approach minimizes such fitness. We designed a MAP-Elites approach to preserve diversity while optimizing the quality of enemies. We discretized our map regarding movement and weapon types; thus, we have nominal values as feature descriptors (dimensions). Since we do not need to calculate numerical equations, our mapping functions are straightforward:

$$D_{\text{movement}} = e_{movement\_type} \tag{1}$$

$$D_{weapon} = e_{weapon\_type} \tag{2}$$

where, e is the enemy. Fig. 1 presents our approach's map. The cell highlighted in red represents an enemy that follows the player to hit with a sword, while in blue represents an enemy that flees from the player while throwing bombs towards them.

The proposed MAP-Elites approach maps 42 enemies in terms of the defined dimensions. When the population receives a new individual, we calculate its feature descriptors to place it in the correct map entry. If a new enemy hits a filled cell,

we apply elitism, i.e., the best enemy fills the cell, discarding the other one.

The evolutionary process starts by generating the initial population thought filling the attributes of n enemies randomly. Since the initialized enemies may hit the same map entry, the initial population generation may take a while. Besides, since we discretized the population in a map, its size does not change. Thus, the best individual is always kept.

Next, we evolve the population using the generation-limit stopping criterion. The authors in [7] replace all the population in each generation by the intermediate population generated. We also have an intermediate population; however, we try to add its individuals in the MAP-Elites population. The reproduction operators create new individuals from two parents, chosen using tournament selection with two competitors.

We first perform a crossover with a 100% rate to generate two new individuals when reproducing enemies. Our crossover is a combination of a fixed-single-point crossover and a BLX-α crossover [18]. We first cross the parents in the fixed point as shown in Table II. We designed the crossover to fill our map faster once new individuals may hit new cells. For instance, if the elites Bow-Flee and Sword-Follow cross, we generate two individuals mapped in Bow-Follow and Sword-Flee cells. After this, we perform the BLX-α crossover for each numerical attribute [18].

After the crossover, we have a chance to mutate both resulting enemies and, when a mutation happens, we apply a multi-gene mutation [19]. To do so, we calculate the chance of mutating each gene. This mutation means that our mutation operator can change all the enemy's attributes. We set a new random value for each gene that mutates, respecting the limited range and the list of nominal values of the attributes.

Our difficulty function has four factors: health, movement, strength, and gameplay. The enemies' life points determine how many hits they endure.

$$d_{health} = 2 \times e_{health} \tag{3}$$

Regarding the movement factor, we consider three attributes of our individuals: movement speed, active time, and rest time.

$$d_{movement} = e_{mv\_spd} + e_{act\_tm}/3 + 1/e_{rst\_tm} \tag{4}$$

where, mv_spd is the movement speed, act_tm is the active time, and rst_tm is the rest time. The faster the enemy, the more difficult it will be for the player to defend the enemy's tackles. The more time active moving, the more difficult the enemy is. We weighed this term with $1/3$ to balance its influence in this equation. Finally, the more time resting, the more easily it will be to defeat; thus, we calculate its inverse.

The strength factor is more complex than the previous difficulty factors; it depends on the types of enemies and, therefore, we multiply three different equations.

$$d_{strength} = d_{s_1} \times d_{s_2} \times d_{s_3} \tag{5}$$

For melee enemies, we multiply damage by movement speed.

$$d_{s_1} = \begin{cases} e_{dmg} \times e_{mv\_spd}, & \text{isMelee}(e) \\ 1, & \text{otherwise} \end{cases} \tag{6}$$

where, dmg is the damage. We multiply attack speed by projectile speed for ranged enemies and weigh the result by three.

$$d_{s_2} = \begin{cases} 3 \times (e_{atk\_spd} \times e_{prjct\_spd}), & \text{isRanged}(e) \\ 1, & \text{otherwise} \end{cases} \tag{7}$$

where, atk_spd is the attack speed, and $prjct\_spd$ is the projectile speed. We consider only the attack speed for healer enemies since they always heal a single life point of all enemies in their heal area range.

$$d_{s_3} = \begin{cases} 2 \times e_{atk\_spd}, & \text{isHealer}(e) \\ 1, & \text{otherwise} \end{cases} \tag{8}$$

We also calculate the gameplay factor considering the enemies' weapons and our game prototype, where we experimented with the generated enemies. Here we also increase the difficulty of incoherent enemies; thus, discarding them based on a threshold defined by the user.

$$d_{gameplay} = d_{g_1} \times d_{g_2} \times d_{g_3} \times d_{g_4} \times d_{g_5} \tag{9}$$

Melee enemies that follow the player are more dangerous, thus, more challenging to defeat. Moreover, melee enemies that flee from the player or stay still are less risky and easier to defeat. Therefore, we weigh the difficulty as follows.

$$d_{g_1} = \begin{cases} 1.25, & \text{isMelee}(e) \text{ and isFollow}(e) \\ 0.5, & \text{isMelee}(e) \text{ and} \\ & (\text{isAnyFlee}(e) \text{ or hasNoMove}(e)) \\ 1, & \text{otherwise} \end{cases} \tag{10}$$

Since ranged enemies perform distance attacks, those that flee from the player present more risk. When ranged enemies stay still, players can defeat them easier since they are static targets. Ranged enemies that follow the player may have the projectile speed faster than their movement speed, or else they will not behave as rangers since their projectiles will be slower than they own. This behavior did not occur in enemies with the movement Follow1D. Thus, we weigh the difficulty as follows.

$$d_{g_2} = \begin{cases} 1.25, & \text{isRanged}(e) \text{ and isFlee}(e) \\ 1.15, & \text{isRanged}(e) \text{ and isFlee1D}(e) \\ 0.5, & \text{isRanged}(e) \text{ and hasNoMove}(e) \\ 1, & \text{otherwise} \end{cases} \tag{11}$$

$$d_{g_3} = \begin{cases} 0.5/(2 \times e_{mv\_spd}), & \text{isRanged}(e) \text{ and} \\ & \text{isFollow}(e) \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

Healers must protect themselves while keeping healing other enemies. Thus, they should not follow players but avoid them. Besides, healers that move faster are more difficult to defeat;

thus, we also weighed this factor by their movement speed. Therefore, we weigh the enemy's difficulty as follows.

$$d_{g_4} = \begin{cases} 1, & \text{isHealer}(e) \text{ and} \\ & (\text{isAnyRandom}(e) \text{ or} \\ & \text{isAnyFlee}(e)) \\ 0.5, & \text{otherwise} \end{cases} \quad (13)$$

$$d_{g_5} = \begin{cases} 1.15 \times e_{mv\_spd}, & \text{isHealer}(e) \\ 1, & \text{otherwise} \end{cases} \quad (14)$$

All the numeric weights in the equations were chosen empirically through gameplay experiments. Finally, we defined the final difficulty equation as follows.

$$d = d_{gameplay} \times (d_{health} + d_{movement} + d_{strength}) \quad (15)$$

IV. Results

In this section, we present how our approach performed computationally and the feedback of human players after playing with our enemies.

A. Performance Results

We defined the parameters of our approach empirically after comparing some range of values. The results comparing different sets of evolutionary parameters are available in a Google Sheets spreadsheet[3]. After this comparison, we set the following parameters: 500 generations as stop-criterion, 35 individuals for initial population, 100 individuals for intermediate population, 20% for mutation rate, 30% for gene mutation rate, 2-size for tournament selection. Next, we collected data from 100 executions of our method for three different difficulty goals to evaluate its performance. In such an experiment, each player could play three different levels with easy, medium, and hard enemies, respectively. We opt for this to provide short gameplay sections. Table III shows the average and standard deviation of the fitness for each Elite (entry) of our MAP-Elites population, where most solutions converged to zero, the best value when using a distance measure for fitness.

We expect values close to zero for lower difficulty values because these are easier to reach. Still, in Table III, Bow and Bomb Thrower enemies with movements of Flee and Follow presented higher values. We expect such values for ranged enemies with Flee movement, once this movement makes enemies more dangerous. Elite gets closer to zero in the remaining tables, meaning the respective Elite is reaching the aimed difficulty. On the other hand, regarding the ranged enemies with the movement Follow, their distance is a little high due to a lack of balance between projectile and movement speed attributes. This result is similar in the remaining cases.

In tables IIId and IIIe, we observe that new Elites in both tables have significantly higher distance values than the others. These Elites are the enemies with the Barehand, Sword, and Shield as weapons and None, Flee, and Flee1D as movements.

[3]Link to the spreadsheet https://docs.google.com/spreadsheets/d/19SMHZYT_pfniZDuNS0BOYMt1kWyBOlvqFq_Y7vBNrS4.

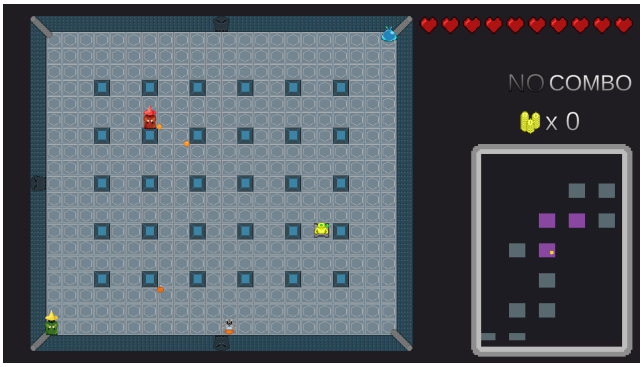
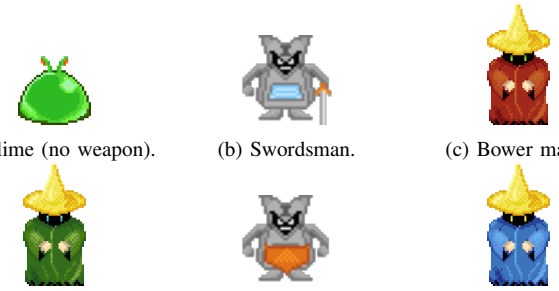


Fig. 2. Game prototype screenshot.



(a) Slime (no weapon). (b) Swordsman. (c) Bower mage.

(d) Bomber mage. (e) Shieldsman. (f) Healer (Cure Spell).

Fig. 3. List of enemies of our game prototype. Slimes have no weapon. Swordsmans use swords. Bower mages shoot arrows. Bomber mages throw bombs. Shieldsmans hold shields. Healers use cure spell to heal other enemies.

We expected such results because these values are the ones we set with lesser values for their weights in the gameplay factor of our difficulty function. Besides, we also observe a difficulty limit in these enemies for both tables. These values vary between 15 and 16 since the distance between them and the difficulties 17 and 19 are, respectively, 2 and 4 approximately.

Regarding the execution time, Table IV presents the average, minimum, maximum, and standard deviation values for duration time (seconds) for the 100 executions. The experiments regarding performance were carried out in a PC with the following setup: Intel Core i7-7700HQ 2.80GHz Processor (8 cores), 16 GB DDR4 RAM, 236GB SSD memory, NVIDIA GeForce GTX 1050 Ti 4GB graphics card. The results show that different values of difficulty goals do not impact the execution time of our approach. Therefore, our approach can generate enemies of any difficulty without performance loss.

The Parallel Evolutionary Algorithm presented by Pereira et al. [7] can generate a huge number of enemies in a single execution. Although our approach generates significantly fewer enemies, it ensures diversity. Our method was slightly faster regarding average execution time, since their loweest average time was 0.168 seconds.

B. Gameplay Feedback

We experimented with anonymous volunteers who evaluated the quality of enemies through an in-game, optional, questionnaire. We shared the game, hosted in a server, via social media and mailing lists. Sets of enemies were generated with

TABLE III
RESULTS OF FITNESS OBTAINED AFTER 100 EXECUTIONS OF OUR APPROACH. EACH FITNESS VALUE CORRESPONDS TO THE DISTANCE BETWEEN THE INPUT DIFFICULTY AND THE DIFFICULTY OF THE FOUND ENEMY – THE CLOSER TO ZERO, THE BETTER.

(a)VERY EASY DIFFICULTY = 11.

	Barehand	Sword	Bow	Bomb Thrower	Shield	Cure Spell
None	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.00	0.01 ± 0.01
Random	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.00	0.01 ± 0.01
Random1D	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.00	0.01 ± 0.01
Flee	0.00 ± 0.00	0.00 ± 0.00	0.16 ± 0.23	0.21 ± 0.38	0.00 ± 0.00	0.03 ± 0.03
Flee1D	0.00 ± 0.00	0.00 ± 0.00	0.06 ± 0.10	0.07 ± 0.12	0.00 ± 0.00	0.03 ± 0.04
Follow	0.01 ± 0.01	0.01 ± 0.01	0.43 ± 0.58	0.43 ± 0.53	0.01 ± 0.02	0.02 ± 0.02
Follow1D	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.00	0.01 ± 0.01

(b)EASY DIFFICULTY = 13.

	Barehand	Sword	Bow	Bomb Thrower	Shield	Cure Spell
None	0.00 ± 0.01	0.00 ± 0.00	0.01 ± 0.02	0.01 ± 0.02	0.00 ± 0.00	0.01 ± 0.01
Random	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	0.01 ± 0.03	0.00 ± 0.00	0.01 ± 0.01
Random1D	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.00	0.01 ± 0.01
Flee	0.00 ± 0.02	0.00 ± 0.01	0.12 ± 0.22	0.09 ± 0.20	0.00 ± 0.01	0.03 ± 0.03
Flee1D	0.00 ± 0.01	0.00 ± 0.01	0.04 ± 0.05	0.04 ± 0.05	0.00 ± 0.01	0.03 ± 0.03
Follow	0.01 ± 0.01	0.01 ± 0.01	1.07 ± 1.12	1.06 ± 0.97	0.01 ± 0.01	0.02 ± 0.02
Follow1D	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.00	0.01 ± 0.01

(c)MEDIUM DIFFICULTY = 15.

	Barehand	Sword	Bow	Bomb Thrower	Shield	Cure Spell
None	0.07 ± 0.14	0.06 ± 0.14	0.02 ± 0.03	0.02 ± 0.03	0.08 ± 0.18	0.02 ± 0.02
Random	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.00	0.01 ± 0.01
Random1D	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.00	0.01 ± 0.02
Flee	0.07 ± 0.14	0.08 ± 0.23	0.03 ± 0.04	0.03 ± 0.04	0.08 ± 0.22	0.03 ± 0.03
Flee1D	0.05 ± 0.11	0.06 ± 0.13	0.02 ± 0.02	0.02 ± 0.03	0.07 ± 0.14	0.02 ± 0.03
Follow	0.01 ± 0.01	0.01 ± 0.01	1.70 ± 1.47	2.15 ± 1.82	0.01 ± 0.01	0.02 ± 0.02
Follow1D	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01

(d)HARD DIFFICULTY = 17.

	Barehand	Sword	Bow	Bomb Thrower	Shield	Cure Spell
None	1.91 ± 0.25	1.90 ± 0.20	0.03 ± 0.03	0.03 ± 0.03	1.91 ± 0.21	0.02 ± 0.02
Random	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01
Random1D	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01
Flee	1.96 ± 0.27	1.94 ± 0.27	0.02 ± 0.03	0.02 ± 0.01	1.96 ± 0.27	0.02 ± 0.02
Flee1D	1.98 ± 0.27	1.94 ± 0.24	0.01 ± 0.01	0.01 ± 0.01	1.95 ± 0.26	0.02 ± 0.02
Follow	0.01 ± 0.01	0.01 ± 0.01	2.32 ± 2.07	2.29 ± 2.16	0.01 ± 0.01	0.01 ± 0.01
Follow1D	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01

(e)VERY HARD DIFFICULTY = 19.

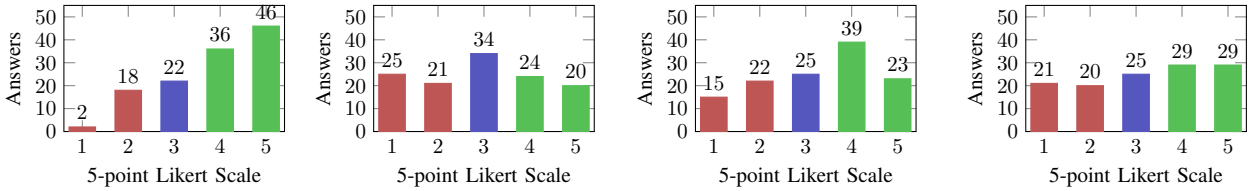
	Barehand	Sword	Bow	Bomb Thrower	Shield	Cure Spell
None	3.91 ± 0.23	3.93 ± 0.24	0.04 ± 0.04	0.03 ± 0.04	3.93 ± 0.27	0.02 ± 0.02
Random	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01
Random1D	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01
Flee	3.95 ± 0.27	3.99 ± 0.32	0.02 ± 0.02	0.01 ± 0.02	3.94 ± 0.29	0.02 ± 0.02
Flee1D	3.90 ± 0.18	3.89 ± 0.17	0.01 ± 0.01	0.01 ± 0.01	3.91 ± 0.18	0.02 ± 0.02
Follow	0.01 ± 0.01	0.01 ± 0.01	3.14 ± 2.39	3.35 ± 2.63	0.01 ± 0.01	0.01 ± 0.01
Follow1D	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.01

TABLE IV
RESULTS OF TIME IN SECONDS OBTAINED AFTER 100 EXECUTIONS OF OUR APPROACH.

Difficulty	Average	Minimum	Maximum	Standard Deviation
11	0.1608	0.1510	0.2345	0.0126
13	0.1609	0.1521	0.2255	0.0115
15	0.1597	0.1508	0.2474	0.0145
17	0.1621	0.1509	0.2000	0.0106
19	0.1581	0.1512	0.1970	0.0072

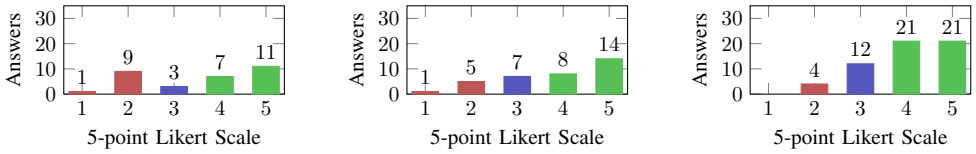
different difficulty levels and included in our game prototype, which is an adapted version of the game applied in [7]. Therefore, the players must solve locked-door puzzles, defeat enemies and reach the level goal. The players can use blocks within some rooms to protect themselves from enemies, which are positioned in rooms like in [7]. Fig. 2 shows a screenshot of such game prototype. In our game, players should defeat the six enemies listed in Fig. 3.

A total of 96 players faced our enemies in the game, and 75 answered all the questions. They played 124 levels with enemies randomly placed in rooms. Regarding difficulty, they played 31 levels with easy enemies, 35 with medium enemies,



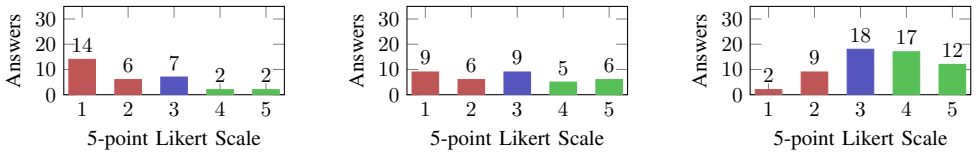
(a) Q1 (Level was fun). (b) Q2 (Enemies were difficult). (c) Q3 (Balance was right). (d) Q4 (Enemies were human-made).

Fig. 4. Bar charts of answers of the 75 players after playing 124 levels.



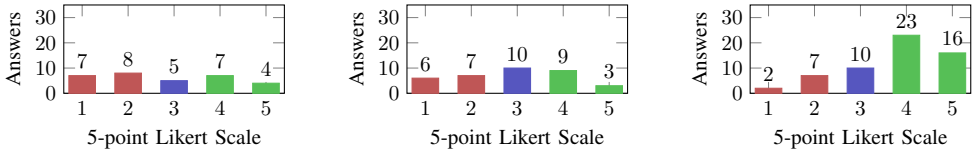
(a) Easy. (b) Medium. (c) Hard.

Fig. 5. Bar charts of answers for question Q1 (“The level was fun to play”).



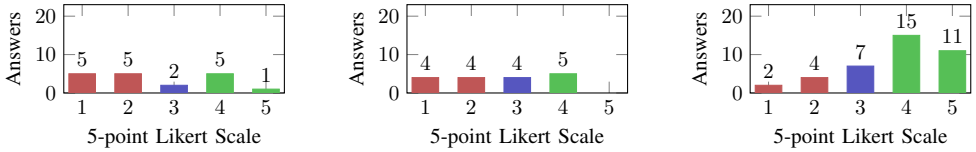
(a) Easy. (b) Medium. (c) Hard.

Fig. 6. Bar charts of answers for question Q2 (“The enemies of this level were difficult to defeat”).



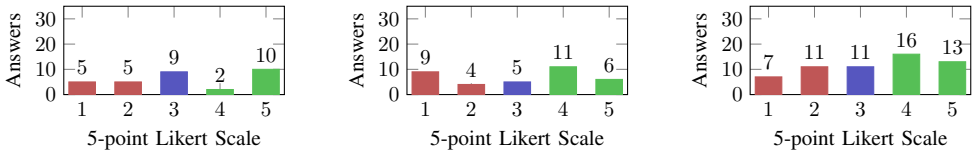
(a) Easy. (b) Medium. (c) Hard.

Fig. 7. Bar charts of answers for question Q3 (“The challenge was just right (balance)”).



(a) Easy. (b) Medium. (c) Hard.

Fig. 8. Bar charts of answers for question Q3 (“The challenge was just right”) of 43 players for 74 levels. These players answered they enjoy battles.



(a) Easy. (b) Medium. (c) Hard.

Fig. 9. Bar charts of answers for question Q4 (“The enemies I faced were created by humans”).

and 58 with hard enemies. After playing the levels, the players answered how much agree or disagree, on a five-point Likert scale, with the following statements:

Q1 The level was fun to play;
Q2 The enemies of this level were difficult to defeat;
Q3 The challenge was just right (balance);
Q4 The enemies I faced were created by humans.

In our experiments, we had 51 out of 75 players declaring to

have considerable experience with games in general and 53 out of 75 with Action-Adventure games. Fig. 4 shows the overall results of players' feedback for each question of our questionnaire. The players enjoyed most the levels played (Fig. 4a), and Fig. 4b shows the players had no difficulty defeating the enemies in 80 levels. About the challenge suitability (Fig. 4c), they answered that the challenge was just right in half of the levels (62 out of 124) and neutral for 25 of the levels. Finally, in Fig. 4d, we observe that the players sustained that humans created our enemies in 58 of the levels.

Fig. 5 shows results of question Q1 for each difficulty range. The players enjoyed most levels regardless of the difficulty of the enemies. However, they preferred medium and hard levels since the negative answers decreased while positive ones increased.

Fig. 6 shows results of question Q2 for each difficulty. In Fig. 6a, the players felt that the enemies in 20 out of 31 easy levels were not challenging to defeat, and there were only four levels with enemies perceived as harder to overcome. For the medium levels, the players reported easy enemies to defeat in 15, hard in 11, and neutral in 9 levels (Fig. 6b). Finally, from 58 hard levels, Fig. 6c shows enemies hard to defeat in 29 levels based on players' feedback. They perceived enemies as easy to defeat in only 10 levels and neutral in 18 levels. Thus, the results about the difficulty range of our enemies corroborate our settings for difficulty values.

Fig. 7 shows results of question Q3 for each difficulty. Figures 7a and 7b illustrate that approximately half of the players agree and half disagree that easy and medium levels challenges were just right. In contrast, Fig. 7c shows that the players perceived the challenge as adequate in most hard levels (39 out of 58).

Besides the questionnaire to evaluate levels, we also asked the players if they enjoyed battling during their gameplay. Fig. 8 shows the feedback of the players who confirmed such question. The answers are analogous to Fig. 7, and these players also preferred the levels with harder enemies, as shown in Fig. 8c. Considering such results, we believe our levels probably could present better challenges if we mix up some enemies with different difficulty degrees.

Fig. 9 shows results of question Q4 for each difficulty. The results indicate that players notice enemies as human-made in most levels regarding the difficulty degree of the enemies. Besides, this perception is more visible for medium and hard enemies than easy ones. These results mean that if the enemy is harder to overcome, the players perceive them as more carefully designed.

V. Conclusion

In this paper, we introduced an illumination approach that extended the enemy generation presented by Pereira et al. [7]. We advanced the previous method by illuminating the enemies through the MAP-Elites approach. Our experiments showed that the players had fun while playing the levels with the enemies generated regardless of the difficulty. The results corroborated the difficulty values we set to create easy, medium, and hard enemies. Furthermore, our enemies were carefully designed in enough way to be perceived as human-made enemies. As future works, we intend to extend our work with the Constrained MAP-Elites approach [15]. With this approach, we can generate multiple enemies and avoid incoherent enemies.

Acknowledgments

We acknowledge the financial support of the National Council for Scientific and Technological Development (CNPq).

References

[1] B. M. F. Viana and S. R. dos Santos, "Procedural dungeon generation: A survey," *Journal on Interactive Systems*, vol. 12, no. 1, pp. 83–101, 2021.

[2] J. Togelius, N. Shaker, and M. J. Nelson, "Introduction," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2016, pp. 1–15.

[3] A. Baldwin, S. Dahlskog, J. M. Font, and J. Holmberg, "Mixed-initiative procedural generation of dungeons using game design patterns," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017, pp. 25–32.

[4] ——, "Towards pattern-based mixed-initiative dungeon generation," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*. ACM, 2017, p. 74.

[5] A. Liapis, "Multi-segment evolution of dungeon game levels," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 203–210.

[6] A. Khalifa, S. Lee, A. Nealen, and J. Togelius, "Talakat: Bullet hell generation through constrained map-elites," in *Proceedings of The Genetic and Evolutionary Computation Conference*, 2018, pp. 1047–1054.

[7] L. T. Pereira, B. M. F. Viana, and C. F. M. Toledo, "Procedural enemy generation through parallel evolutionary algorithm," in *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2021, pp. 126–135.

[8] Maxis™, "Spore," 2008, accessed in: 2021-02-11. [Online]. Available: http://www.spore.com/.

[9] Hello Games, "No man's sky," 2018, accessed in: 2020-09-11. [Online]. Available: https://www.nomanssky.com.

[10] Creature Labs, "Creatures," 1996, accessed in: 2021-02-11. [Online]. Available: https://creatures.fandom.com/wiki/Creatures.

[11] Blizzard, "Diablo iii," 2012, accessed in: 2021-02-11. [Online]. Available: https://us.diablo3.com/en/.

[12] Monolith Productions, "Middle-earth: Shadow of mordor," 2014, accessed in: 2021-02-11. [Online]. Available: https://store.steampowered.com/app/241930/Middleearth_Shadow_of_Mordor/.

[13] Valve, "Left 4 dead 2," 2009, accessed in: 2021-02-11. [Online]. Available: https://store.steampowered.com/app/550/Left_4_Dead_2/.

[14] Undead Labs, "State of decay 2," 2018, accessed in: 2021-02-11. [Online]. Available: https://state-of-decay-2.fandom.com/wiki/State_of_Decay_2_Wiki.

[15] D. Gravina, A. Khalifa, A. Liapis, J. Togelius, and G. N. Yannakakis, "Procedural content generation through quality diversity," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[16] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[17] S. Grand and D. Cliff, "Creatures: Entertainment software agents with artificial life," *Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 1, pp. 39–57, 1998.

[18] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," in *Foundations of genetic algorithms*. Elsevier, 1993, vol. 2, pp. 187–202.

[19] R. Kanagal-Shamanna, B. P. Portier, R. R. Singh, M. J. Routbort, K. D. Aldape, B. A. Handal, H. Rahimi, N. G. Reddy, B. A. Barkoh, B. M. Mishra *et al.*, "Next-generation sequencing-based multi-gene mutation profiling of solid tumors using fine needle aspiration samples: promises and challenges for routine clinical diagnostics," *Modern pathology*, vol. 27, no. 2, pp. 314–327, 2014.