UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Orchestrating and Adapting of Dungeon Levels, Locked-door Missions, and Enemies

Breno Maurício de Freitas Viana

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

Breno Maurício de Freitas Viana

Orchestrating and Adapting of Dungeon Levels, Locked-door Missions, and Enemies

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Claudio Fabiano Motta Toledo

USP – São Carlos
May 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

V614o

Viana, Breno Maurício de Freitas
Orchestrating and Adapting of Dungeon Levels, Locked-door Missions, and Enemies / Breno Maurício de Freitas Viana; orientador Claudio Fabiano Motta Toledo. -- São Carlos, 2022.
101 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2022.

1. Orquestração de Conteúdo. 2. Geração de Níveis. 3. Geração de Inimigos. 4. Geração Adaptativa. 5. MAP-Elites. I. Toledo, Claudio Fabiano Motta, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Breno Maurício de Freitas Viana

Orquestrando e Adaptando Níveis de Calabouço, Missões de Portas Fechadas e Inimigos

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Claudio Fabiano Motta Toledo

USP – São Carlos
Maio de 2022

I dedicate this work to my family and friends, who always give me the support I need.

ACKNOWLEDGEMENTS

First, I thank God for the strength to overcome the difficulties during the master's degree. Second, to my parents, Solange Freitas and Aurélio Viana, sister, Larissa Viana, and girlfriend, Raymara Almeida, for their care, support, concern for my well-being, and faith in me.

I also thank to my old pals from Universidade Federal do Rio Grande do Norte (UFRN), Débora Oliveira, Gustavo Alves (Koruja), Gustavo Silvino, Felipe Barbalho, Murilo Bento, Raul Silveira, and Patrícia Cruz. To my new ones from Universidade de São Paulo (USP), Marcos Gôlo and Adailton Araujo. Thank my other friends, Victor Lael, Allan Freitas, and Selan dos Santos. I also thank my therapist. Thank you all for the talks, advices, moments of relaxation, and your friendship. You were crucial to helping me face the challenge of completing a master's degree at home during the COVID-19 pandemic.

I acknowledge my fellow post-graduate Leonardo Tórtoro Pereira for discussing and helping me during the development of the algorithms of this work. And my advisor Prof. Dr. Claudio Fabiano Motta Toledo, for guiding me during this research. Finally, I acknowledge the financial support of the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

“So why we try so hard in this place?
When pain and suffering is a guarantee and happiness is a phase”
When We Were Younger, SOJA

RESUMO

VIANA, B. M. F. **Orquestrando e Adaptando Níveis de Calabouço, Missões de Portas Fechadas e Inimigos**. 2022. 101 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Técnicas de Geração Procedural de Conteúdo, ou *Procedural Content Generation* (PCG), podem ser usadas para gerar automaticamente o conteúdo de jogos ou aumentar a criatividade e a produtividade dos designers. Além disso, PCG pode funcionar como um recurso de jogo, fornecendo conteúdo diversificado e direcionado aos jogadores. Nesse contexto, abordamos o problema da orquestração de conteúdo adaptativo, especificamente explorando como coordenar a geração de níveis, missões e inimigos para um jogo de ação-aventura e diferentes tipos de jogadores. Assim, a presente dissertação de mestrado propõe um sistema de PCG para experiências de jogo com diferentes jogadores. Nosso sistema é focado em três diferentes facetas do jogo, níveis de masmorras, narrativas (missões) e regras (inimigos), e composto por três módulos, orquestrador, classificador e protótipo de jogo. O módulo orquestrador coordena dois algoritmos para gerar níveis e inimigos; ambos aplicam MAP-Elites para manter uma variedade de soluções sem perder qualidade. A abordagem de geração de níveis cria masmorras com inimigos (faceta de níveis) e missões de portas trancadas (faceta de narrativas). Por sua vez, a abordagem de geração de inimigos cria inimigos com diferentes atributos e comportamentos (faceta de regras). Em seguida, o módulo classificador recebe as respostas dos jogadores dadas a um breve questionário sobre suas preferências de jogo para categorizar seus perfis. Para adaptar os conteúdos, definimos objetivos diferentes de cada gerador para cada tipo de jogador. Em seguida, com base no tipo de jogador, o módulo orquestrador combina adequadamente os níveis e inimigos gerados anteriormente. Para isso, projetamos o orquestrador para filtrar e selecionar inimigos coerentes colocados nas salas dos níveis. O módulo de protótipo de jogo é onde validamos os conteúdos gerados pelo nosso sistema e coletamos dados dos jogadores. Nossos resultados mostram que os dois algoritmos MAP-Elites convergem com precisão quase toda a população na maioria das execuções e maioria dos casos. Os feedbacks dos jogadores mostram que gostaram dos níveis que jogaram e dos inimigos que enfrentaram. Além disso, a maioria deles não poderia indicar que um algoritmo criou os níveis ou os inimigos. Nosso sistema apresentou resultados positivos para entregar conteúdo adaptável de forma adequada para diferentes tipos de jogadores, por meio de um processo simples de criação de perfil de jogadores. Assim, podemos concluir que nosso sistema PCG pode gerar níveis e inimigos capazes de entreter diferentes jogadores.

Palavras-chave: Orquestração de Conteúdo, Geração de Níveis, Geração de Inimigos, Geração Adaptativa, MAP-Elites.

ABSTRACT

VIANA, B. M. F. **Orchestrating and Adapting of Dungeon Levels, Locked-door Missions, and Enemies**. 2022. 101 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Procedural Content Generation (PCG) techniques can be used to automatically generate game content or increase the designers' creativity and productivity. Besides, PCG can work as a game feature by providing diverse and targeted content for players. In this context, we tackle the problem of adaptive content orchestration, specifically by exploring how coordinate the generation of levels, missions, and enemies for an Action-Adventure game and different types of players. Thus, the present master's thesis proposes a PCG system to provide adaptive gameplay experiences for different players. Our system is focused on three different game facets, dungeon levels, narratives (missions), and rules (enemies), and it comprises three modules, orchestrator, classifier, and game prototype. The orchestrator module coordinates two algorithms for generating levels and enemies; both apply MAP-Elites to maintain a variety of solutions without losing quality. The level generation approach creates dungeons with enemies (levels facet) and locked-door missions (narratives facet). Next, the enemy generation approach creates enemies with different attributes and behaviors (rules facet). The classifier module receives the players' answers to a brief questionnaire regarding their gameplay preferences to categorize players' profiles. To adapt the contents, we defined different goals of each generator for each player type. Based on the player type, the orchestrator module appropriately combines the previously generated levels and enemies. We designed the orchestrator to filter and select coherent and good enemies to place in the levels' rooms. The game prototype module is where we validate the contents generated by our system and collect data from the players. Our results show that the two MAP-Elites algorithms accurately converge almost the whole population with many executions and cases. The players' feedbacks show that they enjoyed the levels played and the enemies faced. Besides, most of them could not indicate that an algorithm created the levels or the enemies. Our system presented positive results for delivering adaptive content properly for different types of players through a simple player profiling process. Thus, we can conclude that our PCG system can generate levels and enemies to entertain different players.

Keywords: Content Orchestration, Level Generation, Enemy Generation, Adaptive Generation, MAP-Elites.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

D	Down
KR	Key Room
L	Left
LR	Locked Room
NR	Normal Room
R	Right
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASP	Answer Set Programming
AVG	Average
BDI	Belief-Desire-Intention
BLX-α	Blended Crossover α
CA	Cellular Automata
CME	Constrained MAP-Elites
CNN	Convolutional Neural Network
CoG	Conference on Games
D-NSLC	Deluged Novelty Search Local Competition
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EDD	Evolutionary Dungeon Designer
FDG	Foundations of Digital Games
FI2Pop	Feasible-Infeasible Two Population
FPS	First-Person Shooter
GA	Genetic Algorithm
GG	Generative Grammar
GP	Genetic Programming
GVG-AI	General Video Game Artificial Intelligence
HCI	Human-Computer Interaction
ICME	Interactive Constrained MAP-Elites
MAP-Elites	Multi-dimensional Archive of Phenotypic Elites
MCNS	Minimal Criteria Novelty Search

ME	MAP-Elites
MMO	Massive Multi-Player Online
MMORPG	Massive Multi-Player Online Role-Playing Game
MUD	Multi-User Dungeon
NPCs	Non-Playable Characters
NS	Novelty Search
NSLC	Novelty Search Local Competition
PAE	Parallel Evolutionary Algorithm
PCG	Procedural Content Generation
PDG	Procedural Dungeon Generation
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
PvE	Player versus Environment
PvP	Player versus Player
QD	Quality Diversity
STD	Standard Deviation
UPEQ	Ubisoft Perceived Experience Questionnaire

CONTENTS

CHAPTER

1

INTRODUCTION

The game industry has become the most popular and profitable entertainment industry in the world (Game Terra, 2018; RICHTER, 2020). This industry reached U$139.9 billion in 2020, and it still presents a growth projection for the following years (TAKAHASHI, 2021). Newzoo forecasts the game market will surpass U$200 billion in 2023 (WIJMAN, 2021). The Brazilian game market is the world's 13th most extensive and Latin America's second one, and it had a revenue of about U$1.5 billion in 2019 (Koema, 2019; Statista, 2019).

The game industry's growth has encouraged the increase of investment in game development. For instance, *Grand Theft Auto V* (Rockstar Games, 2013), a.k.a. *GTA V*, is both the most expensive game (costing about $265 million) and the most profitable entertainment product ever made (reaching $ 6 billion in 2018 and remains increasing) (VALENTE, 2019; DONNELLY, 2018). *GTA V*, as well as other AAA games, has a vast map that required several game designers to develop it, and this justifies a big part of the investment of this game. However, new games tend to feature larger maps and more content that requires more game designers, and, consequently, it increases the development time and reduces the profit. Furthermore, indie game developers struggle to develop high-quality games due to limited investment. To solve these problems, both AAA and indie game developers use Procedural Content Generation (PCG) techniques to automatically generate game content to increase the designers' creativity and productivity and provide procedural generation as a game feature for their games.

The gaming audience has been interested in the procedural generation as a game feature since the release of the game *Rogue* (TOY; WICHMAN, 1980) that generated its levels aiming to save memory. After that, PCG become a game feature not only for level generation but also for other types of game content, e.g. weapons in *Borderlands* franchise (Gearbox Software, 2009) and in *Galactic Arms Race* (Evolutionary Games, 2014). The *Rogue*'s success also started two new game genres, the Rogue-like and the Rogue-lite genres. The main common feature of both *Rogue* derived genres is the dungeon generation. *Moonlighter* (Digital Sun, 2018) and *Dead Cells* (Motion Twin, 2018) are two recent examples of successful Rogue-like games.

Eventually, academic researchers started to explore PCG problems due to the challenges in content generation, e.g., the feasibility assurance of peculiar content features. Moreover, research is not restricted to game-related content, but there is also the generation of realistic environments in computer graphics research (HEWGILL; ROSS, 2004). Nevertheless, the research on procedural game content generation explores several contents, e.g., maps, levels, missions, items, weapons, stories, among others (TOGELIUS; SHAKER; NELSON, 2016). Newly, researchers are exploring the creation of multiple contents (facets) aiming to compose a whole game (LIAPIS *et al.*, 2019). These methods use algorithms to orchestrate (control) generative processes and create coherent combinations of such contents (LIAPIS *et al.*, 2019).

In level generation research, the Procedural Dungeon Generation (PDG) is one of the most commonly explored topics of research on PCG without signs of a downward trend (LIAPIS, 2020; VIANA; SANTOS, 2021). It is not a surprise since dungeons are very common scenarios of several game genres, e.g., Action, Adventure, Platform, Role-Playing, Rogue-like, Rogue-lite, among others. Linden, Lopes and Bidarra (2014) define dungeons as labyrinth environments that interrelate game features and affect the players' progression both in time and space. Besides, dungeons are composed of puzzles, rewards, and challenges (most battles with enemies). The enemies' goal is to hinder the players' progression by killing them. The way they hinder this progress is distinct and hence may require players to use specific strategies to defeat them.

In this context, our research tackles the problem of adaptive content orchestration. The PCG research literature presents some works on content orchestration; however, they did not orchestrate the generation of dungeon levels, missions, and enemies nor adapt them for players (see details in Chapter 3). Thus, to fill such a gap in, the present master's thesis proposes a PCG system to provide adaptive gameplay experiences for different players concerning their profiles. This master's project is part of a wider one of our research group, led by the Ph.D. candidate Pereira (2021). Thus, our PCG system is actually part of a bigger PCG system.

Following, we state the research question which guided us in this master's research:

> *How can a system orchestrate the generation of levels, narratives, and rules facets and adapt them to provide coherent combinations of contents for different types of players?*

To answer this question, we contributed to the PCG system in three modules: orchestrator, classifier, and game prototype. Our part of the system is focused on two different game facets: levels, narratives, and rules – see Section 2.3 for more details. The orchestrator module holds two procedural generation algorithms for levels and enemies. The level generation approach creates dungeons with enemies (levels facet) and locked-door missions (narratives facet) – see Section 2.2 for more details. Next, the enemy generation approach creates enemies with different attributes and behaviors (rules facet). Both algorithms apply MAP-Elites, a Quality Diversity (QD) approach that, like others, maintains a variety of solutions without losing quality. The use of such a Evolutionary Algorithm (EA) class is highly recommended for PCG systems

(GRAVINA *et al.*, 2019). In the classifier module, we define player types from answers given to a brief questionnaire regarding their gameplay preferences. To adapt the contents, we defined different goals of each generator for each player type. With such a classification, we appropriately combine the previously generated levels and enemies in the orchestrator module. To combine these contents, we designed the orchestrator to filter and select coherent and good enemies to place in the levels' rooms. The game prototype module is where we validate the algorithms developed and collect data from the players.

The results of our computational experiments show that the two of our algorithms accurately converge almost the whole population within executions for different cases. Regarding the players' feedback, our results show that most of the players enjoyed the levels played and enemies faced. We also successfully created enemies ranked as easy, medium, or hard to face. Besides, most players could not indicate that an algorithm created the levels or the enemies. Through a simple player profiling process, our system presented positive results for delivering adaptive content properly for different types of players. Thus, we conclude that our PCG system can generate levels and enemies to entertain players.

The remaining of this document is structured as follows. Chapter 2 presents the background concepts related to our research: procedural generation in games, evolutionary techniques of PCG, the game genre and its most common features in which we are interested, and how we can adapt content from players' data. Chapter 3 details the most important works in the literature related to this research. In Chapter 4, we present our PCG system and how their modules work and communicate. Furthermore, we describe the game prototype in which we intend to run experiments to validate our approach. Chapter 5 presents the results obtained in our computational experiments and the experiments with human players. Finally, in Chapter 6, we present our final remarks on our master's dissertation.

CHAPTER

2

BACKGROUND

This chapter presents the main concepts present in our research. Section 2.1 introduces the Action-Adventure game genre since it is the genre of our game prototype. Next, Section 2.2 presents the definitions of game features that are present in our game. Then, Section 2.3 briefly describes the history of PCG, the most accepted definition of PCG in the scientific community, the concepts related to the generation of multiple contents, and the PCG taxonomy. In Section 2.4, we overview the evolutionary algorithms, especially the ones applied on PCG, and we present the one we applied in our algorithms. Finally, Section 2.5 introduces the concepts of user classification; then, we describe how it can be applied in the game context to adapt content.

2.1 Action-Adventure Games

As we can assume by its name, an Action-Adventure game is a game genre that combines elements from Action and Adventure games (Video Game History Wiki, 2009), i.e., they feature combat, skill, and other physical challenges together with space exploration and item gathering. These games usually present an elaborated story around a character controlled by the player in a big world. Moreover, Action-Adventure games present non-trivial puzzles that are intrinsically related to the level map and, sometimes, also to their story. This genre presents various types of puzzles, e.g., temporal, cognitive, or dexterity puzzles.

There are several examples of successful Action-Adventure games, some of them are: *The Legend of Zelda* franchise (Nintendo, 1986) from which we highlight *The Legend of Zelda: Link's Awakening* (Figure 1a) and *The Legend of Zelda: Breath of the Wild* (the winner of The Game of the Year in The Game Awards 2017); *Uncharted* franchise (Naughty Dog, 2007) – from which we stand out *Uncharted: The Lost Legacy*; and, *Horizon Zero Dawn* (Guerrilla Games, 2017) (Figure 1b). The massive success of Action-Adventure games in the industry had reflected in the academy. Several researchers developed games of this genre to test their PCG algorithms for both level and puzzle generation (DORMANS, 2010; VALTCHANOV; BROWN, 2012;

SUMMERVILLE *et al.*, 2015; SUMMERVILLE; MATEAS, 2015; LIAPIS, 2017; LAVENDER; THOMPSON, 2017; SMITH; PADGET; VIDLER, 2018; PEREIRA; PRADO; TOLEDO, 2018).

Figure 1 – Screenshots of dungeons of Action-Adventure games.

(a) Link's Awakening (YU, 2008).

Source: Unity Forum[1].

(b) Horizon Zero Dawn (Guerrilla Games, 2017).

Source: MoGaming (YouTube Channel)[2].

2.2 Game Features

Although the differences between game genres, they share several types of game content. Game content refers to any features contained in games: levels, maps, textures, stories, items, missions, music, weapons, characters, among others (TOGELIUS; SHAKER; NELSON, 2016). Thus, to better understand game features, we present some basic concepts of some game contents related to Action-Adventure games.

A *level* is the playable space where the game happens. It can be represented in several ways depending on the game genre. Regarding games that try to reproduce natural environments or buildings, levels may be composed of continuous areas or discretized through *rooms*. Levels may present various features according to their types of scenarios, e.g., they may be represented as *dungeons*. According to Linden, Lopes and Bidarra (2014), dungeons are labyrinth environments composed of challenges, rewards, and puzzles interrelated with the play-space and time. Dungeons, therefore, provide highly structured gameplay experience (LINDEN; LOPES; BIDARRA, 2014). The dungeon topology presents an entrance, a *goal* that may be an exit or a final challenge, and a labyrinth as an intermediate area.

Furthermore, levels are composed of *obstacles* that try to prevent the player's progression; therefore, they must be beaten or solved by the player. These obstacles may be *enemies*, *traps* or *puzzles*. Enemies hinder the player's progression through actions, mainly by damaging the players' life points. These actions may be different for different enemies and may require specific strategies to defeat them. Puzzles usually require thinking, sorting, or searching for items in the

[1] Unity Forum: https://forum.unity.com/threads/new-links-awakening-level-design-tools.694978/.
[2] MoGaming (YouTube Channel): https://youtu.be/ioIBi6E2i10.

level's space to be solved. They typically do not have enemies, e.g., spatial puzzles[3]. *Barriers* are a particular type of obstacle because they temporarily block the player from reaching some game area. Barriers may be treated as puzzles once they usually require similar features to be unblocked (solved). The most common type of barrier in games is locked doors that are open by key items. However, as Figure 2 shows, a barrier may be the sea or a giant sleeping monster blocking a route that requires, respectively, the swimming/surfing skills or only a specific key item that makes such monster get out of the way to unlock other regions in the game. Thus, *items* are game features that allow the player to perform some temporary or permanent effects, leading to changes in the game state whether or not beneficial for gameplay, e.g., keys that unlock doors. A *mission* may be a single task, or it may be composed of a series of multiple tasks (or sub-goals). There are several types of missions, e.g., escort missions and collection missions. Therefore, we consider the collection of keys to unlock barriers as missions. Enemies, traps, puzzles, and missions provide different gameplay *challenges*, e.g., dexterity or cognitive challenges. These challenges must provide *rewards* for the player, such as coins, items, or skills.

Figure 2 – Barriers of *Pokémon* franchise (Nintendo, 1996).

(a) The Snorlax (a Pokémon) is sleeping and blocking the route, and it only can be woken up and unblock the route with a PokéFlute.

Source: Bulbapedia[4].

(b) The sea is not a passable area until players have a Pokémon with the surf skill, i.e., it prevents players from reaching another area.

Source: Bulbapedia[5].

2.3 Procedural Content Generation

In the 80s, neither personal computers nor consoles did not have a billion bytes available as they have nowadays. Therefore, software developers had to deal with this problem by providing robust functionalities through simple codes. This problem was more severe in games than other software since they could have several levels and other game features. To avoid this problem, *Rogue* (TOY; WICHMAN, 1980), a dungeon crawler game, and *Elite* (BELL; BRABEN, 1984),

[3] Spatial puzzles are solved by organizing spread blocks in specific places, for instance, Sokoban puzzles.
[4] PokéFlute: https://bulbapedia.bulbagarden.net/wiki/Pok%C3%A9_Flute.
[5] Surf (move): https://bulbapedia.bulbagarden.net/wiki/Cerulean_Cave.

space exploration and combat game, introduced algorithms capable of dynamically creating dungeon levels and planets, respectively. That was the rising of Procedural Content Generation.

The increase of computer capacities of data processing and graphic processing allows adding more content in games, e.g., creating bigger maps, and improving content quality. Thereby, the game development started to sophisticate, demanding new and more specialized professionals, and, consequently, it became increasingly expensive and required more development time. In this context, PCG can help to solve these issues since it can:

- decrease the need for human designers to generate content;
- increase both productivity and creativity of human designers through design suggestions;
- provide both control and balancing of the game difficulty or other features;
- act as a game feature by increasing the game replayability once it provides content variety.

Examples of games that use PCG as feature are: *Civilization VI*, which procedurally generates maps (Firaxis Games, 2016); *Moonlighter*, which generates dungeon levels (Digital Sun, 2018); *No Man's Sky*, which generates whole worlds, animals, among others (Hello Games, 2018).

So far, we have described the PCG history and the advantages of using it, especially in games. Nevertheless, we still did not define it properly, nor how its methods can be classified. Several researchers have defined Procedural Content Generation (PCG); however, the most accepted definition was proposed by Togelius, Shaker and Nelson (2016). According to them, PCG is a method of game content creation fully performed by computers or associated with human designers or gamers. Therefore, we can classify as PCG programs both an automatic generation library and software that provides suggestions of changes in human designers' levels.

PCG research has been growing in recent years (LIAPIS, 2020). Several research groups worldwide explore different generation approaches for creating different game content. Most of the recent works explore the generation of narratives, levels, rooms, among others (ONO; OGATA, 2018; PEREIRA *et al.*, 2021; GUTIERREZ; SCHRUM, 2020). Moreover, PCG approaches may generate multiple contents at the same time, such as levels with missions (SMITH; PADGET; VIDLER, 2018; PEREIRA *et al.*, 2021; GELLEL; SWEETSER, 2020). The works generating multiple contents are not new; however, the concept of coordinating multiple PCG systems is. Such a concept was introduced by Liapis *et al.* (2019), where they analyzed works that somehow create multiple contents, introducing the concepts of *creative facets* of games and *orchestration*. *Creative facets*, or just facets, regards what can be procedurally generated. A total of six different creative facets is defined:

Visuals determines the visual representation of the game in terms of rendering; the visuals of games range from photo-realistic to abstract.

Audio defines the feel and mood of the game through background music and sounds.

Narrative defines the main motivation to play games. They do not need highly elaborated stories and lengthy dialogues; they may have a simple sequence of events and missions.

Levels are the virtual space where the game takes place. They may be simple, as in Pong (Atari, 1972), or complex spaces, as labyrinths. Furthermore, games may have several short levels or just a single colossal level.

Rules define what a player can or cannot do in the game by determining the transition between game states after players use a mechanic. These mechanics allow players to interact with the play-space of the game.

Gameplay regards simulated experiencing games, i.e., is the process by which an agent interacts with a game. To better simulate, agents should mimic different types of players.

The game design process may require coordinating such facets. The authors defined *orchestration* as "the collaboration of multiple computational designers, each focusing on the creation of content primarily for one facet". Orchestration is a metaphor that came from music orchestration for the generative process of game design. The content orchestration process can be performed in two ways: top-down or bottom-up. In the former way, the compositions must provide as much detail as possible to the generators that work independently. In the latter way, the generators have individual responsibility (one facet) for the content they create and must evaluate how their content matches other generators' content to reach consensus (i.e., generate a coherent and playable combination of the contents they generated).

Furthermore, there are four intermediate approaches. Liapis *et al.* (2019) named these approaches based on terms of music. The first is the *Creative Maestro*, which is a flexible top-down approach. The maestro commands the generators to create content, but they can reinterpret these commands to generate better content. In this case, the modification propagates to the maestro and the other generators to ensure feasibility when combining the contents. The second approach is the *Jamming with Fake Sheets*, which consists in defining the necessary elements that compose the game structure and letting the generators build on and expand it without restriction regarding content. This one is the approach where our research fits in. In third, we have the approach of *Vertical Slices*, where contents are generated and may be upgraded through interactions to add a new game mechanic. However, when an iteration generates content that under-performs regarding the previous one, the previous version prevails while the new one is discarded. Finally, the fourth approach is *Post-production*. This approach evaluates the generated content and applies some repairs to smooth out errors or dissonances and incoherence among contents. The post-production can be performed together with the other approaches.

Besides the classification of creative facets and orchestration processes, PCG methods also have their taxonomy, defined by Togelius, Shaker and Nelson (2016). This taxonomy

has seven classifications of PCG approaches: Content Requirement, Outcome Randomness, Generation Time, Generation Control, Generality, Generation Method, and Content Audience – we entitled these classes in a published paper (VIANA; SANTOS, 2021).

Content Requirement classifies a game content as necessary or as optional, i.e., it defines the content is needed to finalize the game and the content that can be ignored by the player without penalization, respectively.

Outcome Randomness defines if a technique is deterministic, i.e., when a software generates the same content for the same parameters; or it is stochastic, i.e., when it always generates different contents even with the same parameters.

Generation Time determines if the content is generated offline, i.e., before the gameplay, or online, i.e., during the gameplay (i.e., while the player is playing the game).

Generation Control classifies the method regarding its degrees of control provided; it can be provided by a single random seed (a single dimension of control) or by a set of parameters (several dimensions of control).

Generality defines the target audience for which the content was generated; it is categorized as an adaptive generation when the generator can learn the gamers' tastes and use it to generate targeted content; otherwise, it is generic generation.

Generation Method determines how the generation is carried out; a method is constructive if it generates the content at once in a single execution, or generate-and-test if it performs tests to validate the content feasibility.

Content Authorship classifies as an automatic generation if the content is generated only by computers or as mixed-initiative if generated with human support.

Independently of the PCG classification, there are several ways of generating content. For instance, some constructive approaches are usually specially designed according to the type of content they intend to generate. Nevertheless, search-based approaches, especially the evolutionary ones, are widely used to develop PCG methods to provide feasibility and variety.

2.4 Evolutionary Computation

This section presents an overview of Evolutionary Computation (EC) approaches usually applied to PCG methods. EC is a family of stochastic metaheuristic algorithms for search, and global optimization inspired by theories evolution from biology (EIBEN; SMITH *et al.*, 2003). These algorithms present the same structure:

population corresponds to a set of individuals that represent the solutions for the tackled problem (an evolutionary algorithm may have multiple populations);

fitness function evaluates the individuals' quality (or capacity of survival);

selection operator selects the parent individuals that will reproduce to generate new individuals (it may be random or based on their qualities);

reproduction operator breeds an offspring of new individuals;

survival operator selects the individuals that will survive based on their qualities (i.e., the fittest individuals will remain in the population for the next generation); and,

stop criterion defines when the evolutionary process stops (the most common stop criterion is set with a fixed number of generations).

Algorithm 1 presents the basic evolutionary process: a stochastic method creates the initial population; this population reproduces through the reproduction operators (recombination and mutation) after selecting individuals as parents; then the survival criteria filter the individuals that will stay alive in the population; the reproduction and survival processes repeat until the stop criterion is reached. One of the most known EC approach is the Genetic Algorithm (GA), which is inspired by the *natural selection*, *mutation* and *crossover* over *chromosomes* (individuals) from the neo-Darwinian theory of evolution (REEVES, 2010).

Algorithm 1 – Basic Evolutionary Algorithm. Adapted from (EIBEN; SMITH *et al.*, 2003).

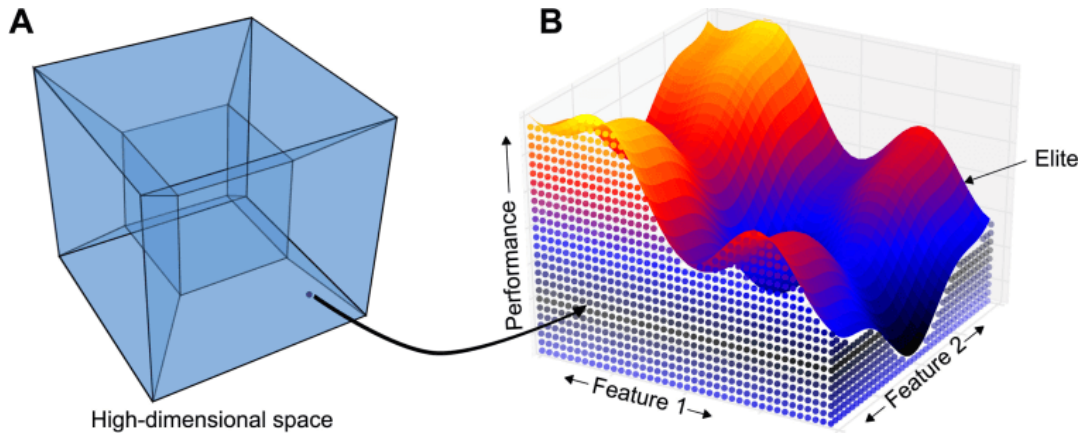
```
1:  procedure EVOLUTIONARYPROCESS
2:      INITIALIZE population with random individuals;
3:      EVALUATE each individual of the population;
4:      while STOP CRITERION not satisfied do
5:          SELECT parents from the population;
6:          RECOMBINE pairs of parents;
7:          MUTATE the resulting offspring;
8:          EVALUATE the new individuals;
9:          SELECT individuals for the new generation;
10:     end while
11:     return the best individual;
12: end procedure
```

The first EC approaches intend to find the single best individual in the search spaces. Thus, the individuals' similarity is intensified, hindering searching for the best global individual. One solution for this problem is the Feasible-Infeasible Two Population (FI2Pop), which evolves two populations of feasible and infeasible individuals and allows interbreeding between them (KIMBROUGH *et al.*, 2005). Besides, researchers found that different return solutions could also be attractive for some problems. For instance, it is especially interesting for procedural generation methods (GRAVINA *et al.*, 2019). Therefore, to solve this problem, Pugh, Soros and

Stanley (2016) introduced a new class of algorithms called Quality Diversity, aiming to maintain the diversity of solutions without losing quality. They introduced a function to calculate the distance of similarity of the solutions, which measures the population diversity.

Mouret and Clune (2015) introduced the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites), or just MAP-Elites (ME), the method we applied for our PCG approaches. The MAP-Elites approach discretizes the search space into a matrix (*feature map*), where each matrix's cell corresponds to the best individual (*elite*) of the mapped features, such characteristic was named as *illumination*. To discretize the search space, the authors introduced the concept of *feature descriptors*. Feature descriptors are lists of values corresponding to an individual's attribute or calculated property. Figure 3 shows a visual example of the MAP-Elites population; in this case, the performance ax refers to the solutions' quality. The initial population is usually created by generating random solutions. If the cell is empty, the individual is stored in the cell. Otherwise, there is a *local competition* between the individuals and the best one stored in the cell. The elite individuals are selected as parents of the recombination and or mutation operations in the offspring process. Inspired by the FI2Pop approach, Khalifa *et al.* (2018) introduced the Constrained MAP-Elites (CME), a variation of ME that evolves two populations of feasible and infeasible individuals for each cell.

Figure 3 – The MAP-Elites algorithm tries to find the best solution (or performance) for each map's point (i.e., low-dimensional feature space) searching in the high-dimensional space.The user chooses two features to define the interested space dimensions, which discretizes the search space.



Source: Mouret and Clune (2015).

2.5 Player Profiling and Content Adaptation

So far, we have explained game features, content generation, and methods of generating content. However, we need to understand their preferences independently of the method to adapt content for players. Thus, this section describes how to classify players and how it could be applied in content adaptation.

Regarding general systems, Jameson (2008) established some ideas about understanding systems' usability and adapting these systems to their users. There are two methods of data collection: explicit input via self-reports and self-assessments; and non-explicit input. Self-reports aim to objectively collect users' personal properties (e.g., age, profession, and state). Self-assessments aim to collect general data to obtain insight into the users' interest in a topic and knowledge about it. Finally, non-explicit inputs can be collected from users' experiences while using systems for the long run, e.g., using sensors to measure their reactions. These methods are widely applied by Human-Computer Interaction (HCI) field (JAMESON, 2008). For instance, Orji, Nacke and Marco (2017) conducted experiments with 660 people based on both implicit and explicit input and identified personality traces that can define if users would keep using health games or gamified systems.

In the game context, explicit input can be applied to external or in-game questionnaires, and the implicit input is the gameplay data. However, in-game questionnaires and gameplay data are more suitable for content adaptation of digital games. The data gathered can be used to, for instance, classify players' preferences regarding battling, explore and provide more (fewer) harder (easier) enemies, as well as levels with higher (lesser) degrees of exploration, (BARTLE, 2004; VAHLO *et al.*, 2017). In the next chapter, we describe the related works of literature that somehow tried to classify players and or tried to provide adapted content.

CHAPTER

3

LITERATURE REVIEW

This chapter presents the literature review. First, in Section 3.1, we present the works on player profiling and adaptive content generation. Next, we describe works that focused on generating levels and enemies, respectively, in sections 3.2 and 3.3. Finally, in Section 3.4, we review works applying PCG to generate multiple contents. The related works were found in Google Scholar without applying systematic review criteria.

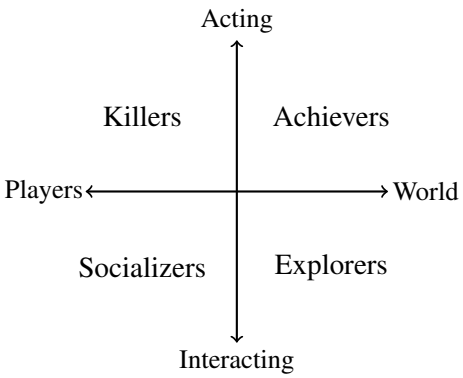
3.1 Player Profiling and Content Adaptation

To provide adaptive content for players, we need to identify what they enjoy while playing games. To do so, we need ways to collect data from players and gather information from them. Therefore, this section presents works that somehow classified players and some works that used such classification to provide adapted content for players.

Bartle (2004) attempted to identify personality traces and profiles from Multi-User Dungeon (MUD) and Massive Multi-Player Online Role-Playing Game (MMORPG). The work defined four categories of players based on two axes (Figure 4). One from *interacting* to *acting*, and the another from *players* to *world*, i.e., *players* may prefer to *interact with* or *act on* the *game world* or *other players*. Each quadrant corresponds to one player type: *achievers*, *explorers*, *socializers*, and *killers*. Achievers prefer acting in the game world to achieve goals, e.g., gain points, equipment, and other measurable things. Explorers prefer interacting with the game world by discovering areas and immersing themselves in the world. Socializers also prefer interacting, but with other players or Non-Playable Characters (NPCs), i.e., they like the social aspect instead of the game world. Finally, killers prefer competing and combating other players.

However, some researches contradicted the results of Bartle (2004). For instance, Yee (2006) created similar categories for Massive Multi-Player Online (MMO) players' motivations through explicit data collection. They performed an online survey with a questionnaire composed

Figure 4 – Bartle's player types chart.



Source: Adapted from Bartle (2004).

of 40 questions based on Bartle (2004). The authors describe three primary motivations:

Achievement derived from advancement, mechanics, and competition;

Social derived from socializing, relationship and teamwork;

Immersion derived from discovery, role-play, customization and escapism.

The results showed that grouping players by their motivations were better than dividing them into exclusive categories. One of their main findings is that these motivations are not only, i.e., a player could also be motivated simultaneously and equally by achievement and immersion.

Vahlo *et al.* (2017) developed a new player classification based on thousands of player preferences. They identified the contents and keywords from a qualitative analysis of 700 written reviews on digital games. Next, all findings are divided into 33 core game dynamics, from which they created a 7-point Likert questionnaire of 33 items, and the user had to answer how pleasant they found each game dynamic. A total of 2594 volunteers attended the experiment, and factor analysis was applied over the results to group the dynamics into five preference factors:

Assault killing, destroying, shooting, running, and others;

Manage acquiring and managing resources, expanding buildings, upgrading items, and others;

Journey exploring, uncovering secrets, making decisions, making in-game friends, and others;

Care taking care of pets and having romantic relationships with characters;

Coordinate puzzle-solving, precise platforming, rhythmic actions, and music-related features.

These factors allowed clustering seven different players' groups, where four of these groups were interested mainly in only one factor (except Journey). The remaining groups preferred mixes of two factors, which corroborates with the results found by Yee (2006).

Melhart *et al.* (2019), differently from the previous works, collected in-game data to empirically validate the Ubisoft Perceived Experience Questionnaire (UPEQ), developed by Azadvar and Canossa (2018) and composed of 21 4-point Likert questions. They collected data from 298 anonymous players that played *Tom Clancy's The Division* (Massive Entertainment, 2016) from 2016 to 2018. Data analysis was carried on to predict the players' motivations and feelings. After the implicit data collection, UPEQ is sent to the players to apply the k-means clustering algorithm identifying Four players types:

Adventurers did not find their preferred niche of play because they are still figuring out the game's systems (i.e., new players);

PvE All-Rounders like cooperative Player versus Environment (PvE) missions;

Social DarkZoners prefer to interact and battle with other players – i.e., they prefer Player versus Player (PvP);

Elites have vast preferences in the game.

Aiming to learn the players' tastes and behaviors, the authors used the collected data to feed an Artificial Neural Network (ANN) combined with Radial Basis Function kernels. Nevertheless, some players diverged their reported tastes from the actual gameplay data. Their approach presented issues on classifying some characteristics and did not accurately predict some subjective elements. They concluded that, although it is a challenging task, it is possible to group players and forecast their preferences, and it should be necessary to model the players directly.

We have described just ways of gathering data from players so far. However, some researchers used the collected information to provide adaptive content for players. Bicho and Martinho (2018) developed a PCG approach capable of generating levels online for Infinite Runner games with adaptive difficulty. The concept of game flow supports their approach to provide challenges increasingly hard for the players. Instead of collecting data to identify players' types, they collect gameplay data to better adapt the game difficulty to the players' skills. Obstacles control the game's difficulty, and there are two ways to avoid them: jumping over them or using the dash skill to traverse objects. Since they were not trying to identify the player type, their profiling approach was simple. When the players use one way to overcome an obstacle using the same strategy, it will be more challenging to avoid the next one. Thus, their approach was capable of generating adaptive content according to the identified favorite mechanics of players. Moreover, they found that the players usually apply their favorite mechanics, even it would be easier to change them. These results encourage the research for more types and more complex procedurally generated game contents based on the players' preference.

Heijne (2016) also collected explicit and implicit data for player profiling and used it in the adaptive content generation process. The players played a clone of *The Legend of Zelda: a Link to the Past* (Nintendo, 1986) they developed (HEIJNE; BAKKES, 2017), which

is called *Procedural Zelda*. This game can generate levels, puzzles, and place enemies based on the players' performance. Nevertheless, the generation processes are limited in terms of diversity and difficulty adjustment. The *Procedural Zelda* presents two pre-questionnaires, one for demography (composed of eight questions) and the other for personality testing (consisting of 120 questions), both in the format of self-report about objective personal characteristics. The players' interaction with NPCs, exploration of levels, puzzle-solving, and fights with enemies are recorded implicitly. After the gameplay, the game presents a post-questionnaire composed of 41 questions: game difficulty, the difficulty of each component, the player's behavior, and preferences regarding each element. Heijne (2016) gathered data from 25 players and classified them into three groups:

Hardcore spend many hours a week playing in harder difficulties;

Casual do not spend much time playing in easier difficulties;

Ambiguous fit in the middle of both categories mentioned above.

They analyzed the correlations among classifications, personalities, and the players' performances with four main findings:

1. overall game experience correlates with the understanding of using the mechanics;
2. performance metrics do not correlate with explicitly provided preferences on mechanics;
3. the best indicators for performance where the players' previous experience in similar games and preferring for harder difficulties;
4. players who start exploring a lot tend to eventually lose interest in exploration and focus only on the main path.

Although the satisfactory reported results, the personality questionnaire they applied presented 120 questions that could not accurately identify the players' preferences in terms of game content and difficulty. The respondents may answer inaccurately huge questionnaires or even give up on them (JAMESON, 2008).

Therefore, when we use questionnaires to define the players' profiles, we have to provide little questions to gather high-quality data. For instance, aiming to simulate players' behaviors, Rivera-Villicana *et al.* (2018) create the Belief-Desire-Intention (BDI) model, which is composed of a questionnaire with ten questions, semi-structured interviews, and gameplay data. They first collected data from 23 players to create models to simulate their behaviors. Next, to evaluate their models, they ran them in a Point-and-Click interactive fiction and compared them with an uninformed player model, which did not mimic any player style. The implicitly gathered gameplay data was more accurate than the explicitly collected data, which may mean that the

implicit data gathering can be more reliable. Their questionnaire is still a sub-optimal way to measure the players' styles. The latter finding of Rivera-Villicana *et al.* (2018) is interesting because other works on profile identification, through short questionnaires and in-game data, often present indecisive results (KONERT *et al.*, 2014; LORIA; MARCONI, 2018).

Bontchev and Georgieva (2018) showed that it is possible to develop an approach capable of creating adaptive content according to the players' emotions and gameplay data. Their method applies multiple linear regression for estimating player styles. However, modeling players from their feelings, personality, and in-game activity is not a simple task since external sensors are necessary to measure their sensations, e.g., cameras filming their faces and heart rate sensors. The sensors like those present theoretical potential for player profiling and content adaptation purposes, but there is still no evidence that they work. Moreover, these sensors are based on techniques from psychology and demand help from a specialist to develop the models of players (COWLEY; CHARLES, 2016).

Table 1 summarizes the reviewed works of this section regarding the use of questionnaires, gameplay input, emotions, and if they adapted content for players. As we observe, most of them did not try to adapt content; however, they focused on player type classification. Note that we disregard the work of Bontchev and Georgieva (2018) in the table since they did not perform classification from players' emotions nor content adaptation; they only showed that it is possible. Thus, in this section, we have pieces of evidence that robust player profiling approaches may provide the information needed to offer customized content for players. Inspired by these works, our work applies a pre-gameplay questionnaire to classify players and, then, at finishing each level, the players are reclassified. In this way, we believe we can provide target-designed and customized content for the players.

Table 1 – Player profiling and content adaptation literature summarizing and comparison with our work.

Work	**Questionnaire**	**Gameplay**	**Emotions**	**Adapt Content**
Bartle (2004)	-	✓	-	-
Yee (2006)	✓	-	-	-
Vahlo *et al.* (2017)	✓	-	-	-
Melhart *et al.* (2019)	✓	✓	-	-
Bicho and Martinho (2018)	-	✓	-	✓
Heijne (2016)	✓	✓	-	✓
Rivera-Villicana *et al.* (2018)	✓	✓	-	-
This work	✓	-	-	✓

Source: Elaborated by the author.

3.2 Level Generation

Several works dealt with level generation (LIAPIS, 2020), but we are particularly interested in those which dealt with dungeons. Thus, in this section, we focus on PDG works.

Here, we present some of the works found in our survey (VIANA; SANTOS, 2021) and also some other related works. We carried out the survey based on the guidelines of the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) model (MOHER *et al.*, 2009; LIBERATI *et al.*, 2009) - See (VIANA; SANTOS, 2021) for details of the survey process.

Dormans (2010) developed a two-step method with a constructive approach that uses Generative Grammar (GG). The solution creates a graph of missions and applies it to generate the play-space for Action-Adventure games. This approach generated two kinds of missions: defeating enemies and locked-door missions. Inspired by Dormans (2010), Lavender and Thompson (2017) applied the Dormans' approach to creating mission graphs and turn them into levels. They used the generated levels as dungeons in the open-source 2D game *The Legend of Zelda: The Mystery of the Solarus* (Solarus, 1986).

Similar to Dormans (2010), Linden, Lopes and Bidarra (2013) presented a GG-based solution for level generation. The work uses gameplay as a vocabulary to control the generative process. The player's gameplay actions are described semantically, e.g., "fight melee enemy" and "pickup health potion". The nodes of their graphs express player actions as gameplay design constraints. Thus, the gameplay grammar allows designers to specify their expected gameplay. This grammar generates graphs of player actions used as a base to create levels for *Dwarf Quest* game (Wild Card Games, 2012). This work inspired Karavolos, Liapis and Yannakakis (2016) that introduced a search-based algorithm capable of evolving gameplay graphs. In their work, the graphs are also translated to *Dwarf Quest*'s levels.

Smith, Padget and Vidler (2018) also introduced a solution for dungeon generation and locked-door missions with enemies as challenges, but their solution also presents placement of weapons. Similar to the previous works, their dungeon levels are also represented by graphs. However, they modeled the graph generation problem as an Answer Set Programming (ASP) problem formulation to produce dungeon level.

Pereira, Prado and Toledo (2018) presented a search-based algorithm focused on generating dungeon levels with locked-door missions for Action-Adventure games. The missions' goals are to collect keys to open locked doors in the levels and find a symbol (the levels' goals), similar to Zelda's triforce. Unlike the previous works, tree structures represent dungeon levels to ensure feasibility, and Genetic Programming (GP) approach evolves such levels and missions. The keys are in the tree's nodes (rooms), and the doors are in the tree's edges (corridors). Although the tree structure ensures the generation of feasible levels, it cannot generate some level structures, e.g., cycles. Furthermore, their dungeon representation also has semantic information for mission purposes. Later, the authors improved this approach by updating their fitness to maximize the visited rooms and unlocked doors (PEREIRA *et al.*, 2021).

Gellel and Sweetser (2020) hybridize generative grammars, inspired by Dormans (2010), and a non-traditional Cellular Automata (CA) approach based on random neighbor selection to generate dungeon levels for Rogue-like games. First, the GG generates the locked-door missions

and various types of rooms codified as a string. After that, they applied the CA-inspired method to generate the play-space. The play-space, represented as a grid, is divided into regions (or subsections as they called them) based on the locked doors. This method is composed of two rules that place the rooms for each character of the mission string. The first rule places one room at a time by moving randomly in the grid, while the second one calculates which neighbor of the active room is the closest (based on the Manhattan distance) to the region center point. The latter rule allows the level to appear growing around a natural cluster. They proposed two different ways of searching for these connections; one always uses the second rule, and the other applies the first rule until finding a dead-end room.

Differently from the previous works that performed mission generation, Baldwin *et al.* (2017a) developed the Evolutionary Dungeon Designer (EDD) to assist game designers in their creative process. The EDD generates dungeon rooms with enemies and rewards through a FI2Pop GA approach. Their rooms are represented as grids and are generated to match the micro-game patterns regarding levels' structure defined by users in the approach's input. To do so, EDD provides to the user control of frequency, shape, and type over the generated design patterns and the placement of enemies, treasures, doors, and walls.

Works like those of Dormans (2010), Lavender and Thompson (2017), Linden, Lopes and Bidarra (2013), Karavolos, Liapis and Yannakakis (2016), Smith, Padget and Vidler (2018), Gellel and Sweetser (2020), and Baldwin *et al.* (2017a) introduced approaches that generate levels and also perform the placement of enemies. Besides them, Baldwin *et al.* (2017b) and Liapis (2017) presented more sophisticated methods for enemy placement.

Baldwin *et al.* (2017b) extended the EDD to evolve the placement of enemies and rewards based on patterns. To do so, they introduced meso-game patterns that consider enemies and rewards besides the level structure. These meso-patterns represent four types of rooms: guard chamber, a room with only enemies; ambush, a room with enemies and an entrance; treasure chamber, a room with only a treasure; and guarded treasure, a room with enemies and a treasure.

Liapis (2017) introduced a two-step automatic evolutionary process for dungeon generation. Similar to the previous work, this approach applies FI2Pop GA, in both steps. The first stage generates dungeon sketches by strategically placing eight segments representing dungeon rooms, which describe impassable (wall), passable rooms and define the number of enemies and rewards in each room. In the second step, each segment becomes a room. Each room evolves independently to create a cavern environment, following connections between the segments and their types. For instance, the enemies are strategically placed around a reward. The solution is very interesting because it returned a wide variety of levels. The rooms in the levels had several combinations from seven types of rooms and four connections to other rooms.

So far, we described works on PDG that focused on generating the best dungeon levels, where some of them could have missions and or enemies. Therefore, we present works that applied QD approaches to generate dungeons.

Melotti and Moraes (2018) introduced a new QD approach by combining the Novelty Search Local Competition (NSLC) and Minimal Criteria Novelty Search (MCNS). Novelty Search (NS) introduced the concept of niches (sub-populations) and the diversity measurement to ensure the diversity of solutions; the competition between the solutions is global. Therefore, NSLC introduced the local competition to limit it between the solutions in their respective niches. On the other hand, MCNS introduced the minimum criterion dependent on the fitness score. The solutions that meet this criterion receive the novelty score usually, whereas the others receive zero. The solutions with zero can only reproduce if no other solution reaches the minimum criteria. Their approach combines the local competition of NSLC and the minimum criteria of MCNS to eliminate entire niches instead of only some solutions. This approach also adds new niches, and they found new types of solutions. Their new approach is named as Deluged Novelty Search Local Competition (D-NSLC). To validate it, they evolved simple dungeon levels with four different EAs: a usual GA, NS, NSLC, and D-NSLC.

Alvarez *et al.* (2019) extended the EDD and the evolutionary algorithm by providing several suggestions of changes to the user as a matrix of rooms. This feature was possible due to the Interactive Constrained MAP-Elites (ICME), which they introduced. This method can generate suggestions based on the room during the edition process. Each user modification leads to the generation of new suggestions, and it is possible to map the matrix of suggestions into linearity, symmetry, and other specific metrics of the EDD. In this version, the designers can change the population's levels and activate the evolutionary process to generate new levels.

Charity *et al.* (2020) introduced an automatic method to generate rooms for general games. Their method applies Constrained MAP-Elites to illuminate the rooms regarding the mechanics of a game. They do this by mapping the game mechanics into the CME's matrix. For instance, in a matrix with four elements, one of the matrix's axis may be labeled with "no 'get key"' and "get key" and the other one with "no 'kill enemy"' and "kill enemy", then the matrix is composed of four different cells as follows:

- "no 'get key"' and "no 'kill enemy"';
- "no 'get key"' and "kill enemy";
- "get key" and "no 'kill enemy"', and;
- "get key" and "kill enemy".

They applied this approach in four different games inspired by successful industry games: Zelda, Solarfox, Plants, and RealPortals. This mapping of different mechanics for different games is possible due to the use of General Video Game Artificial Intelligence (GVG-AI) framework since it allows their approach to deal with all the games' mechanics. They also use the GVG-AI to generate the initial population and to calculate the fitness functions with the framework's agents.

The fitness functions are based on the survival and conclusion time of agents. They claim that the generated rooms can be used as tutorials to teach players how to use the game mechanics.

Before the applying ICME in EDD (ALVAREZ *et al.*, 2019), Alvarez *et al.* (2018) extended the tool and the evolutionary algorithm to maintain the designers' aesthetics. They introduced functionality to "freeze" some of the levels' tiles, i.e., the selected tiles cannot change during the evolution. Besides tiles, users also could freeze shapes, patterns, routes, and meso-game patterns. Furthermore, this approach also measures similarity and symmetric to learn the design aesthetics and provide adaptive suggestions.

Table 2 summarizes the reviewed works on level generation methods and compares them with our level generation approach. Our method extends the algorithm introduced by Pereira *et al.* (2021) by applying MAP-Elites to illuminate the dungeons. Besides generating levels with locked-door missions, we place enemies within the levels' rooms (the number of enemies is given as input). Thus, we can generate multiple and diverse levels with different degrees of exploration coefficient and leniency in a single execution. Moreover, differently from the works described in this section, we also adapt the levels to different types of players.

Table 2 – Level generation literature summarizing and comparison with our work.

Work	**Level/Room**	**Missions**	**Enemies**	**Adaptive**	**QD**
Heijne (2016)	Level	-	✓	✓	-
Dormans (2010)	Level	✓	✓	-	-
Lavender and Thompson (2017)	Level	✓	✓	-	-
Linden, Lopes and Bidarra (2013)	Level	✓	✓	-	-
Karavolos, Liapis and Yannakakis (2016)	Level	✓	✓	-	-
Smith, Padget and Vidler (2018)	Level	✓	✓	-	-
Pereira, Prado and Toledo (2018)	Level	✓	-	-	-
Pereira *et al.* (2021)	Level	✓	-	-	-
Gellel and Sweetser (2020)	Level	✓	✓	-	-
Baldwin *et al.* (2017a)	Room	-	✓	-	-
Baldwin *et al.* (2017b)	Room	-	✓	-	-
Liapis (2017)	Level	-	✓	-	-
Melotti and Moraes (2018)	Level	-	-	-	✓
Alvarez *et al.* (2019)	Room	-	✓	-	✓
Charity *et al.* (2020)	Room	-	✓	-	✓
Alvarez *et al.* (2018)	Room	-	✓	✓	-
This work	Level	✓	✓	✓	✓

Source: Elaborated by the author.

3.3 Enemy Generation

Enemies in PCG research focus mainly on the placement of enemies with different amounts, difficulties, and types. In the previous section, we described some works that performed such enemy placement (DORMANS, 2010; LAVENDER; THOMPSON, 2017; LINDEN; LOPES; BIDARRA, 2013; KARAVOLOS; LIAPIS; YANNAKAKIS, 2016; SMITH;

PADGET; VIDLER, 2018; GELLEL; SWEETSER, 2020; BALDWIN *et al.*, 2017a; BALDWIN *et al.*, 2017b; LIAPIS, 2017; ALVAREZ *et al.*, 2019; ALVAREZ *et al.*, 2018). However, the research on procedural enemy generation is recent, with few papers introducing approaches of such kind (KHALIFA *et al.*, 2018; PEREIRA; VIANA; TOLEDO, 2021). Thus, this section describes enemy generation methods from academic papers and industry games regarding any stage of their generative processes and gameplay progress.

Khalifa *et al.* (2018) 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 they created to describe their levels, through Constrained MAP-Elites. 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 defines 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. Moreover, the authors generated, in a single execution, a variety of levels without losing quality by using a QD approach.

Nevertheless, enemies are more than the place they are and their behavior. As far as we know, The first work that generated enemies for Action-Adventure games was developed by our research group (PEREIRA; VIANA; TOLEDO, 2021) by introducing the Parallel Evolutionary Algorithm (PAE) for enemy generation in a rogue-like game prototype. The authors extracted the most common variables from enemies in different Action-Adventure games to build our enemy's genotype: health, damage, attack speed, movement speed, active time, rest time, movement type, weapon type, and projectile speed. PEA evolves enemies matching their difficulty degrees with the difficulty goal given by the user as an input.

Regarding the industry games, the creation of NPCs is present in *Spore* (Maxis™, 2008) and *No Man's Sky* (Hello Games, 2018). 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. 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* (Creature Labs, 1996) generated NPCs by evolving them physically and making them learn. These NPCs learn about the environment, and the player's actions via a neural network that receives simulated senses using semi-symbolic approximation techniques as input (GRAND; CLIFF, 1998).

Regarding the generation of actual enemies, i.e., NPCs that actively look for fighting players, *Diablo 3* (Blizzard, 2012) and *Shadow of Mordor* (Monolith Productions, 2014) created their enemies by changing some predefined characteristics. This approach allowed these games to make more diverse and unique challenges. In *Left 4 Dead 2* (Valve, 2009) and *State of Decay 2* (Undead Labs, 2018), the enemies can adapt to the players. However, instead of changing

their features, like in the previous two games, *Left 4 Dead 2* (Valve, 2009) and *State of Decay 2* (Undead Labs, 2018) 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].

Table 3 summarizes the reviewed scientific works and commercial games in this section, according to the generated features, and compares them with our enemy generation method. Besides placing a certain amount of enemies in our level generation approach, the enemy generation method extends our first contribution in such research area (PEREIRA; VIANA; TOLEDO, 2021). We apply MAP-Elites in this approach to illuminate enemies; thus, we can generate enemies with different movement and weapon types. Furthermore, we also adapt them for players providing enemies with different difficulties.

Table 3 – Enemy generation literature summarizing and comparison with our work. 'P' defines the partially generated enemy features.

Work	Placement	Amount	Status	Visuals	Adaptive	QD
Heijne (2016)	✓	-	-	-	✓	-
Dormans (2010)	✓	-	-	-	-	-
Lavender and Thompson (2017)	✓	-	-	-	-	-
Linden, Lopes and Bidarra (2013)	✓	-	-	-	-	-
Karavolos, Liapis and Yannakakis (2016)	✓	-	-	-	-	-
Smith, Padget and Vidler (2018)	✓	-	-	-	-	-
Gellel and Sweetser (2020)	✓	-	-	-	-	-
Baldwin *et al.* (2017a)	✓	✓	-	-	-	-
Baldwin *et al.* (2017b)	✓	✓	-	-	-	-
Liapis (2017)	✓	-	-	-	-	-
Alvarez *et al.* (2019)	✓	✓	-	-	-	-
Charity *et al.* (2020)	✓	✓	-	-	-	-
Alvarez *et al.* (2018)	✓	✓	-	-	✓	-
Khalifa *et al.* (2018)	✓	✓	P	P	-	✓
Pereira, Viana and Toledo (2021)	-	✓	✓	P	✓	-
Spore (Maxis™, 2008)	-	-	P	P	-	-
No Man's Sky (Hello Games, 2018)	-	-	P	P	-	-
Creatures (Creature Labs, 1996)	-	-	✓	P	✓	-
Diablo 3 (Blizzard, 2012)	-	-	P	-	-	-
Middle Earth: Shadow of Mordor (Monolith Productions, 2014)	-	-	P	-	-	-
Left 4 Dead 2 (Valve, 2009)	✓	✓	-	-	✓	-
State of Decay 2 (Undead Labs, 2018)	✓	✓	-	-	✓	-
This work	✓	✓	✓	P	✓	✓

Source: Elaborated by the author.

[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->).

3.4 Generation of Multiple Content

We have described some dungeon and enemy generation works in the previous sections. The PDG works obviously fit in the levels facet orchestration, and the enemy generation works fit in the rules facet orchestration since they define enemies' behavior. The PDG works that places enemies in their levels are not classified as rules facet orchestration; the enemy placement fits in levels facet orchestration (LIAPIS *et al.*, 2019). However, some of them also perform the generation of multiple contents by creating levels and narrative facets through the generation of dungeons with locked-door missions (DORMANS, 2010; LAVENDER; THOMPSON, 2017; LINDEN; LOPES; BIDARRA, 2013; KARAVOLOS; LIAPIS; YANNAKAKIS, 2016; SMITH; PADGET; VIDLER, 2018; PEREIRA; PRADO; TOLEDO, 2018; PEREIRA *et al.*, 2021; GELLEL; SWEETSER, 2020), with mechanics (CHARITY *et al.*, 2020). In this section, we present the works that somehow introduced multi-faceted content generation systems; some of the works were described first for Liapis *et al.* (2019).

All creative facets were found in PCG systems; however, none combined all facets. Accordingly to Liapis *et al.* (2019). The first PCG system known in the literature is *Ludi*, and it was developed by Browne and Maire (2010). This system defines the board layout (levels facet) and rules of different pieces (rules facet) for combinatorial games applying an evolutionary algorithm. The quality of the games the tool creates is measured by simulating playthroughs with artificial agents (gameplay facet). Similarly, *Mechanic Miner*, a PCG tool introduced by Cook *et al.* (2013), generates simple game mechanics (rules facet) and levels (levels facet) that have such mechanics, both through search-based approaches. The quality of the generated levels is evaluated regarding gameplay by random agents (gameplay facet). Rules were also orchestrated together with visuals and audio by Hoover *et al.* (2015) in the *AudioInSpace* game. In the game, players can control the evolution of their favorite weapons interactively, controlling their bullets' trajectory, speed, and color (rules and visuals facet). Furthermore, bullets and players' firing actions affect the audio in the game (audio facet).

Orchestrating applications have also generated narratives. Hartsook *et al.* (2011) introduced the GAME FORGE, a PCG tool that orchestrates narratives (through missions) and levels facets. Narratives are represented by sequences of actions made by the game hero and NPCs; once created, they guide the generation of levels. The system carries out the generation process through two search-based approaches to evolve both narratives and levels. In *Game-O-Matic*, a tool developed by Treanor *et al.* (2012), users can build narratives with graphs (narrative facet). With the narrative, the tool then generates rules, objects, and the game world (visuals, rules, and levels facets), in which their visual forms are taken from online sources. Another tool that generates the game world from narratives is *Angelina* introduced by Cook, Colton and Pease (2012). This tool is capable of orchestrating visuals and audio from articles that work as narratives (visuals, audio, and narrative facet). Moreover, *Angelina* also has an independent level generator that creates platforming stages (levels facet). The game *A Rogue Dream*, developed

by Cook and Colton (2014), receives as input the players' name as the seed of its generation process. Next, it selects the names and visuals for enemies, items, and goals and the name and mechanic of the players' special ability (narrative, visuals, and rules facets). In the end, the game generates its whole levels. *Sonancia* is a game that creates their levels (levels facet) accordingly with the tension progression in the narrative facet, which can be authored or procedurally created (LOPES; LIAPIS; YANNAKAKIS, 2016). Furthermore, the progression of levels influence sounds (audio facet). Finally, *Data Adventures* is a game that gathers Wikipedia articles and links them to generate plots (narrative facet) and searches for visuals with their titles (visuals facet) (GREEN *et al.*, 2018). Next, the game creates levels (levels facet) from the layout of the real world's cities gathered from OpenStreetMap.

Recent works also tackled the orchestration of multiple contents. Prager *et al.* (2019) developed a maze game where visual and audio style can change the environment feel, e.g., from a dark and tense to a soothing and joyful setting. They experimented with 20 players their reactions to visual and audio compositions. The players answered if they considered the compositions usual (if audio and visuals were matching) or unusual (if audio and visuals were from different settings), applying a machine learning model to predict users' reactions. As they expected, the results showed that players perceived the homogeneous settings as funnier and less difficult than the heterogeneous ones. The authors also found that players self-reported arousal when playing with homogeneous settings. The game orchestrated only two facets and experimented with a small sample of users; however, this research shows the importance of an orchestration system to create fun games.

Liapis *et al.* (2019) generated levels and characters (rules facet) for a shooter game. Then, they simulated real players through artificial agents to play the game and collect the gameplay outcomes. Next, they train a Convolutional Neural Network (CNN) with levels, class parameters of characters, and the gameplay outcomes. CNN was designed to predict the kill ratio between two players, the time of the match, and two entropy scores (all death locations and the players' positions). Their experiment showed that the most critical parameters for correct predictions were related to the rules facet; however, the data from the level facet improved the CNN's accuracy. Besides the relevant results, regarding the prediction of the outcome of the gameplay facet from two others (level and rules facets), the results also show that the data of multiple facets can be used to create more accurate models for player profiling. Moreover, machine learning models could be used to generate customized content for different players.

Similarly, Karavolos, Liapis and Yannakakis (2019) introduced a framework for generating level and ruleset components of games, which consists of a surrogate model and a search-based approach. The surrogate model is a deep learning model trained with large sets of procedurally generated levels, character classes, and simulations from the gameplay of agents. This model combines the facets of level and rules as input and gameplay outcomes as output. Then, they applied a search-based approach to generate adapted content towards a target game-

play outcome. The results show that the framework can adapt levels and rules accordingly to the designer-specified targets.

Finally, Migkotzidis and Liapis (2021) applied the previous framework to a mixed-initiative tool called *SuSketch*, a design tool for First-Person Shooter (FPS) levels. Besides providing suggestions of changes, like other co-creation applications, the tool offers designers varied types of feedback such as path information, predicted balance between players in a complete playthrough, and a predicted heatmap of the locations of player deaths. Concerning creative facets, this tool orchestrates levels, rules and gameplay facets, like Liapis *et al.* (2019) and Karavolos, Liapis and Yannakakis (2019).

Table 4 summarizes the reviewed works on PCG systems regarding the orchestration of creative facets and technique, comparing them with our work. We designed a PCG system to orchestrate level, narrative, and rules facets and adapt them to different players. In the next chapter, we describe all the modules of our PCG system.

Table 4 – Multiple content generation system literature summarizing and comparison with our work. The letters abbreviates the creative facets: **V**isuals, **A**udio, **N**arrative, **L**evel, **R**ules, and **G**ameplay. 'P' defines the partially generated creative facets.

Work	**V**	**A**	**N**	**L**	**R**	**G**	**Adaptive**	**Profiling**
Dormans (2010)	-	-	✓	✓	-	-	-	-
Lavender and Thompson (2017)	-	-	✓	✓	-	-	-	-
Linden, Lopes and Bidarra (2013)	-	-	✓	✓	-	-	-	-
Karavolos, Liapis and Yannakakis (2016)	-	-	✓	✓	-	-	-	-
Smith, Padget and Vidler (2018)	-	-	✓	✓	-	-	-	-
Pereira, Prado and Toledo (2018)	-	-	✓	✓	-	-	-	-
Pereira *et al.* (2021)	-	-	✓	✓	-	-	-	-
Gellel and Sweetser (2020)	-	-	✓	✓	-	-	-	-
Charity *et al.* (2020)	-	-	-	✓	✓	-	-	-
Ludi, Browne and Maire (2010)	-	-	-	P	✓	✓	-	-
Mechanic Miner, Cook *et al.* (2013)	-	-	-	✓	✓	P	-	-
AudioInSpace, Hoover *et al.* (2015)	P	✓	-	-	P	-	P	-
GAME FORGE, Hartsook *et al.* (2011)	-	-	✓	✓	-	-	-	-
Game-O-Matic, Treanor *et al.* (2012)	✓	-	P	P	✓	-	-	-
Angelina, Cook, Colton and Pease (2012)	✓	✓	P	✓	-	-	-	-
A Rogue Dream Cook and Colton (2014)	✓	-	✓	✓	P	-	-	-
Sonancia, Lopes, Liapis and Yannakakis (2016)	-	✓	P	✓	-	-	-	-
Data Adventures, Green *et al.* (2018)	✓	-	✓	P	-	-	-	-
Prager *et al.* (2019)	✓	✓	-	-	-	-	-	✓
Liapis *et al.* (2019)	-	-	-	✓	✓	✓	-	✓
Karavolos, Liapis and Yannakakis (2019)	-	-	-	✓	✓	✓	✓	-
Migkotzidis and Liapis (2021)	-	-	-	✓	✓	✓	✓	-
This work	-	-	✓	✓	✓	-	✓	✓

Source: Elaborated by the author.

CHAPTER

4

METHODOLOGY

In this chapter, we describe our PCG system and each one of its modules. First, in Section 4.1, we overview our PCG system by describing its modules and how they communicate with each other. Next, Section 4.2 describes the game prototype where our experiments were carried out. Then, in Section 4.3, we detail how we classify players. Finally, we describe how we generate the levels with missions and enemies as well how we orchestrate them to provide adapted content, respectively, in sections 4.4, 4.5, and 4.6.

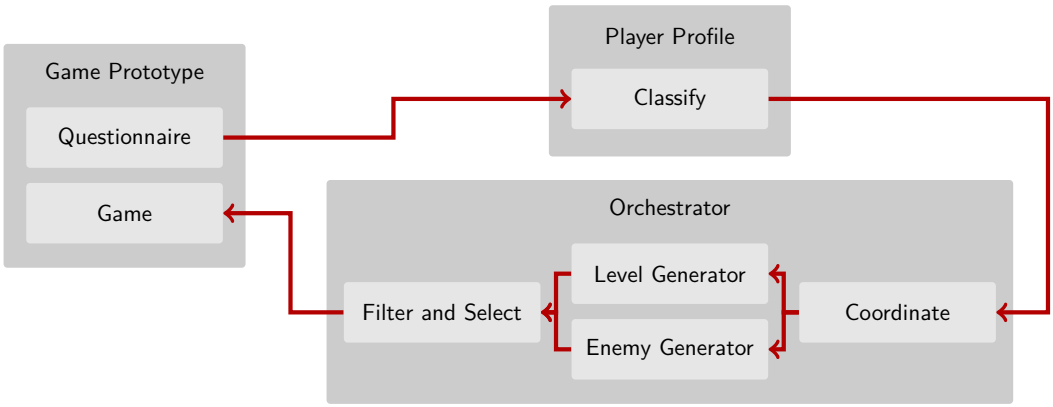
4.1 PCG system Overview

Our PCG system consists of three modules: Game Prototype, Player Profile, and Orchestrator. Figure 5 presents the PCG system diagram of how is the communication between the modules. The Game Prototype gathers the player information through in-game questionnaires and runs the game with the generated content. The game of our system is a 2D dungeon-based and Action-Adventure game inspired on *The Legend of Zelda* and *The Binding of Isaac* series (Nintendo, 1986; MCMILLEN; HIMSL, 2011). The Player Profile module receives the first questionnaire's answers (the pre-questionnaire) and classifies the player type. The player type classification is sent to the Orchestrator module that coordinates the generation of the aimed sets of levels and enemies according to the received classification – this is what makes our system adaptive. These sets of levels and enemies are generated at once by two independent PCG algorithms. Next, the orchestrator combines the contents by distributing the generated enemies through the generated levels, ignoring the unfitted contents. Finally, the generated and populated levels go to the Game Prototype to set the game. As we mentioned in Chapter 1, our PCG system is part of a major PCG system, called Overlord. The info that guides our level and enemy orchestration comes from narratives generated by the Overlord (more details in Section 4.6).

After playing all levels, the player must answer the post-questionnaire (also in the Game Prototype module) to evaluate their gameplay experience. The post-questionnaire gives

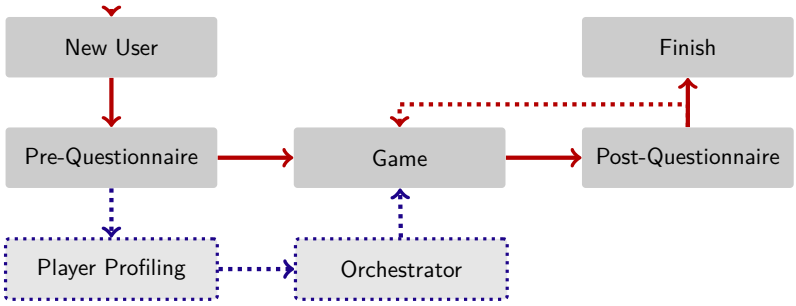
us the results indicating how players perceived the levels they played and the enemies they faced. Figure 6 presents the flowchart of a new player's interaction with the system. Therefore, Following the definition of game facets introduced by Liapis *et al.* (2019), our PCG system fits in orchestrating of levels, narrative (as lock and key missions), and rules (enemies), totaling three creative facets orchestrated concurrently. We present the methodology behind the system's modules in the following sections.

Figure 5 – PCG System Diagram. The red arrows represent the communication between modules. First, the Player Profile classifies the player type from the pre-questionnaire answers and then sends it to the Orchestrator. Based on the player type, the orchestrator coordinates the creation of levels and enemies and returns a list of levels populated by enemies to the Game Prototype.



Source: Elaborated by the author.

Figure 6 – PCG System usage flowchart. The gray rectangles and the red arrows represent the player usage flow. The player can play several levels; thus, they may loop between game and post-questionnaire (the dotted red arrow). The light gray rectangles and the dotted blue arrows represent the game prototype's background tasks: identifying the player type and sending adapted content to the game prototype.



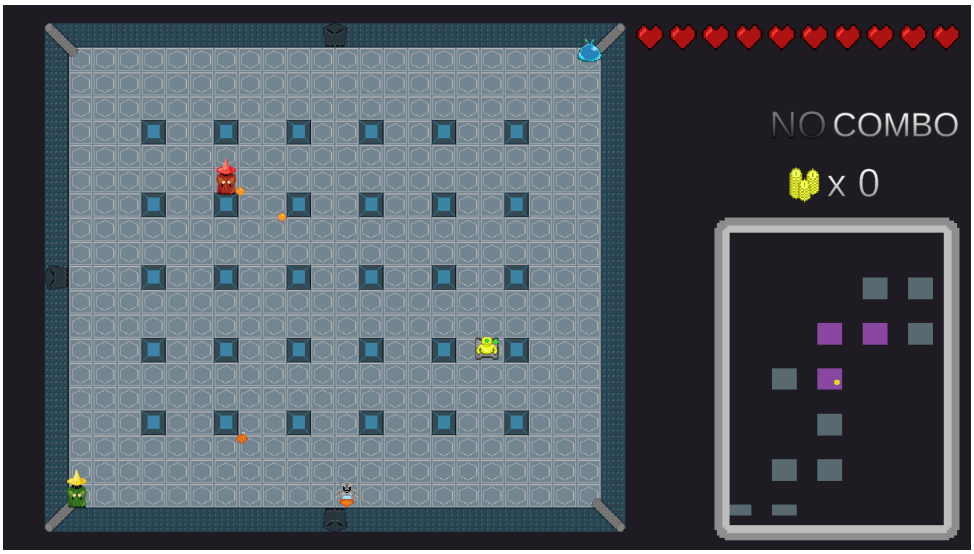
Source: Elaborated by the author.

4.2 Game Prototype

Our game prototype is an improved and adapted version of the one developed by Pereira *et al.* (2021). The game is an Action-Adventure game inspired on *The Legend of Zelda* and *The*

Binding of Isaac games (Nintendo, 1986; MCMILLEN; HIMSL, 2011). The inspired elements are the dungeon exploration from the former and the combat mechanics from the latter. In the game, the player must control a yellow robot (the protagonist of the game) to explore the levels' rooms, collect items (e.g., keys and treasures), open doors and find the levels' goals (a green triangle). Besides the locked-doors puzzle challenges, the player must defeat enemies (slimes, robots, and mages) by shooting green projectiles. Enemies fill some of the levels' rooms, but never the starting and final rooms, and the player must defeat them to proceed to other rooms. After the player wins or loses a level, a score screen is shown to give feedback about: the victory or failure, the highest combo reached, the amount of treasure collected, and the number of visited rooms. The players can only progress in the game if they win the levels. Figure 7 presents a screenshot from our game prototype. As we observe in the figure, all doors are locked, the health points are full (the hearts at the top-right), and, in the mini-map, the neighboring rooms of each room and the visited rooms are highlighted (at the bottom-right). Below we list the main game features that are present in our prototype.

Figure 7 – Screenshot from the game prototype. The player is the yellow robot. The other characters are the enemies. The orange sprites are the enemies' projectiles they shoot and bombs they throw towards the player.



Source: Elaborated by the author.

Enemies. They are obstacles to hinder the players' progression by decreasing their health points (more details in the next bullet). The enemies have difficulty degrees, which define their stats, i.e., powers and behavior. In the prototype, they are controlled by Artificial Intelligence (AI) agents and cannot respawn after being defeated. Figure 8 shows our six enemies;

Health. The players start each level with ten hearts, which corresponds to their health. There is no way to recover the hearts during the gameplay. If the players lose all their hearts in

a level, they must choose to retry the level or give up the rest of the game. Enemies also have health points, so, to defeat them, the player must attack with some weapon;

Weapons. The players must use weapons to shoot projectiles at the enemies to defeat them. They can choose a weapon before entering a dungeon. They cannot change the weapons during gameplay. There are currently four weapons: the normal gun (the initial weapon), which shoots with default speed and damage; machine gun, which shoots faster but weaker projectiles; triple-shot gun, which shoots three weaker projectiles in three directions; and cannon, which the strongest and slowest projectile;

Locked doors and keys. Some of the levels' doors are locked even after the player defeats the room's enemies (Figure 9b). These doors can be opened if the player finds the respective keys (Figure 9a). In the game, the doors' and keys' colors identify which key opens which door, i.e., a blue door must be opened by a blue key;

Treasure. The game rewards the players with treasures that are randomly distributed in the levels after every locked door and dead-end rooms without keys. This treasure distribution intends to reward explorer players. Figure 9c presents an example of treasure in the game;

Combo system. The combo system is a feature to provide engagement for players who prefer combat rather than explore. The combo system counts each shot that hits an enemy. However, if an enemy hits players, they lose the combos. Section 4.2 shows the combo counter without hits ("No Combo"). These words update with better combos as long as players keep hitting enemies and reaching their thresholds;

Map system. To help the players to orient themselves while exploring the levels, we provide a map system. This system shows a mini-map that corresponds to the whole level. The system also highlights the visited rooms and centralizes the map to allow the player to identify which rooms are neighboring.

Figure 8 – 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.

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



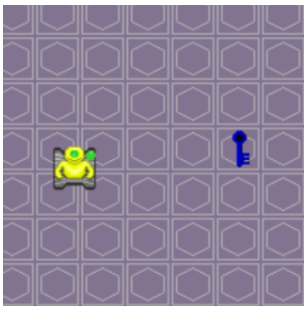
(d) Bomber mage. (e) Shieldsman. (f) Healer (Cure Spell).



Source: Elaborated by the author.

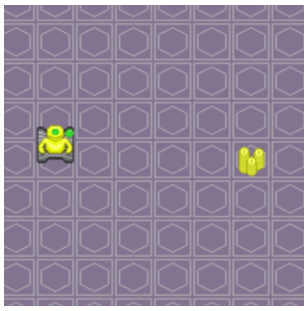
Figure 9 – The main interactive objects of the levels of our game prototype.

(a) A blue key, required to open a blue door.



(b) A blue door, which requires a blue key to be opened.

(c) A treasure, a reward for explorer players.



Source: Elaborated by the author.

Before the gameplay section, the players must answer a questionnaire about their interests in games. Then, the Player Profile module, fed with these answers, classifies the players' profiles. We detail the classification process in the following section.

4.3 Player Profiling

The Player Profile is the module responsible for classifying players. In this work, we are interested in four types of players adapted from Yee and Ducheneaut (2018). Besides, since our game does not focus on action or socialization, we excluded both classes from our PCG system. We classify players into the following types:

Achievement Those who want to complete and collect everything;

Creativity Those who enjoy exploring vast and complex dungeons;

Immersion Those who like to take part in the lore of the game game, and;

Mastery Those who prefer combat and display their skills.

Although our game prototype has no dialogue or NPCs, we kept the Immersion class for further use in the major PCG system (PEREIRA, 2021).

To carry this classification out, we calculate players through a weighted rule based on the explicitly gathered data from the pre-questionnaire shown in Table 5. This questionnaire consists of 12 questions regarding the players' preferences in games (i.e., the game features they prefer), and they answer only once when starting the game[1,2]. This questionnaire combines some questions from three different questionnaires of Heijne (2016), Vahlo *et al.* (2017), and Rivera-Villicana *et al.* (2018), all of them on a 5-point Likert scale. We did not choose all questions

[1] The game was in Portuguese, so we translated such questions from English into Portuguese.
[2] We did not collect demographic data in this questionnaire.

to provide a short questionnaire to minimize bias in mind (JAMESON, 2008). Thus, our pre-questionnaire is classified as a *self-assessment with respect to general dimensions* according to the definition proposed by Jameson (2008). As advised by Jameson (2008), we also allow the players, if they wish, to ignore all the questionnaires and only play the game. Moreover, we let clear to the players that we are not evaluating their skills but the system.

Table 5 – Player preferences pre-questionnaire on a 5-point Likert scale, designed to understand each player's profile and experience. The player types are abbreviated as follows: Mastery (M), Achievement (A), Creativity (C), and Immersion (I). '*' means that the 5-point Likert answer is converted to [-2, 2] interval.

Question	**Profile Affected**	**Weight**
1 - I am an experienced player.	-	0
2 - I am an experienced player in the Action-Adventure genre.	-	0
3 - In which difficulty do you usually play? (There are many naming standards. Try to select the one closest to what you are used to).	M	[-2, 2]*
4 - I like playing games where I can explode, crush, destroy, shoot and kill.	M	[1, 5]
5 - I like playing games where I can fight using close combat skills and evade fast attacks.	M	[1, 5]
6 - I like playing games where I can explore the game world and uncover secrets and mysteries.	C	[1, 5]
7 - I explore all the places, elements and characters of the virtual world.	C	
8 - I complete all quests, including those that aren't necessary to finish the game.	A	[1, 5]
9 - I like playing games where I can collect rare items and hidden treasures.	A	[1, 5]
10 - I like playing games where I can build friendships between game characters and work toward a common goal.	I	[1, 5]
11 - I like playing games where I can immerse myself in the role of the character and make meaningful decisions.	I	[1, 5]
12 - I usually only do what is necessary to pass a level or complete a quest.	ACI	[-2, 2]*

Source: Elaborated by the author.

Table 5 shows the questions and how their weights influence the player profiling. After answering the pre-questionnaire, Equations 4.1, 4.2, 4.3, and 4.4 calculate the player compatibility for each category. Next, we rank them in descending order, where the higher value corresponds to the most compatible class and the lesser value to the less compatible one. We set weights to each profile regarding their preference order: the most preferred profile weights 7, and the remaining weights 5, 3, and 1, respectively. Such values fed the narrative generator of Overlord, which defines the parameters for generating levels and enemies. Thus, the narrative defines how much content appears in the game, respecting the players' preference. We present more details about it in Section 4.6.

$$C_{Achievement} = Q_8 + Q_9 + Q_{12} - 3 \tag{4.1}$$

$$C_{Creativity} = Q_6 + Q_7 + Q_{12} - 3 \tag{4.2}$$

$$C_{Immersion} = Q_{10} + Q_{11} + Q_{12} - 3 \tag{4.3}$$

$$C_{Mastery} = Q_3 - 3 + Q_4 + Q_5 \tag{4.4}$$

Before the gameplay section, the player type is sent to the Orchestrator module. This module generates sets of levels and enemies, selecting those with the most relevant content based on the player's profile. Next, we describe the processes of level and enemy generation algorithms, as well as their orchestration.

4.4 Level Generator

In this section, we present an approach that generates dungeon levels with locked-door missions by extending the EA introduced by Pereira *et al.* (2021). The said work presented an EA capable of generating levels with locked-door missions that match the game designer parameters for levels and missions. Our extended version offers two main contributions from Pereira *et al.* (2021) and procedure dungeons generation (PDG) related works. The first one is to advance from the previous EA by also evolving the enemies distribution through the levels' rooms. The second contribution is the application of a MAP-Elites algorithm for enhancing Quality Diversity in content generation, taking into account the level design with lock, keys, and enemies placement. Besides, this algorithm itself fits in orchestrating of levels and narratives, as lock and key missions (LIAPIS *et al.*, 2019). Our approach was submitted to Foundations of Digital Games (FDG) 2022 (VIANA *et al.*, 2022a).

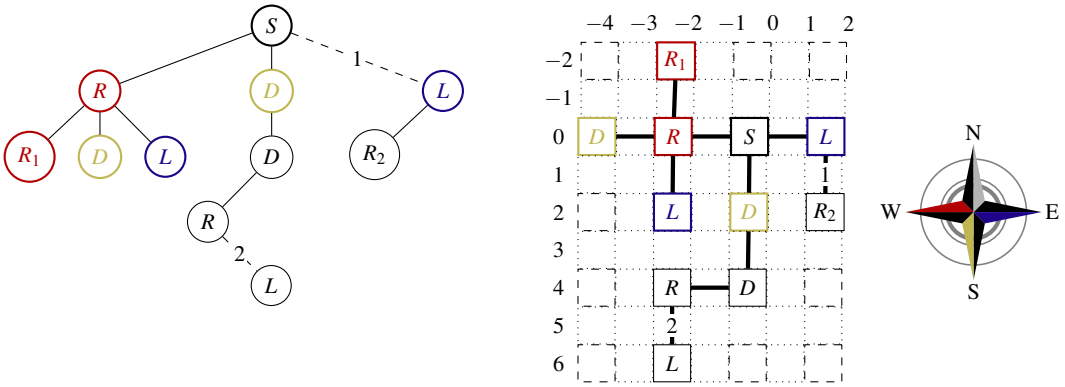
4.4.1 Level Representation

One individual in our evolutionary algorithm states a dungeon level, where a tree structure represents such an individual, as shown in Figure 10a. Each node defines a room that encodes its type and position in the dungeon. We have a Key Room (*KR*), which indicates an available key to open a locked door, a Locked Room (*LR*) with a locked door, and a Normal Room (*NR*) that has nothing special. The node also encodes the number of enemies in that room and the room position concerning the parent node: Right (*R*), Down (*D*), and Left (*L*). We place the room assuming that its parent room (node) is in the north, positioning it (*R*, *D*, and *L*) correctly when decoding the individual into a dungeon level.

To ensure no room overlaps the level, we decode the tree representation (genotype) to a 2D grid (phenotype). If there are overlapping rooms, the branch that causes the overlap is removed from the tree. Thus, we ensure that no level is infeasible following the process of branch removal detailed in Pereira *et al.* (2021). A single key can open a locked door; therefore, the keys are bound with their locks through a shared ID. Rooms can have only a key, only a locker, or none of them; they can never have only one of each or multiples of a kind at the same time. Moreover, our representation does not require keys to be collected and unlocked in a specific sequence; they are placed without further control.

Figure 10 – Handcrafted example of a genotype-phenotype level translation. (a) presents the level genotype and (b) presents the resulting phenotype. The root node *S* represents the starting room. Nodes with *R* (Right), *D* (Down), and *L* (Left) represent the direction the parent node connects with them. The numbers in the nodes are keys. The numbers in the dashed edges are locks. Rooms are always placed in even values of the x and y coordinates, while corridors are placed in coordinates with different parities. By comparing the node colors with the wind rose, we see that a parent room is considered in the north direction regarding any of its child rooms.

(a) Dungeon level genotype. (b) Dungeon level phenotype.



Source: Elaborated by the author.

4.4.2 Level Generation Process

Our dungeon generation process evolves tree structures of feasible levels. The parameters that our algorithm receives are the number of rooms, number of keys, number of locks, number of enemies, and linear coefficient (linearity). We designed our approach to evolve dungeons by preserving diversity and optimizing quality. To do so, we applied a MAP-Elites approach for variety by mapping the feature descriptors (or dimensions) of the leniency of enemies and exploration coefficient. To measure the leniency of enemies in our levels, we apply the Equation 4.5 presented by Smith, Padget and Vidler (2018). The leniency is calculated by the number of safe rooms, i.e., without enemies, divided by the total number of rooms.

$$D_{\text{leniency}} = \frac{Number\ of\ Safe\ Rooms}{Total\ Rooms} \tag{4.5}$$

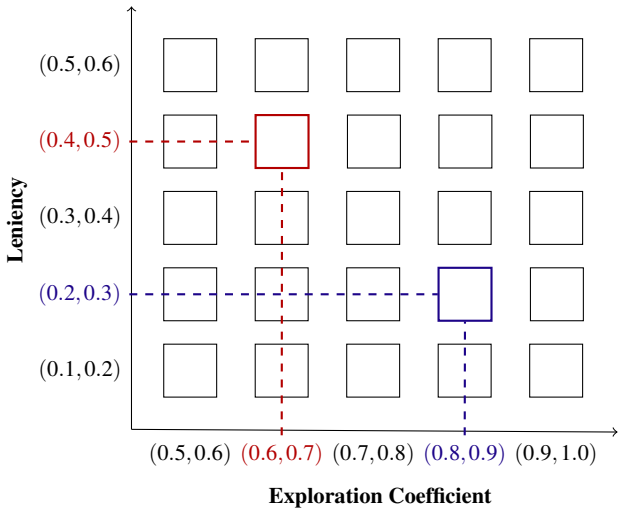
Equation 4.6 measures our exploration coefficient, inspired by the exploration measure introduced by Liapis, Yannakakis and Togelius (2013). We run a flood fill algorithm between rooms to simulate the map coverage, where the reached rooms represent the required exploration from each starting room and its corresponding goal room.

$$D_{\text{exploration}} = \frac{1}{\#RR} \sum_{(r_s, r_g) \in RR} \frac{Coverage(r_s, r_g)}{Total\ Rooms} \tag{4.6}$$

where RR is the set of pairs of reference rooms containing the pair of starting and goal rooms and all pairs of key and locked rooms; $\#RR$ is the size of RR; r_s is the room where the flood fill starts, and; r_g is the goal room where the algorithm ends.

Since our equations result in values between 0 and 1, we discretized such dimensions. For the leniency dimension, the intervals are (0.5, 0.6), (0.4, 0.5), (0.3, 0.4), (0.2, 0.3), and (0.2, 0.1). Levels with greater leniency values have most rooms without enemies or some of them with several enemies. For exploration coefficient, the intervals are (0.5, 0.6), (0.6, 0.7), (0.7, 0.8), (0.8, 0.9), and (1.0, 0.9). Levels with exploration coefficient values lesser than these lead to rooms much closer to each other. Figure 11 presents our approach's map.

Figure 11 – The map of MAP-Elites population. The red cell represents a dungeon with leniency between 0.4 and 0.5 and an exploration coefficient between 0.6 and 0.7. The blue cell represents a dungeon with leniency between 0.2 and 0.3 and an exploration coefficient between 0.8 and 0.9. Thus, the blue level has more reference rooms further to each other than the red one, and it also has more rooms with enemies.



Source: Elaborated by the author.

Our MAP-Elites for levels maps 25 individuals based on the defined intervals. When the map receives a new individual, we must calculate the feature descriptors to place it in the correct entry of the MAP-Elites table. If an individual fills a map cell and a new one hits the same cell, the latter replaces the former if it has a better fitness; otherwise, we discard the new individual.

The evolutionary process starts generating individuals for the initial population by following the initialization algorithm described in their work to create rooms, keys, and lockers. However, we introduced two changes in the initialization procedure. First, we are dealing with the placement of enemies in our approach; therefore, after generating each dungeon, we place enemies in random rooms, one by one, except by the starting and goal rooms. Besides, if the created level has no locker, we add one with a key to ensure that we can calculate the goal room. If this level also has keys, we first remove one of them. Second, we add individuals to the initial population until it reaches n individuals (obviously, 25 is the max value). Since the initialized individuals may hit the same entry in the MAP-Elites table, it can take a while. In this case, the population size does not change once the best individual is always kept for that entry.

Next, we evolve the population using the time-limit stopping criterion. Pereira *et al.*

(2021) create an intermediate population that always replaces the current one, except by the best individual found so far. In our case, after stating the intermediate population, we try to insert its individuals in the MAP-Elites population. Our intermediate population has new individuals created from two parents, which are chosen using tournament selection with two competitors.

The crossover randomly selects one parent node as the cut point to swap the selected nodes. After the swap, we remove all overlaps and rebuild the grid. We try to preserve the locks and keys of the original branches in the new individual by applying the repair algorithm described in Pereira *et al.* (2021). We always apply crossover, while mutation has a chance of 15% to be applied, where a pair of a lock and key has 50% chance to be added or removed from the tree structure. We visit the tree structure through a breadth-first order to add a pair and convert a random *NR* into a *KR*. Next, we do the same to convert a random *NR* into a *LR* among the non-visited rooms. To remove a pair, we randomly select a *KR*, and its related *LR*, converting both into *NR* nodes. After this, we perform an enemy transfer operation. To do so, we select two rooms to transfer and to receive them. If both rooms have no enemies, nothing is done. If the receiver has enemies and the transferer does not have them, we swap the rooms. Then, we randomly chose from 1 to the transferer's number of enemies to move to the receiver room.

After the crossover and mutation operators, we repair the new individuals regarding the distribution of enemies. The crossover may generate levels that have more or fewer than the associated input parameter. On the other hand, the mutation may transfer enemies to rooms that cannot have them, i.e., the goal room. If there are enemies in this room, we remove them. When the number of enemies is higher, we remove them, prioritizing the rooms with more of them. Otherwise, when the number of enemies is lesser, we add them, prioritizing the non-empty rooms with fewer enemies.

Finally, the new individuals in the intermediate population are evaluated using an extended version of the fitness function in Pereira *et al.* (2021). Our function calculates three fitness factors. First, we measure the distance of the input parameters and the generated level:

$$\begin{aligned} f_{\text{goal}} = {} & abs(G_{rooms} - L_{rooms}) + \\ & abs(G_{keys} - L_{keys}) + \\ & abs(G_{locks} - L_{locks}) + \\ & abs(G_{linear\_coefficient} - L_{linear\_coefficient}) + \\ & L_{rooms} - L_{needed\_rooms} + \\ & L_{locks} - L_{needed\_locks} \end{aligned} \tag{4.7}$$

where G is the set of goals and L is the set of the level's attributes (number of rooms, number of keys, number of locks, and linear coefficient); $needed\_rooms$ is calculated by an adaptation of a Depth-First Search algorithm, and; $needed\_locks$ calculated by an adaptation of an A* algorithm, both algorithms are described in Pereira *et al.* (2021).

The second factor is an extension of the enemy sparsity equation introduced by Sum-

merville *et al.* (2017) to evaluate the distribution of enemies in the 2D maps:

$$f_{\text{es}} = \frac{\sum_{e \in E}(e_x - \mu_x)^2 + (e_y - \mu_y)^2}{Number\ of\ Enemies} \tag{4.8}$$

where e_x and e_y are the x-position and y-position of an enemy e, μ_x and μ_y are the average x-position and y-position of all enemies, and E is the set of enemies. In the third term, we calculate the standard deviation of enemies in the rooms:

$$f_{\text{std}} = \sqrt{\frac{1}{N-2}\sum_{r \in R}(r_{enemies} - \mu_{enemies})^2} \tag{4.9}$$

where r is a room in the set of rooms R, $r_{enemies}$ is the number of enemies of a room, $\mu_{enemies}$ is the average number of enemies in the rooms, and N is the number of rooms. We subtract the starting and the goal rooms since they cannot have enemies. The final fitness expression follows:

$$L_{fitness} = f_{goal} - f_{es} + f_{std} \tag{4.10}$$

We subtract the enemy sparsity f_{es} because higher values are better for such metric, and we aim to minimize f_{goal} and f_{std} as well as our fitness function as a whole.

4.5 Enemy Generator

In this section, we present our enemy generation algorithm, which is an extension of the parallel EA introduced by Pereira, Viana and Toledo (2021). This work evolved enemies represented by common features of Action-Adventure games. Our approach advances this EA by applying MAP-Elites population to illuminate enemies' space, and we did not evolve the enemies through parallel evolution. Besides, we submitted this algorithm to Conference on Games (CoG) 2022 (VIANA *et al.*, 2022b).

4.5.1 Enemy Representation

As mentioned, our enemy genotype comes 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, weapon type, and projectile speed. Table 6 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 6, 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:

Table 6 – List of attributes of the enemy's genotype. The line between attributes represents the crossover point.

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 projectiles are shot (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.

Source: Adapted from Pereira, Viana and Toledo (2021).

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 ax 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 ax 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.

4.5.2 Enemy 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:

$$D_{\text{movement}} = e_{movement\_type} \tag{4.11}$$

$$D_{weapon} = e_{weapon\_type} \tag{4.12}$$

where, e is the enemy. Figure 12 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.

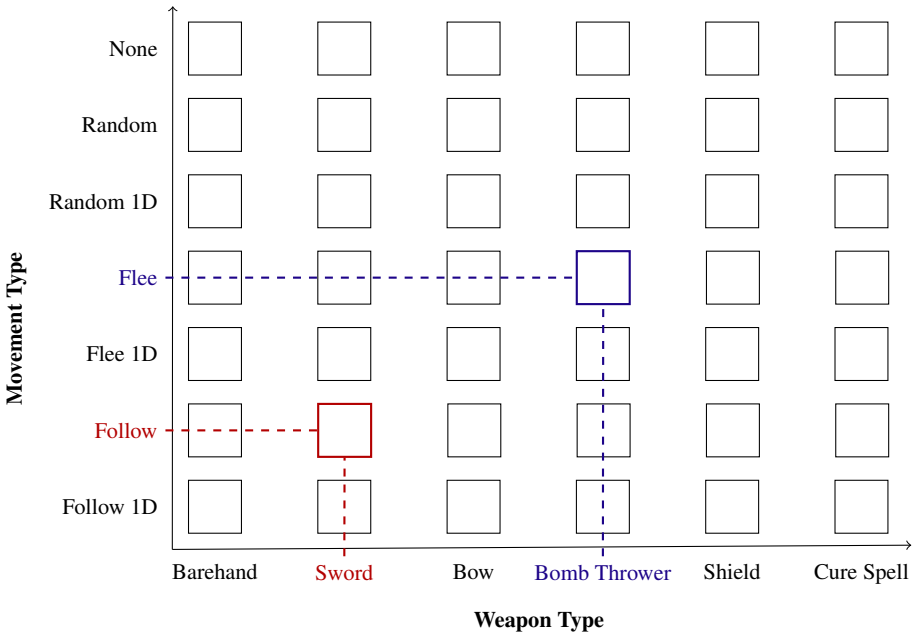
The proposed MAP-Elites approach maps 42 enemies in terms of the defined dimensions. When the population receives a new individual, we calculate its feature descriptors to place it in the correct map entry. If a new enemy hits a filled cell, we apply elitism, i.e., the best enemy fills the cell, discarding 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. Pereira, Viana and Toledo (2021) 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

Figure 12 – 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.



Source: Elaborated by the author.

Blended Crossover α (BLX-α) (ESHELMAN; SCHAFFER, 1993). We first cross the parents in the fixed point as shown in Table 6. 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 (ESHELMAN; SCHAFFER, 1993).

After the crossover, we have a chance to mutate both resulting enemies and, when a mutation happens, we apply a multi-gene mutation (KANAGAL-SHAMANNA *et al.*, 2014). 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.

$$d_{health} = 2 \times e_{health} \tag{4.13}$$

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

$$d_{movement} = e_{mv\_spd} + e_{act\_tm}/3 + 1/e_{rst\_tm} \tag{4.14}$$

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.

$$d_{strength} = d_{s_1} \times d_{s_2} \times d_{s_3} \tag{4.15}$$

For melee enemies, we multiply damage by movement speed.

$$d_{s_1} = \begin{cases} e_{dmg} \times e_{mv\_spd}, & \text{IsMelee}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.16}$$

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

$$d_{s_2} = \begin{cases} 3 \times (e_{atk\_spd} \times e_{prjct\_spd}), & \text{IsRanged}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.17}$$

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.

$$d_{s_3} = \begin{cases} 2 \times e_{atk\_spd}, & \text{IsHealer}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.18}$$

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.

$$d_{gameplay} = d_{g_1} \times d_{g_2} \times d_{g_3} \times d_{g_4} \times d_{g_5} \tag{4.19}$$

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.

$$d_{g_1} = \begin{cases} 1.25, & \text{IsMelee}(e) \text{ and IsFollow}(e) \\ 0.5, & \text{IsMelee}(e) \text{ and} \\ & (\text{IsAnyFlee}(e) \text{ or HasNoMove}(e)) \\ 1, & \text{otherwise} \end{cases} \tag{4.20}$$

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.

$$d_{g_2} = \begin{cases} 1.25, & \text{isRANGED}(e) \text{ and } \text{isFLEE}(e) \\ 1.15, & \text{isRANGED}(e) \text{ and } \text{isFLEE1D}(e) \\ 0.5, & \text{isRANGED}(e) \text{ and } \text{hasNoMove}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.21}$$

$$d_{g_3} = \begin{cases} 0.5/(2 \times e_{mv\_spd}), & \text{isRANGED}(e) \text{ and} \\ & \text{isFOLLOW}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.22}$$

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.

$$d_{g_4} = \begin{cases} 1, & \text{isHEALER}(e) \text{ and} \\ & (\text{isANYRANDOM}(e) \text{ or} \\ & \text{isANYFLEE}(e)) \\ 0.5, & \text{otherwise} \end{cases} \tag{4.23}$$

$$d_{g_5} = \begin{cases} 1.15 \times e_{mv\_spd}, & \text{isHEALER}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.24}$$

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

$$d = d_{gameplay} \times (d_{health} + d_{movement} + d_{strength}) \tag{4.25}$$

4.6 Orchestrating Adaptive Content

Our orchestrator filters and selects the generated levels and enemies to send to the Game Prototype. Thus, our PCG system is hybrid regarding generation time. We coordinate the generation of dungeon levels and enemies offline before the experiments with players. In the online step, we classified players and selected the appropriate content previously created for their profiles. In this section, we describe how our orchestrator coordinates the described generators, and how it filters and selects the generated contents for players.

As we mentioned earlier, our PCG system is part of a major PCG system (PEREIRA, 2021). Such a system generates narratives based on player types that define the levels and enemy types within these levels. This definition is made by setting values for the parameters of both our evolutionary generators to create levels and enemies. Such information guides our orchestrator to coordinate the level and enemy generators and perform the post-processing.

With the contents ready, the orchestrator starts filtering and selecting levels. Our level filter discards levels with fitness values higher than 2. These levels did not reach the aimed features of our level generation approach since they, for instance, did not generate the intended number of keys. The level selection process is simple; we draw a level with the appropriate features for players according to their types.

Next, we filter the enemies by discarding all the bad ones from the pool of enemies. Bad enemies are those that are melee and cannot move or have a sword as a weapon and flee from the player. We designed such a filter after playing and observing the bad behaviors of the enemies. Finally, we populate the enemies in the chosen level's rooms accordingly to the following rules:

1. If the room has only an enemy, we select an enemy randomly;
2. If the room has at least two enemies, we try to place different types of enemies;
3. Repeat until there are no more enemies to place.

Since the narrative defines the number of enemies by type, we must be careful when placing enemies. We control melee, ranged, and healer enemies with lists and transfer them one by one to the rooms as shown in pseudo-code Algorithm 2. We only place a healer in a room that must have a single enemy if no more melees or rangers remain. We place a healer in a room if the room must have at least two enemies in it. If the room already has a healer, we only add another healer if there is no other type of enemy.

Algorithm 2 – Enemy Selection Process.

```
1:  procedure EnemySelector(room: Room, enemies: Enemies)
2:      melees ← enemies.GetMelees();
3:      rangers ← enemies.GetRangers();
4:      healers ← enemies.GetHealers();
5:      amount ← room.total_enemies;
6:      healer ← False;
7:      while amount > 0 do
8:          hasMelees ← melees.Size() > 0;
9:          hasRangers ← rangers.Size() > 0;
10:         hasHealers ← healers.Size() > 0;
11:         if amount == 1 then
12:             if hasHealers and not hasMelees and not hasRangers then
13:                 room.AddRandomEnemy(healers)
14:             else
15:                 room.AddRandomEnemy(melees, rangers)
16:             end if
17:             amount ← amount − 1
18:         else if amount >= 2 then
19:             if (not healer and hasHealers) or (not hasMelees and not hasRangers) then
20:                 room.AddRandomEnemy(healers);
21:                 amount ← amount − 1
22:                 healer ← True;
23:             else if (melees, rangers).Size() > 1 then
24:                 room.AddRandomEnemy(melees, rangers)
25:                 room.AddRandomEnemy(melees, rangers)
26:                 amount ← amount − 2
27:             else
28:                 room.AddRandomEnemy(melees, rangers)
29:                 amount ← amount − 1
30:             end if
31:         end if
32:     end while
33: end procedure
```

CHAPTER

5

RESULTS

In this chapter, we present the results of our PCG system. We present the computational results of our level generation algorithm and our enemy generation algorithm, respectively, in sections 5.1 and 5.2. Then, in Section 5.3, we present the results regarding the gameplay feedback from players regarding levels, enemies, and the combination of both.

5.1 Level Generation

This section reports the computational results achieved by our level generation approach and some of the levels generated by it. We defined the evolutionary parameters empirically after comparing some range of values. The results comparing different configurations are available in a Google Sheets spreadsheet[1]. After such evaluation, we set the following method's parameters: 20 individuals for initial population, 15% for mutation rate, 100 individuals for intermediate population, 2-size for tournament selection, and 60 seconds as stop-criterion.

Next, we collected data from 30 executions of our method for six different sets of parameters to evaluate the algorithm performance. Table 7 shows the average and standard deviation of the fitness for each Elite (entry) of our MAP-Elites population. We observe that the fitness values tend to decrease as leniency decreases, which is expected because there are more safe rooms in L1 levels (50% to 60%) than L2 levels (40% to 50%), less in L3 levels, and so on. Moreover, L2 naturally presents their enemies distributed in more rooms than L1 levels; thus, increasing the enemy sparsity and decreasing the standard deviation of enemies. By comparing tables 7b and 7c, we observe that increasing the linear coefficient decreases the dungeons' fitness. That means that our algorithm works slightly better for lower linear coefficients.

E5 column in Table 7d presents only subpar fitness values, and these results happen mainly due to the high number of locks at such a small level. To be mapped in E5, the map

[1] Link to the spreadsheet: https://docs.google.com/spreadsheets/d/1Rx79rBWl3gHE0KSU7AuZ8VKVwB047yrKrZ0041SLIE8.

Table 7 – Results of fitness obtained after 30 executions of our level generation approach. Each table caption represents a set of parameters: (number of rooms)-(number of keys)-(number of locks)-(number of enemies)-(linear coefficient). Each table cell corresponds to an Elite. Descriptors of leniency are the rows. Descriptors of exploration coefficient are the columns.

(a) 15-3-2-20-2.

	E1 (0.5,0.6)	**E2 (0.6,0.7)**	**E3 (0.7,0.8)**	**E4 (0.8,0.9)**	**E5 (0.9,1.0)**
L1 (0.5,0.6)	0.53±0.39	0.45±0.35	0.50±0.40	0.45±0.36	0.59±0.41
L2 (0.4,0.5)	-0.04±0.37	-0.12±0.38	-0.12±0.36	-0.12±0.36	0.01±0.40
L3 (0.3,0.4)	-0.29±0.37	-0.37±0.38	-0.36±0.36	-0.36±0.36	-0.23±0.41
L4 (0.2,0.3)	-0.53±0.36	-0.61±0.37	-0.59±0.36	-0.59±0.36	-0.45±0.42
L5 (0.1,0.2)	-0.50±0.54	-0.63±0.48	-0.61±0.46	-0.52±0.60	-0.25±0.89

(b) 20-4-4-30-1.

	E1 (0.5,0.6)	**E2 (0.6,0.7)**	**E3 (0.7,0.8)**	**E4 (0.8,0.9)**	**E5 (0.9,1.0)**
L1 (0.5,0.6)	0.03±0.41	-0.05±0.46	-0.09±0.41	-0.08±0.37	0.31±0.99
L2 (0.4,0.5)	-0.59±0.41	-0.63±0.40	-0.63±0.37	-0.61±0.37	-0.32±0.80
L3 (0.3,0.4)	-0.95±0.42	-0.99±0.40	-0.98±0.38	-0.97±0.37	-0.70±0.76
L4 (0.2,0.3)	-1.18±0.40	-1.24±0.40	-1.21±0.39	-1.20±0.38	-0.92±0.79
L5 (0.1,0.2)	-1.23±0.38	-1.32±0.40	-1.31±0.39	-1.31±0.39	-0.74±1.24

(c) 20-4-4-30-2.

	E1 (0.5,0.6)	**E2 (0.6,0.7)**	**E3 (0.7,0.8)**	**E4 (0.8,0.9)**	**E5 (0.9,1.0)**
L1 (0.5,0.6)	0.24±0.36	0.20±0.34	0.16±0.26	0.29±0.33	0.76±0.85
L2 (0.4,0.5)	-0.33±0.27	-0.34±0.28	-0.36±0.23	-0.27±0.27	0.00±0.60
L3 (0.3,0.4)	-0.69±0.26	-0.69±0.26	-0.71±0.23	-0.65±0.25	-0.38±0.58
L4 (0.2,0.3)	-0.93±0.25	-0.93±0.25	-0.94±0.22	-0.90±0.22	-0.63±0.55
L5 (0.1,0.2)	-0.92±0.42	-0.92±0.43	-0.94±0.40	-0.86±0.45	-0.29±1.01

(d) 25-8-8-30-2.

	E1 (0.5,0.6)	**E2 (0.6,0.7)**	**E3 (0.7,0.8)**	**E4 (0.8,0.9)**	**E5 (0.9,1.0)**
L1 (0.5,0.6)	-0.14±0.55	-0.32±0.42	-0.36±0.34	-0.03±0.93	8.97±5.89
L2 (0.4,0.5)	-0.92±0.28	-1.02±0.31	-1.00±0.30	-0.73±0.79	6.46±4.88
L3 (0.3,0.4)	-1.23±0.31	-1.29±0.32	-1.27±0.31	-1.02±0.80	5.94±4.72
L4 (0.2,0.3)	-1.57±0.32	-1.64±0.33	-1.63±0.32	-1.38±0.82	5.33±4.53
L5 (0.1,0.2)	-1.55±0.70	-1.71±0.45	-1.73±0.34	-1.49±0.83	5.90±4.74

(e) 30-4-4-50-2.

	E1 (0.5,0.6)	**E2 (0.6,0.7)**	**E3 (0.7,0.8)**	**E4 (0.8,0.9)**	**E5 (0.9,1.0)**
L1 (0.5,0.6)	1.33±1.43	0.55±0.67	0.69±0.68	1.20±1.41	4.22±2.70
L2 (0.4,0.5)	-0.16±0.31	-0.17±0.49	-0.21±0.26	-0.18±0.24	0.54±1.38
L3 (0.3,0.4)	-0.68±0.25	-0.70±0.37	-0.75±0.21	-0.69±0.21	-0.34±0.86
L4 (0.2,0.3)	-1.08±0.21	-1.07±0.32	-1.11±0.19	-1.06±0.20	-0.78±0.58
L5 (0.1,0.2)	-1.21±0.25	-1.19±0.34	-1.25±0.22	-1.19±0.22	-0.80±0.69

(f) 30-6-6-50-1.5.

	E1 (0.5,0.6)	**E2 (0.6,0.7)**	**E3 (0.7,0.8)**	**E4 (0.8,0.9)**	**E5 (0.9,1.0)**
L1 (0.5,0.6)	0.23±1.51	-0.37±0.76	-0.17±0.90	0.61±1.58	7.96±17.62
L2 (0.4,0.5)	-0.92±0.63	-1.13±0.19	-1.11±0.20	-1.08±0.15	0.02±1.58
L3 (0.3,0.4)	-1.53±0.21	-1.60±0.17	-1.58±0.18	-1.55±0.18	-1.12±0.75
L4 (0.2,0.3)	-1.90±0.17	-1.96±0.16	-1.94±0.18	-1.91±0.16	-1.63±0.67
L5 (0.1,0.2)	-2.06±0.17	-2.12±0.16	-2.09±0.18	-2.08±0.18	-1.79±0.69

Source: Elaborated by the author.

coverage must fill 90% to 100% of the level's rooms. Nevertheless, the keys are usually closer to their locks; thus, they cannot be mapped. We believe this result is caused mainly due to the

crossover operation, which must ensure that both lock and key must be in the swapped branch. Once lock and key are in the same branch, the coverage cannot fill 90% of the level's rooms. Thus, the levels found in the E5 column in Table 7d have fewer locks than required by the input.

Finally, the Elite L1-E5 presents the worst fitness value in all the tables. In this case, our algorithm should fill enemies in levels with 50% up to 60% safe rooms and ensure they present 90% up to 100% of exploration coefficient. Our algorithm struggled to find good results for such Elites. Besides, this Elite has poor fitness values, particularly in the tables 7e and 7f. Such a result is an accumulation of bad values of the factors of f_{goal}, in which the main one is the number of rooms. Since our levels are randomly generated in the initial population, the difficulty of generating levels with a higher number of rooms is somewhat expected.

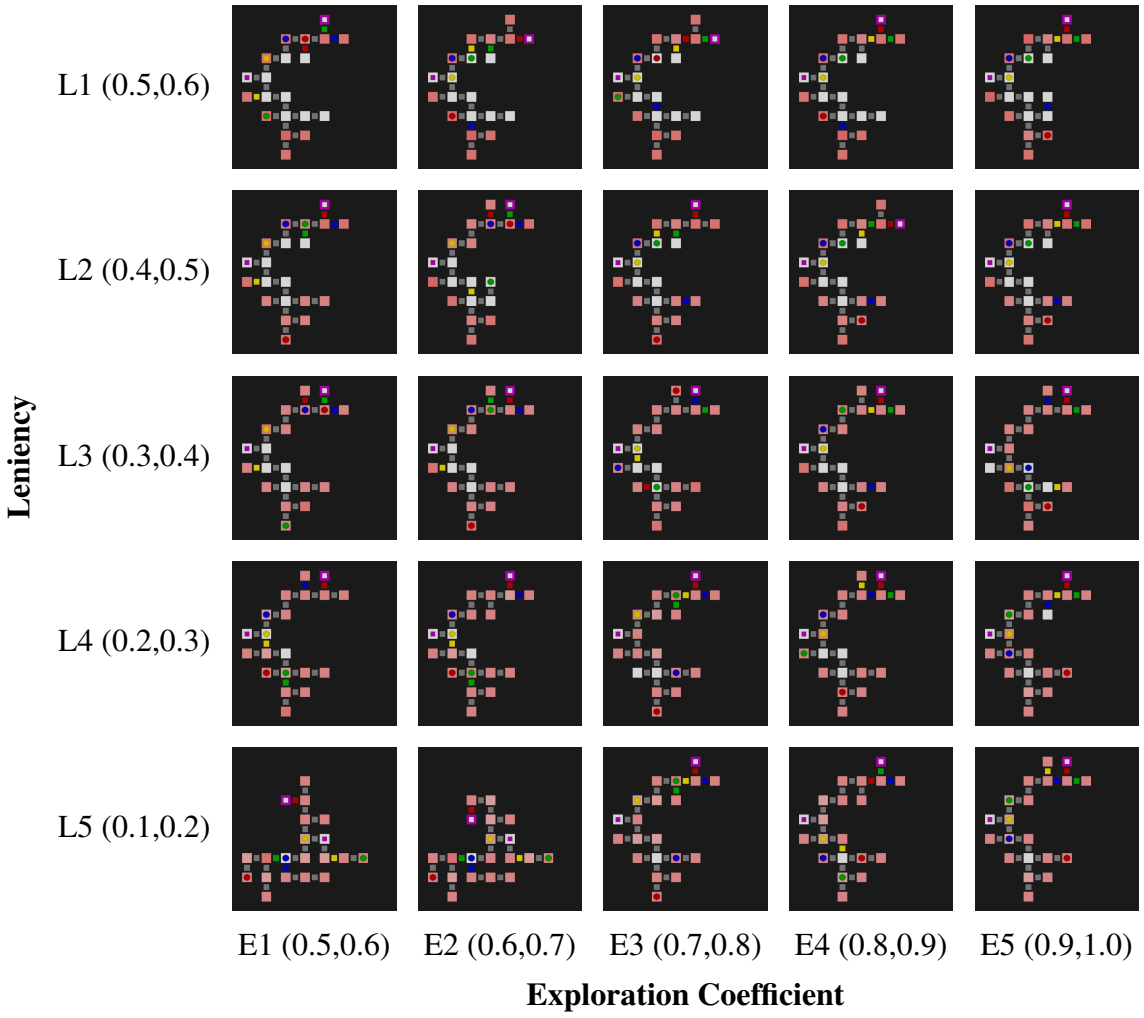
Figure 13 shows the result of an execution of our method. The figure shows the mapping performed by our algorithm, where we can see that the lower the leniency in the levels, the more the number of rooms with enemies (represented by squares with shades of red). The empty rooms (white squares) tend to be closer to each other, while rooms with enemies tend to be closer to the edges of the levels. We believe this behavior occurs due to the enemy sparsity since it encourages the distribution of enemies regarding their position on the map. Besides, there are stronger shades of red in the rooms with lesser leniency values, which is an expected result since we maintain the number of enemies in the whole level. The levels with high leniency degrees present more rooms with more enemies.

Regarding exploration coefficient, levels with lesser values for such metric present some keys closer to their locks; some are just in front of the lock they should open. Hence, as expected, the levels with higher exploration coefficients present a higher distance between the keys and their locks. For instance, the closer distance between a key and its lock in E5 levels is four rooms (e.g., in L1-E5 level). Nonetheless, there is at least a key in all the levels, far from its lock. In E1 levels, this key is the one that opens the goal room.

Moreover, considering only the positions of the rooms, the structure of most levels is very similar. Some similar levels differ only in terms of the enemies' position, such as L4-E3 and L5-E3. Locks and keys may also appear in the same positions (rooms), but they may change the required gameplay significantly. For instance, in level L2-E2, the player must collect the yellow key and open the yellow lock to collect the green key and open the goal room. In L2-E2, however, the player can access all the keys without unlocking any door. Nonetheless, this particular feature of chained locked-door missions is rare to appear in our approach; in Figure 13 there are 7 out of 25 levels with this feature.

Although some of the levels are similar in terms of room placement, we can observe that the algorithm worked as intended: a level with a set of rooms different in both exploration and leniency was created, with most of them converging very close to the designer's needs and having interesting contents.

Figure 13 – Example of a MAP-Elites population of levels with 20 rooms, 4 keys, 4 locks, 30 enemies, and linear coefficient equal to 2. Each table cell corresponds to an Elite. The small squares represent corridors, and the bigger squares represent rooms. The white room with a purple square within it is the start room. The purple room with a white square within it is the goal room. White rooms have no enemies while red rooms have enemies within, the more intense the shade of red, the more enemies there are. Colored corridors are locked, and their keys are colored circles within rooms.



Source: Elaborated by the author.

5.2 Enemy Generation

This section reports the computational results achieved by our enemy generation approach. 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[2]. After this comparison, we defined 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.

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

Table 8 – Results of fitness obtained after 100 executions of our enemy generation approach.

(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

Source: Elaborated by the author.

Next, we collected data from 100 executions of our method for three different difficulty goals to evaluate its performance. Table 8 shows the average and standard deviation of the fitness for each Elite (entry) of our MAP-Elites population.

In Table 8, we observe that most solutions converged to zero, which is the best value since we are using a distance fitness. This result is expected because lower difficulty values are somewhat easy to reach. However, in Table 8, Bow and Bomb Thrower enemies with movements of Flee and Follow presented higher values. For ranged enemies with Flee movement, is expected since this movement makes this ranged enemies more dangerous. This Elite gets closer to zero in the remaining tables, which means 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 analogous in the remaining test cases.

In tables 8d and 8e, 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. We expected such results since 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 in 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 approach execution time, Table 9 presents the average, minimum, maximum, and standard deviation values for duration time, in seconds, of the 100 executions. The experiments 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.

Table 9 – Results of time in seconds obtained after 100 executions of our enemy generation 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

Source: Elaborated by the author.

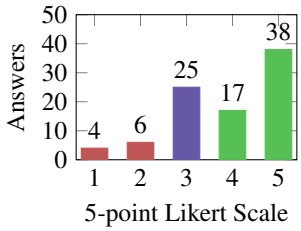
The Evolutionary Algorithm presented by Pereira, Viana and Toledo (2021) can generate a huge number of enemies in a single execution. Although our approach generates significantly fewer enemies, it ensures their diversity. Our approach was slightly faster regarding average execution time since their lower average time was 0.168 seconds.

5.3 Gameplay Feedback

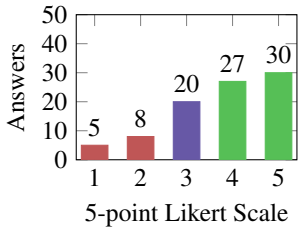
This section reports the results of volunteer players' feedback regarding the levels, locked-door missions, and enemies generated by our approaches and put together by our orchestrator. The players answered a post-questionnaire for each level played, and they were identified with a unique ID, but we cannot ensure that the same player played more than once playthrough. Figure 14 presents the distribution of the volunteer players regarding their experience with general games and with Action-Adventure games. Most volunteers are experienced in both.

Figure 14 – The volunteer players' experience (90 out of 96). Six of them did not answer the questions.

(a) Q1 - I am an experienced player.



(b) Q2 - Experience in Action-Adventure games.



Source: Elaborated by the author.

5.3.1 Feedback of Levels

Finally, we asked people to play a game prototype with the generated levels, and the players had to answer a questionnaire about each played level. The game prototype is the same introduced by Pereira *et al.* (2021), but our locked-door missions are not generic, which means a key can open only a specific locked door. Also, in our gameplay, the players must defeat enemies to progress, and our rooms may also have blocks that players can use to protect themselves from enemies. Thus, the gameplay in the game prototype, using levels procedurally generated by our MAP-Elites approach, advances from the original one in Pereira *et al.* (2021).

A total of 96 people played the levels, where 74 answered all the questions. They played 121 levels, selected by our orchestrator to feed the game prototype. After finishing a level, the players answered how much they agree or not, on a five-point Likert scale, with the following statements, which were adapted from (HEIJNE, 2016; FERREIRA; TOLEDO, 2017)[3]:

Q1 The level was fun to play;

Q2 The level was difficult to complete;

Q3 The challenge was just right (balanced);

Q4 I liked the amount of exploration available on this level;

[3] The game was in Portuguese, so we translated such questions from English into Portuguese.

Q5 I liked the challenge of finding the keys to this level;

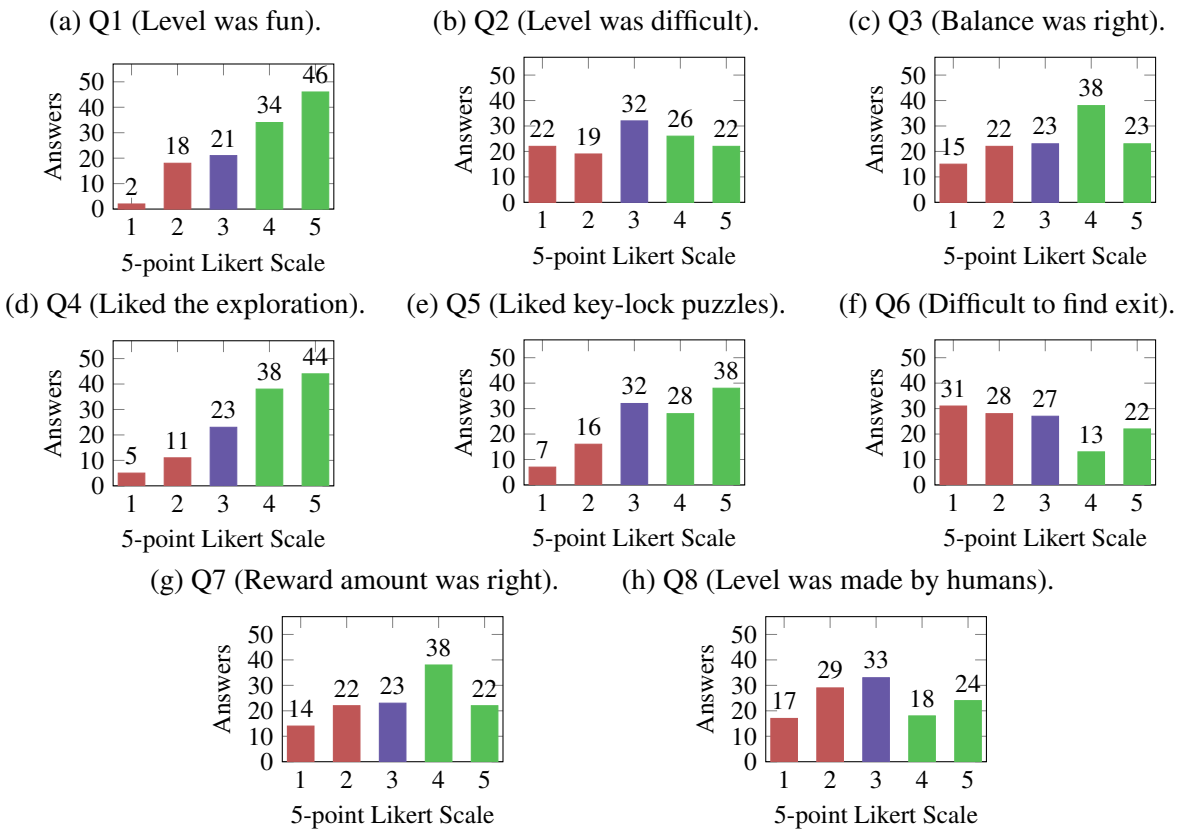
Q6 It was difficult to find the exit/goal of this level;

Q7 The rewards were on the right amount;

Q8 The levels I played were created by humans.

Figure 15 presents seven bar charts, each one with the answers to a question. Table 10 summarizes these answers by presenting the Average (AVG) and Standard Deviation (STD) of each question. The low SD ($\cong 1$) shows that the responses vary slightly. In Figure 15a, the players had fun while playing 80 out of 121 levels, and only 20 did not enjoy it. Figure 15b shows that most players did not have difficulty completing most of our levels (73 out of 121) and Figure 15c shows the players felt that the challenge of 61 levels was just right, it was not good for 37 levels, and they felt neutral for 23 levels.

Figure 15 – Bar charts of answers of the 74 players for 121 levels. Each bar corresponds to the number of levels evaluated for the respective value of the five-point Likert scale.



Source: Elaborated by the author.

Players liked the exploration of 82 levels, as shown in Figure 15d, they did not enjoy only 16 levels, and they were neutral for 23 levels. We observe in Figure 15e that the players liked the locked-doors puzzles in 66 levels; they did not like it in 23 levels and were neutral about it in 32 levels. The players easily found the goal room in 59 levels in Figure 15f; the goal

Table 10 – Average (AVG) and Standard Deviation (STD) of answers of the 74 players for 121 levels.

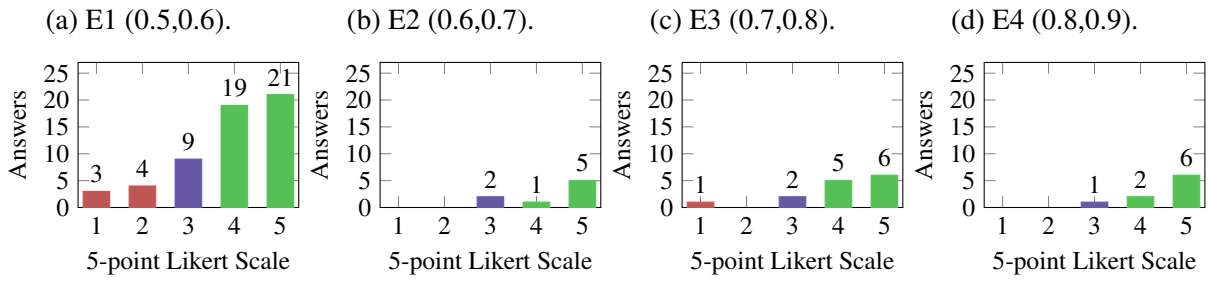
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
AVG	3.85	3.05	3.26	3.86	3.61	2.72	3.12	3.02
STD	1.13	1.35	1.30	1.13	1.22	1.42	1.24	1.32

Source: Elaborated by the author.

room was difficult to find in only 35 levels. Figure 15g shows that the number of rewards was enough for 60 players, neutral for 23, and not enough for 36. Finally, Figure 15h shows that the players believed that humans created 42 levels, 46 levels were generated by a PCG algorithm, and 33 had no sure.

Therefore, even while playing different levels, most players felt that playing the generated content was fun, with a balanced difficulty that brought a good challenge in combat against enemies and a good feeling of exploration in the dungeons. They liked the locked-doors puzzles while not finding it very difficult to find the exit. Thus, our algorithm was able to bring quality and diversity to the solutions while also creating the content so that players could not accurately point out if an algorithm made it.

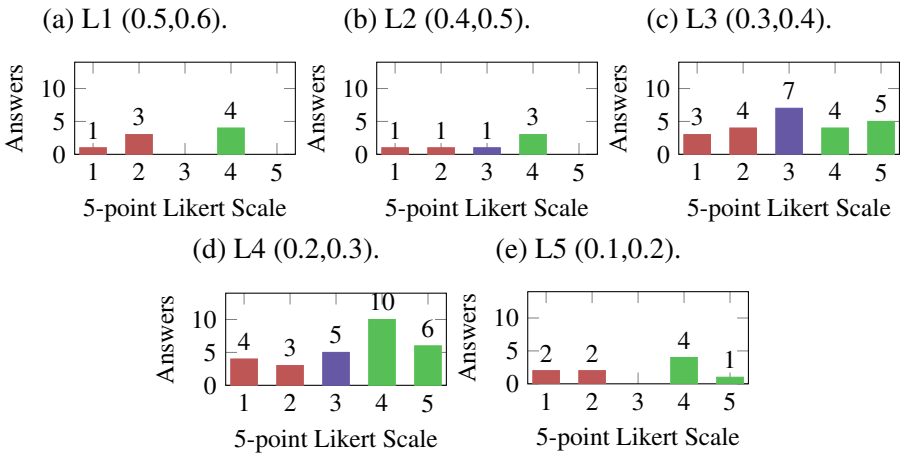
Figure 16 – Bar charts of answers for question Q5 (“I liked the challenge of finding the keys to this level”) of 57 players for 93 levels. These players answered, through a pre-questionnaire, they enjoy exploring. Each bar corresponds to the number of levels evaluated for the respective value of the five-point Likert scale. Each figure correspond to a descriptor of exploration coefficient.



Source: Elaborated by the author.

Besides the questionnaire to evaluate levels, we also asked the players if they enjoyed exploring and battling during their gameplay. Figure 16 shows the feedback of the exploration of levels that were played by players that enjoy exploring by class of exploration coefficient. Most players played E1 levels, and no one played E5 levels; however, most enjoyed playing all the levels independent of the value of the exploration coefficient. Figure 17 shows the feedback of levels of the players that enjoy battling. The results vary more in these charts, with most players playing L4 levels and agreeing with the challenge in the levels they played. Regarding the levels with the remaining leniency values, we cannot declare if they presented the just right amount of challenge since the number of players who agreed and disagreed with that is too close.

Figure 17 – Bar charts of answers for question Q3 (“The challenge was just right”) of 43 players for 74 levels. These players answered, through a pre-questionnaire, they enjoy battles. Each bar corresponds to the number of levels evaluated for the respective value of the five-point Likert scale. Each figure correspond to a descriptor of leniency.



Source: Elaborated by the author.

5.3.2 Feedback of Enemies

A total of 96 people faced our enemies in the game; 75 of them answered all the two following questions. They played 124 levels with enemies placed in rooms by our orchestrator. Regarding difficulty, they played 31 levels with easy enemies, 35 with medium enemies, and 58 with hard enemies. After playing the levels, the players answered how much they agree or disagree, on a five-point Likert scale, with the following statements, which were adapted from (HEIJNE, 2016; FERREIRA; TOLEDO, 2017)[4]:

Q9 The enemies of this level were difficult to defeat;

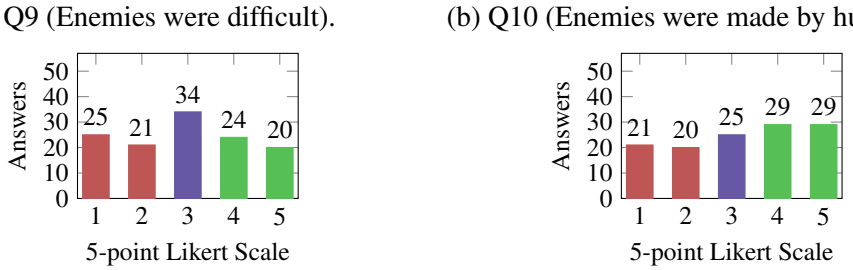
Q10 The enemies I faced were created by humans.

Figure 18 shows the overall results of players’ feedback from our enemies for each question in the questionnaire. Figure 18b shows the players felt the enemies in 80 levels were not difficult to defeat. However, in terms of the challenge suitability (Figure 15c), they answered that the challenge presented was just right in half of the levels (61 out of 121), and they were neutral for 23 of the levels. Finally, in Figure 18b, we observe that the players felt that humans created our enemies in 58 of the levels, and they were neutral for enemies in 25 levels.

Figure 19 shows results of question Q1 for each difficulty. We observe that the players enjoyed most levels they played regardless of the difficulty of the enemies. However, it is clear that they preferred medium and hard levels since the negative answers decreases while the positive answers increases.

[4] The game was in Portuguese, so we translated such questions from English into Portuguese.

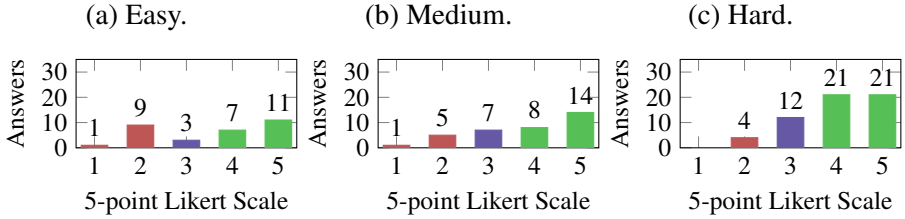
Figure 18 – Bar charts of answers of the 75 players after playing 124 levels. Each bar corresponds to the number of levels evaluated for the respective value of the five-point Likert scale.



Source: Elaborated by the author.

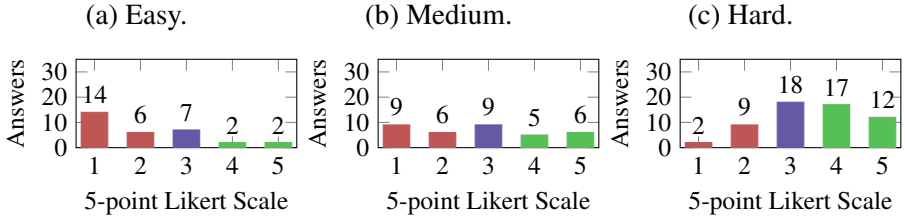
Figure 20 shows results of question Q2 for each difficulty. In Figure 20a, our expectations were confirmed since the players felt that the enemies in 20 out of 31 easy levels were not difficult to defeat. Only in 4 levels, the enemies were perceived as hard to overcome. The results for medium levels show that the players felt enemies easy to defeat in 15 levels, hard in 11 levels, and they were neutral in 9 levels (Figure 20b). Finally, from 58 hard levels, Figure 20c shows that the players felt the enemies hard to defeat in 29 levels. They perceived enemies as easy to defeat in only 10 levels and neutral in 18 levels. The results of feedback for the difficulty of our enemies are good enough to corroborate our choices of difficulty values.

Figure 19 – Bar charts of answers for question Q1 (“The level was fun to play”). Each bar corresponds to the number of levels evaluated for the respective value of the five-point Likert scale.



Source: Elaborated by the author.

Figure 20 – Bar charts of answers for question Q9 (“The enemies of this level were difficult to defeat”). Each bar corresponds to the number of levels evaluated for the respective value of the five-point Likert scale.

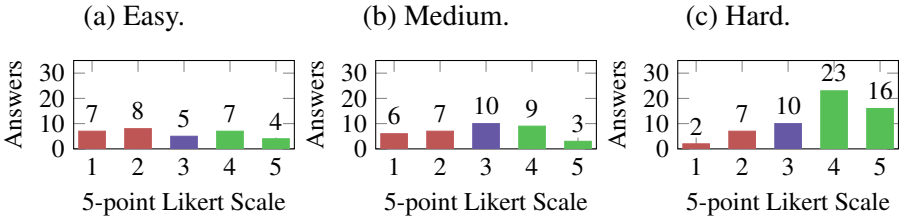


Source: Elaborated by the author.

Figure 21 shows results of question Q3 for each difficulty. Figures 21a and 21b show that approximately half of the players agree and half disagree that easy and medium levels1

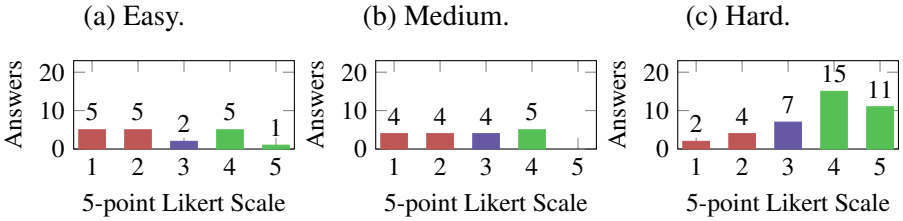
challenges were just right. In contrast, Figure 21c shows that the players perceived the challenge as just right in most hard levels (39 out of 58). Besides the questionnaire to evaluate levels, we also asked the players if they enjoyed battling during their gameplay. Figure 22 shows the results of the players who confirmed such question. The results are analogous to Figure 21, and these players also preferred the levels with the harder enemies, as shown in Figure 22c. Considering such results, we believe our levels probably could present better challenges if we mix up some enemies with different difficulty degrees.

Figure 21 – Bar charts of answers for question Q3 (“The challenge was just right”). Each bar corresponds to the number of levels evaluated for the respective value of the five-point Likert scale.



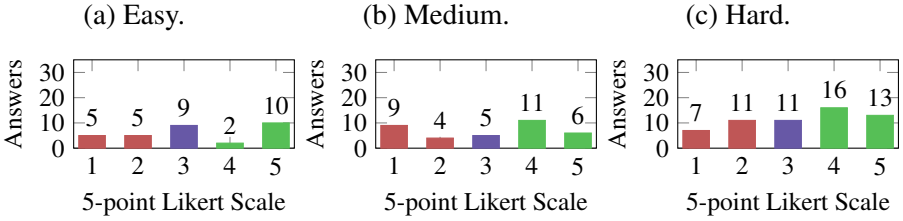
Source: Elaborated by the author.

Figure 22 – Bar charts of answers for question Q3 (“The challenge was just right”) of 43 players for 74 levels. These players answered they enjoy battles. Each bar corresponds to the number of levels evaluated for the respective value of the five-point Likert scale.



Source: Elaborated by the author.

Figure 23 – Bar charts of answers for question Q10 (“The enemies I faced were created by humans”). Each bar corresponds to the number of levels evaluated for the respective value of the five-point Likert scale.



Source: Elaborated by the author.

Figure 23 shows results of question Q4 for each difficulty. The results show that the players perceived the enemies as human-made in most levels regardless of the difficulty degree of the enemies. Besides, this perception is more visible for medium and hard enemies than easy

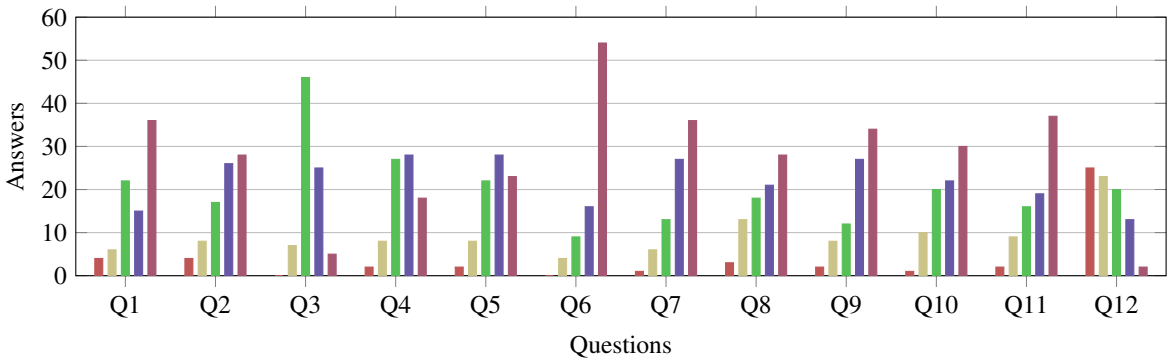
ones. These results mean that the harder the enemy to be overcome, the more players felt they were more carefully designed.

5.4 Player Profiling

In this section, we present the results regarding player profiling. As in feedback of levels, we only considered the answers from 74 players since they answered all the questions of our post-questionnaire. Figure 24 shows the bar plot for each question, counting the responses of each item in the 5-point Likert scale for each question. We observe that most volunteers classify themselves as experient players in both general games and Action-Adventure ones (Q1 and Q2).

Questions from 3 to 12 are used to classify players. The players prefer medium difficulty when playing games and slightly prefer harder difficulties than easier ones (Q3). They were mostly neutral or enjoyed a little the action elements on games (Q4 and Q5). Positive answers to such questions can lead players to the Mastery profile. In contrast, the volunteers generally presented a greater tendency for exploration (Q6 and Q7); such preference defines them as Creativity profile. The majority of players also answered positively for questions Q9 and Q10, showing they enjoy collecting items and finishing quests (Achievement profile), and for Q10 and Q11, indicating they like interacting with the game story and NPCs. Finally, the volunteers answered negatively last question (Q12), which shows that most players prefer to have free time to experience the game instead of rushing to look for action.

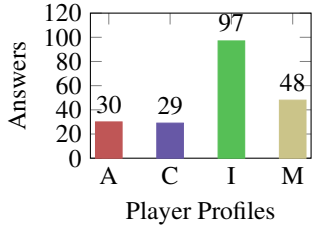
Figure 24 – Bar chart grouping the count of answers for each point in the 5-point Likert scale for all questions in the pre-questionnaire (Table 5). Strongly Disagree responses are in red, Disagree in yellow, Neutral in green, Agree in blue, and Strongly Agree in purple.



Source: Elaborated by the author.

Figure 25 presents the distribution of answers of our post-questionnaire grouped regarding the player profiles. The pre-questionnaire answers indicated that our levels were played by Creativity-oriented players. However, Figure 25 shows that most players classified as Immersion profile played more levels (97), followed by Mastery players (48). Few players were classified as Achievement and Creativity; hence we had fewer answers, 30 and 29, respectively.

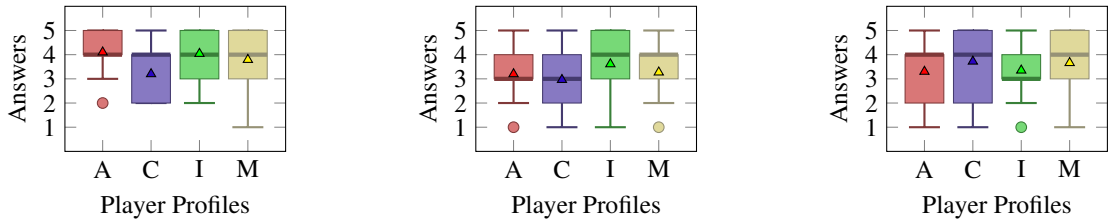
Figure 25 – Bar chart of answers by profile. Since players could answer after each playthrough (i.e., playing each level), there are more answers than the number of total players. From left to right: Achievement (A), Creativity (C), Immersion (I), and Mastery (M).



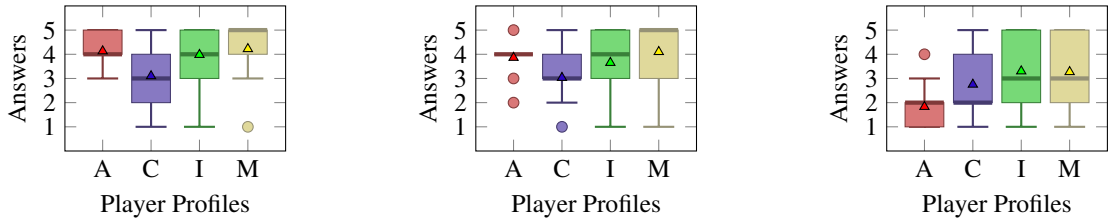
Source: Elaborated by the author.

Figure 26 – Box plot charts, grouping the answers for each post-questionnaire question by the player's actual profile. The dark lines highlight the median, and the triangles mark the average. From left to right: Achievement (A), Creativity (C), Immersion (I), and Mastery (M).

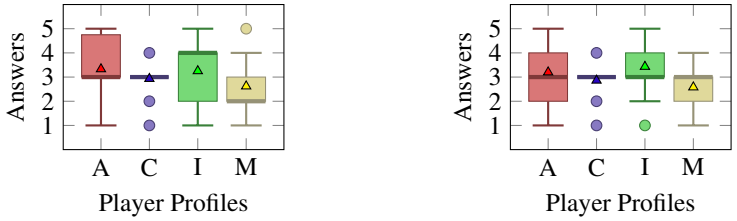
(a) Q1 (Level was fun). (b) Q2 (Level was difficult). (c) Q3 (Balance was right).



(d) Q4 (Liked the exploration). (e) Q5 (Liked key-lock puzzles). (f) Q6 (Difficult to find exit).



(g) Q7 (Reward amount was right). (h) Q9 (Enemies were difficult).



Source: Elaborated by the author.

Figure 26 presents the box plot charts of answers for each post-questionnaire question, which show the difference in the player profiles' answers. First, we can observe that, in general, most answers' average and standard deviations are slightly distinct for different profiles. Figure 26a shows that, regarding average, Creativity-oriented players had less fun when playing our levels than the remaining group of players. The Immersion players perceived the level difficulty as slightly harder, while the Creativity ones perceived it as a little easier (Figure 26b). Regarding the difficulty of enemies (Figure 26h), the Mastery players felt them easier, in general, than the

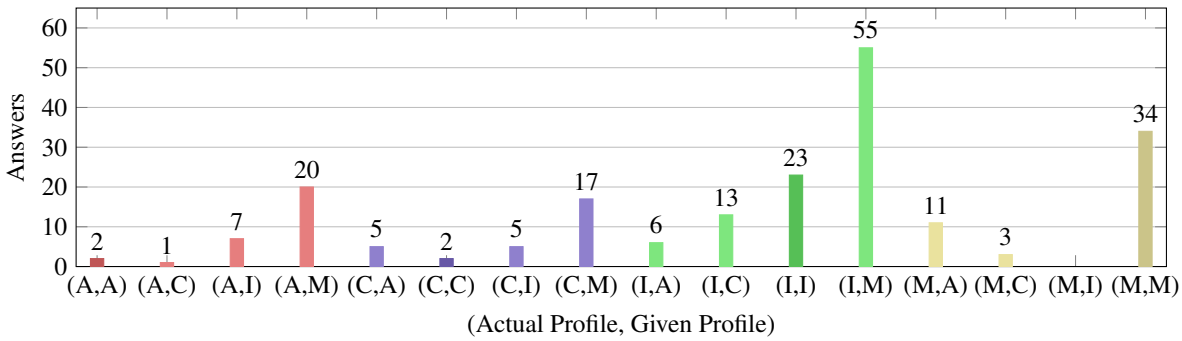
remaining groups. Such a result is expected since these players enjoy combat over the other motivations.

The players of Creativity and Mastery groups felt the balance slightly better than those of Achievement and Immersion groups (Figure 26c). Regarding rewards Figure 26g, they were perceived as not enough for Mastery players. Creativity and Immersion players felt, on average, neutral about rewards. In contrast, the Achievement players found rewards enough for them. We believe such a result is a consequence of their motivation to collect items, such as rewards.

Regarding exploration (Figure 26d), all groups but the Creativity one enjoyed the exploration of our levels. This result is similar for the answers of Q5 regarding the challenge of finding locks and keys (Figure 26e), except for Achievement players that found it slightly less challenging than the players of Mastery and Immersion. Achievement players found to find the levels' exit (Figure 26f) easier than the remaining groups, followed by the Creativity group. The Immersion and Mastery players felt like an average difficulty.

However, we divided the players into the test and the control groups. We delivered the appropriate content for the players in the test group, and for the players in the control group, we delivered the content of other groups. We made this to analyze if our PCG system could deliver better entertainment and challenge to the test group when compared to this control group. Figure 27 presents the distribution of players grouped by their actual and given profile. The players in the given profile are those in the control group. There are few Achievement and Creativity players; thus, we do not have enough data to make a comparative analysis for these groups. Therefore, we analyzed the answers of Mastery and Immersion profiles from groups that presented at least 10 answers.

Figure 27 – Bar chart of answers by Actual Profile and Given Profile. The dark colors are the answers of players who were classified as their actual profile. The light colors are the answers of players who were given other profiles.



Source: Elaborated by the author.

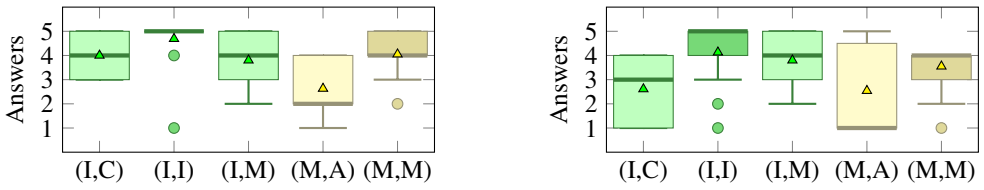
Such grouping resulted in the charts of Figure 28. It shows similar box plots to those of Figure 26. Thus, the figure shows the distribution of answers to each question of the post-questionnaire; however, it now groups the actual player profile and the given profile. The Immersion profile is in green and Mastery profile is in yellow. The lighter hue represents the

group of players with different profiles from the actual one. The darker hue is for those who played with content matching their actual profile.

Figure 28 – Box plot charts, grouping the answers for each post-questionnaire question by the pair of player's actual profile and given profile. The dark lines highlight the median, and the triangles mark the average. From left to right: Achievement (A), Creativity (C), Immersion (I), and Mastery (M).

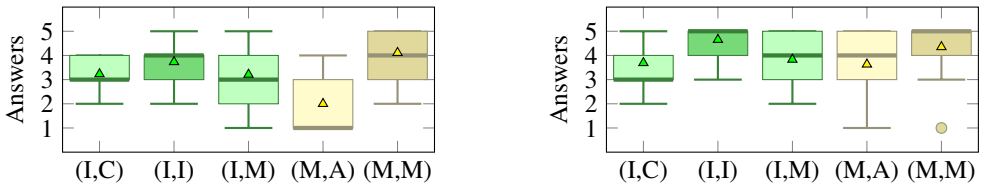
(a) Q1 (Level was fun).

(b) Q2 (Level was difficult).



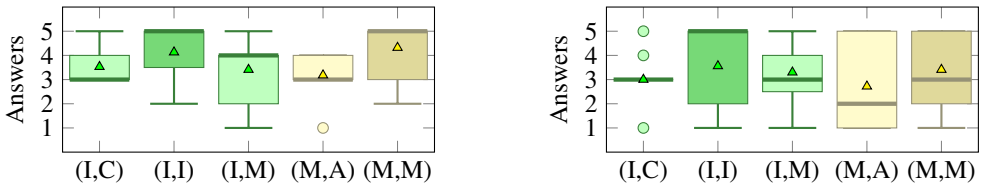
(c) Q3 (Balance was right).

(d) Q4 (Liked the exploration).



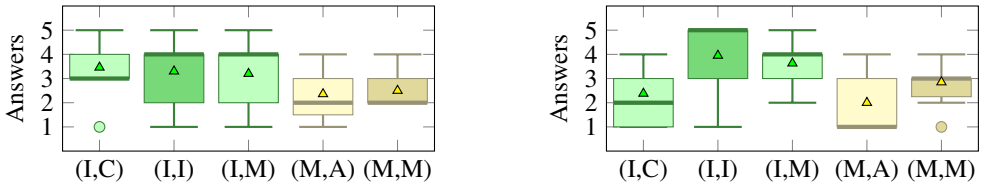
(e) Q5 (Liked key-lock puzzles).

(f) Q6 (Difficult to find exit).



(g) Q7 (Reward amount was right).

(h) Q9 (Enemies were difficult).



Source: Elaborated by the author.

In Figure 28, we can observe differences between both test and control groups. Particularly, Figure 28a shows very interesting results for Q1: the players in the test group enjoyed, on average, more playing the game than those in the control group. The test group also found the levels and the enemies more difficult, as shown in Figure 28b and Figure 28h, respectively. However, regarding the challenge to find the levels' exit was felt somewhat the same between all the groups (Figure 28f). Although perceiving the levels and enemies more difficult (Figure 28c), both the test and control groups of Immersion had a similar positive opinion regarding the game balance. The Mastery test group found the game more balanced in contrast to the remaining

groups. Both groups felt similarly about the rewards for the same profile (Figure 28g). Besides, both groups had a somewhat similar opinion regarding rewards (Figure 28g) and also regarding exploration (Figure 28d), the latter with slightly more positive answers. Finally, all the test groups enjoyed the challenge of locked-door missions (Figure 28e).

CHAPTER

6

FINAL REMARKS

The present dissertation described how we solved the problem of adaptive content orchestration regards the coordination of levels, missions, and enemies for an Action-Adventure game and players of different profiles. We also described the contributions achieved by a PCG system that integrates the so-called Overlord project (PEREIRA, 2021). We designed such a system to generate multiple contents customized to entertain different types of players aiming to answer the following research question:

> *How can a system orchestrate the generation of levels, narratives, and rules facets and adapt them to provide coherent combinations of contents for different types of players?*

Our system consisted of three modules: Game Prototype, Orchestrator, and Player Profile. The Game Prototype module was responsible for providing the game for players and collecting data from them. The Orchestrator coordinated two generators, one for levels and the other for enemies, and then performed a post-processing process to combine levels and enemies accordingly to the player profile. Such profile was classified by our Player Profile module, which is based on four equations and weights from our player preferences questionnaire questions.

Thus, our work contributed to PCG research by advancing two PCG algorithms. The first one generates levels with locked-door missions while placing enemies in the levels' rooms; such an enemy placement is an advance the method of Pereira *et al.* (2021). Another advancement in such an algorithm was the application of a MAP-Elites population to evolve diverse levels without losing quality. The second algorithm is the enemy generator; we advanced it from Pereira, Viana and Toledo (2021) also by applying a MAP-Elites approach to illuminate the space and, thus, generating diverse enemies. Our PCG system is a relevant contribution concerning content orchestration since it can coordinate three creative facets (levels, narrative, and rules) by generating levels with locked-door missions and enemies. Furthermore, our system can also adapt this content to provide different experiences to different players based on their profiles.

Our computational results corroborate such contributions. The experiments showed that both level and enemy generation algorithms accurately converged almost the entire population for most executions and cases. Our experiments with volunteer players indicated that most of them liked the levels and the enemies. We also successfully created enemies that could be distinguished as easy, medium, or hard to face. Besides, most players could not identify if algorithms or humans created the levels and enemies. Our experiments with players also showed that our PCG system presented positive results in terms of content adaptation since the players of Mastery and Immersion enjoyed the content we delivered for them. Therefore, we can conclude that our PCG system can create levels, missions, and enemies capable of entertaining players.

As future works, we intend to improve our player profiling by using a gamified version of our pre-questionnaire to provide a funnier way of classifying players. Thus, we could collect more meaningful data and reduce bias, turning our classification more trustworthy. We believe that such an alternative could improve our results regarding the players' feedback. For instance, we could provide options of playable characters with descriptions: warrior (players who prefer close combat), hunter (for players who prefer ranged combat), diplomat (for players who prefer interacting with the world), among others. Besides, we could improve adapted content by classifying players with gameplay data, which can allow us to perform online classification without using multiple times (boring) questionnaires and, hence, online orchestration.

Regarding our level generation approach, its main limitation is that its levels present similar tree structures. Therefore, to allow more distinct levels in the population, we intend to add a novelty score as proposed in (CONTI *et al.*, 2017). Another way is to explore how our level generator can be integrated into an interactive evolution in a game designer tool. In this way, this tool could improve the productivity of game designers by providing suggestions for changes in levels created by them, like in EDD (ALVAREZ *et al.*, 2019). Concerning our enemy generation approach, we aim to extend our work with the Constrained MAP-Elites approach (GRAVINA *et al.*, 2019). With this approach, we can generate multiple enemies and avoid incoherent enemies. The main limitation of our enemy generator is the fitness function that is strongly related to our game prototype. To overcome such a limitation, we intend to apply simulation-based fitness, like in Khalifa *et al.* (2018). Finally, regarding our orchestration process, we aim to explore how could we add new creative facets, e.g., NPCs, music, among others, and how other types of orchestrators could improve our PCG system.

Furthermore, in our next steps, we intend to contact specialists in game designing to cooperate with us to design better the game rhythm, fitness functions, and MAP-Elites mapping, hopefully, for all the creative facets that we will be working on. Thus, we better attend to the expectations of the industry's game designers when using PCG methods for automatizing the creative process of creating content.

BIBLIOGRAPHY

ALVAREZ, A.; DAHLSKOG, S.; FONT, J.; HOLMBERG, J.; JOHANSSON, S. Assessing aesthetic criteria in the evolutionary dungeon designer. In: ACM. **Proceedings of the 13th International Conference on the Foundations of Digital Games**. [S.l.], 2018. p. 44. Citations on pages 49, 50, and 51.

ALVAREZ, A.; DAHLSKOG, S.; FONT, J.; TOGELIUS, J. Empowering quality diversity in dungeon design with interactive constrained map-elites. In: IEEE. **2019 IEEE Conference on Games (CoG)**. [S.l.], 2019. p. 1–8. Citations on pages 48, 49, 50, 51, and 92.

Atari. **Pong**. 1972. Accessed in: 2022-01-28. Available: <https://www.ponggame.org/.> Citation on page 35.

AZADVAR, A.; CANOSSA, A. Upeq: ubisoft perceived experience questionnaire: a self-determination evaluation tool for video games. In: **Proceedings of the 13th international conference on the foundations of digital games**. [S.l.: s.n.], 2018. p. 1–7. Citation on page 43.

BALDWIN, A.; DAHLSKOG, S.; FONT, J. M.; HOLMBERG, J. Mixed-initiative procedural generation of dungeons using game design patterns. In: IEEE. **Computational Intelligence and Games (CIG), 2017 IEEE Conference on**. [S.l.], 2017. p. 25–32. Citations on pages 47, 49, 50, and 51.

____. Towards pattern-based mixed-initiative dungeon generation. In: ACM. **Proceedings of the 12th International Conference on the Foundations of Digital Games**. [S.l.], 2017. p. 74. Citations on pages 47, 49, 50, and 51.

BARTLE, R. A. **Designing virtual worlds**. [S.l.]: New Riders, 2004. Citations on pages 39, 41, 42, and 45.

BELL, I.; BRABEN, D. **Elite**. 1984. Accessed in: 2020-09-11. Available: <https://www.mobygames.com/game/elite.> Citation on page 33.

BICHO, F.; MARTINHO, C. Multi-dimensional player skill progression modelling for procedural content generation. In: **Proceedings of the 13th International Conference on the Foundations of Digital Games**. [S.l.: s.n.], 2018. p. 1–10. Citations on pages 43 and 45.

Blizzard. **Diablo III**. 2012. Acessado em: 2021-02-11. Available: <https://us.diablo3.com/en/.> Citations on pages 50 and 51.

BONTCHEV, B.; GEORGIEVA, O. Playing style recognition through an adaptive video game. **Computers in Human Behavior**, Elsevier, v. 82, p. 136–147, 2018. Citation on page 45.

BROWNE, C.; MAIRE, F. Evolutionary game design. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE, v. 2, n. 1, p. 1–16, 2010. Citations on pages 52 and 54.

CHARITY, M.; GREEN, M. C.; KHALIFA, A.; TOGELIUS, J. Mech-elites: Illuminating the mechanic space of gvgai. **arXiv preprint arXiv:2002.04733**, 2020. Citations on pages 48, 49, 51, 52, and 54.

CONTI, E.; MADHAVAN, V.; SUCH, F. P.; LEHMAN, J.; STANLEY, K. O.; CLUNE, J. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. **arXiv preprint arXiv:1712.06560**, 2017. Citation on page 92.

COOK, M.; COLTON, S. A rogue dream: Automatically generating meaningful content for games. In: **Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**. [S.l.: s.n.], 2014. v. 10, n. 1. Citations on pages 53 and 54.

COOK, M.; COLTON, S.; PEASE, A. Aesthetic considerations for automated platformer design. In: **Eighth Artificial Intelligence and Interactive Digital Entertainment Conference**. [S.l.: s.n.], 2012. Citations on pages 52 and 54.

COOK, M.; COLTON, S.; RAAD, A.; GOW, J. Mechanic miner: Reflection-driven game mechanic discovery and level design. In: SPRINGER. **European Conference on the Applications of Evolutionary Computation**. [S.l.], 2013. p. 284–293. Citations on pages 52 and 54.

COWLEY, B.; CHARLES, D. Behavlets: a method for practical player modelling using psychology-based player traits and domain specific features. **User Modeling and User-Adapted Interaction**, Springer, v. 26, n. 2, p. 257–306, 2016. Citation on page 45.

Creature Labs. **Creatures**. 1996. Accessed in: 2021-02-11. Available: <https://creatures.fandom.com/wiki/Creatures.> Citations on pages 50 and 51.

Digital Sun. **Moonlighter**. 2018. Accessed in: 2020-07-25. Available: <http://moonlighterthegame.com/.> Citations on pages 27 and 34.

DONNELLY, J. **GTA 5 estimated to be the most profitable entertainment product of all time**. 2018. Accessed in: 2020-08-15. Available: <https://www.pcgamer.com/gta-5-estimated-to-be-the-most-profitable-entertainment-product-of-all-time/.> Citation on page 27.

DORMANS, J. Adventures in level design: generating missions and spaces for action adventure games. In: ACM. **Proceedings of the 2010 workshop on procedural content generation in games**. [S.l.], 2010. p. 1. Citations on pages 31, 32, 46, 47, 49, 50, 51, 52, and 54.

EIBEN, A. E.; SMITH, J. E. *et al.* **Introduction to evolutionary computing**. [S.l.]: Springer, 2003. Citations on pages 19, 36, and 37.

ESHELMAN, L. J.; SCHAFFER, J. D. Real-coded genetic algorithms and interval-schemata. In: **Foundations of genetic algorithms**. [S.l.]: Elsevier, 1993. v. 2, p. 187–202. Citation on page 68.

Evolutionary Games. **Galact Arms Race**. 2014. Accessed in: 2020-09-10. Available: <http://galacticarmsrace.blogspot.com/.> Citation on page 27.

FERREIRA, L. N.; TOLEDO, C. F. M. Tanager: A generator of feasible and engaging levels for angry birds. **IEEE Transactions on Games**, IEEE, v. 10, n. 3, p. 304–316, 2017. Citations on pages 79 and 82.

Firaxis Games. **Civilization VI**. 2016. Accessed in: 2020-09-11. Available: <https://civilization.com.> Citation on page 34.

Game Terra. **Statistically, video games are now the most popular and profitable form of entertainment**. 2018. Available: <https://www.gamerterra.com/news/statistically-video-games-are-now-most-popular-and-profitable-form-entertainment/>. Citation on page 27.

Gearbox Software. **Borderlands**. 2009. Accessed in: 2020-09-10. Available: <https://borderlands.com/.> Citation on page 27.

GELLEL, A.; SWEETSER, P. A hybrid approach to procedural generation of roguelike video game levels. In: **International Conference on the Foundations of Digital Games**. [S.l.: s.n.], 2020. p. 1–10. Citations on pages 34, 46, 47, 49, 50, 51, 52, and 54.

GRAND, S.; CLIFF, D. Creatures: Entertainment software agents with artificial life. **Autonomous Agents and Multi-Agent Systems**, Springer, v. 1, n. 1, p. 39–57, 1998. Citation on page 50.

GRAVINA, D.; KHALIFA, A.; LIAPIS, A.; TOGELIUS, J.; YANNAKAKIS, G. N. Procedural content generation through quality diversity. In: IEEE. **2019 IEEE Conference on Games (CoG)**. [S.l.], 2019. p. 1–8. Citations on pages 29, 37, and 92.

GREEN, M. C.; BARROS, G. A. B.; LIAPIS, A.; TOGELIUS, J. Data agent. In: **Proceedings of the 13th International Conference on the Foundations of Digital Games**. New York, NY, USA: Association for Computing Machinery, 2018. (FDG '18). ISBN 9781450365710. Available: <https://doi.org/10.1145/3235765.3235792>. Citations on pages 53 and 54.

Guerrilla Games. **Horizon Zero Dawn**. 2017. Accessed in: 2020-08-10. Available: <https://store.playstation.com/pt-br/product/UP9000-CUSA10237_00-HRZCE00000000000?smcid=en%3Apage%20name%3A%3AHorizon%20Zero%20Dawn%E2%84%A2%20Complete%20Edition.> Citations on pages 31 and 32.

GUTIERREZ, J.; SCHRUM, J. Generative adversarial network rooms in generative graph grammar dungeons for the legend of zelda. **arXiv preprint arXiv:2001.05065**, 2020. Citation on page 34.

HARTSOOK, K.; ZOOK, A.; DAS, S.; RIEDL, M. O. Toward supporting stories with procedurally generated game worlds. In: IEEE. **2011 IEEE Conference on Computational Intelligence and Games (CIG'11)**. [S.l.], 2011. p. 297–304. Citations on pages 52 and 54.

HEIJNE, N. **Investigating the Relationship between FFM, Game Literacy, Content Generation and Game-play Preference**. Phd Thesis (PhD Thesis) — Dissertação (Mestrado)—University of Amsterdam, 2016. Disponível em:< https ..., 2016. Citations on pages 43, 44, 45, 49, 51, 59, 79, and 82.

HEIJNE, N.; BAKKES, S. Procedural zelda: A pcg environment for player experience research. In: **Proceedings of the 12th International Conference on the Foundations of Digital Games**. [S.l.: s.n.], 2017. p. 1–10. Citation on page 43.

Hello Games. **No Man's Sky**. 2018. Accessed in: 2020-09-11. Available: <https://www.nomanssky.com.> Citations on pages 34, 50, and 51.

HEWGILL, A.; ROSS, B. J. Procedural 3d texture synthesis using genetic programming. **Computers & Graphics**, Elsevier, v. 28, n. 4, p. 569–584, 2004. Citation on page 28.

HOOVER, A. K.; CACHIA, W.; LIAPIS, A.; YANNAKAKIS, G. N. Audioinspace: Exploring the creative fusion of generative audio, visuals and gameplay. In: SPRINGER. **International Conference on Evolutionary and Biologically Inspired Music and Art**. [S.l.], 2015. p. 101–112. Citations on pages 52 and 54.

JAMESON, A. Adaptive interfaces and agents. **The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications**, CRC Press, Boca Raton, FL, p. 305–330, 2008. Citations on pages 39, 44, and 60.

KANAGAL-SHAMANNA, R.; PORTIER, B. P.; SINGH, R. R.; ROUTBORT, M. J.; ALDAPE, K. D.; HANDAL, B. A.; RAHIMI, H.; REDDY, N. G.; BARKOH, B. A.; MISHRA, B. M. *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**, Nature Publishing Group, v. 27, n. 2, p. 314–327, 2014. Citation on page 68.

KARAVOLOS, D.; LIAPIS, A.; YANNAKAKIS, G. N. Evolving missions for dwarf quest dungeons. In: **2016 IEEE Conference on Computational Intelligence and Games (CIG)**. [S.l.: s.n.], 2016. p. 1–2. ISSN 2325-4289. Citations on pages 46, 47, 49, 50, 51, 52, and 54.

____. A multi-faceted surrogate model for search-based procedural content generation. **IEEE Transactions on Games**, IEEE, 2019. Citations on pages 53 and 54.

KHALIFA, A.; LEE, S.; NEALEN, A.; TOGELIUS, J. Talakat: Bullet hell generation through constrained map-elites. In: **Proceedings of The Genetic and Evolutionary Computation Conference**. [S.l.: s.n.], 2018. p. 1047–1054. Citations on pages 38, 50, 51, and 92.

KIMBROUGH, S. O.; KOEHLER, G. J.; LU, M.; WOOD, D. H. Introducing a feasible-infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. **European Journal of Operational Research**, Citeseer, v. 190, 2005. Citation on page 37.

Koema. **Top 100 Countries by Game Revenues**. 2019. Available: <https://knoema.com/infographics/tqldbq/top-100-countries-by-game-revenues>. Citation on page 27.

KONERT, J.; GUTJAHR, M.; GÖBEL, S.; STEINMETZ, R. Modeling the player: Predictability of the models of bartle and kolb based on neo-ffi (big5) and the implications for game based learning. **International Journal of Game-Based Learning (IJGBL)**, IGI Global, v. 4, n. 2, p. 36–50, 2014. Citation on page 45.

LAVENDER, B.; THOMPSON, T. A generative grammar approach for action-adventure map generation in the legend of zelda. 2017. Citations on pages 31, 32, 46, 47, 49, 50, 51, 52, and 54.

LIAPIS, A. Multi-segment evolution of dungeon game levels. In: ACM. **Proceedings of the Genetic and Evolutionary Computation Conference**. [S.l.], 2017. p. 203–210. Citations on pages 31, 32, 47, 49, 50, and 51.

____. 10 years of the pcg workshop: Past and future trends. In: **International Conference on the Foundations of Digital Games**. [S.l.: s.n.], 2020. p. 1–10. Citations on pages 28, 34, and 45.

LIAPIS, A.; KARAVOLOS, D.; MAKANTASIS, K.; SFIKAS, K.; YANNAKAKIS, G. N. Fusing level and ruleset features for multimodal learning of gameplay outcomes. In: IEEE. **2019 IEEE Conference on Games (CoG)**. [S.l.], 2019. p. 1–8. Citations on pages 53 and 54.

LIAPIS, A.; YANNAKAKIS, G.; TOGELIUS, J. Towards a generic method of evaluating game levels. In: **Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**. [S.l.: s.n.], 2013. v. 9, n. 1. Citation on page 62.

LIAPIS, A.; YANNAKAKIS, G. N.; NELSON, M. J.; PREUSS, M.; BIDARRA, R. Orchestrating game generation. **IEEE Transactions on Games**, v. 11, n. 1, p. 48–68, March 2019. ISSN 2475-1502. Citations on pages 28, 34, 35, 52, 56, and 61.

LIBERATI, A.; ALTMAN, D. G.; TETZLAFF, J.; MULROW, C.; GØTZSCHE, P. C.; IOANNIDIS, J. P.; CLARKE, M.; DEVEREAUX, P. J.; KLEIJNEN, J.; MOHER, D. The prisma statement for reporting systematic reviews and meta-analyses of studies that evaluate health care interventions: explanation and elaboration. **Journal of clinical epidemiology**, Elsevier, v. 62, n. 10, p. e1–e34, 2009. Citation on page 46.

LINDEN, R. van der; LOPES, R.; BIDARRA, R. Designing procedurally generated levels. In: **Proceedings of the the second workshop on Artificial Intelligence in the Game Design Process**. [S.l.: s.n.], 2013. Citations on pages 46, 47, 49, 50, 51, 52, and 54.

____. Procedural generation of dungeons. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE, v. 6, n. 1, p. 78–89, 2014. Citations on pages 28 and 32.

LOPES, P.; LIAPIS, A.; YANNAKAKIS, G. N. Framing tension for game generation. In: ICCC. [S.l.], 2016. Citations on pages 53 and 54.

LORIA, E.; MARCONI, A. Player types and player behaviors: analyzing correlations in an on-the-field gamified system. In: **Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts**. [S.l.: s.n.], 2018. p. 531–538. Citation on page 45.

Massive Entertainment. **Tom Clancy's The Division**. 2016. Accessed in: 2021-02-19. Available: <https://thedivision.fandom.com/wiki/Tom_Clancy%27s_The_Division.> Citation on page 43.

Maxis^{TM}. **Spore**. 2008. Accessed in: 2021-02-11. Available: <http://www.spore.com/.> Citations on pages 50 and 51.

MCMILLEN, E.; HIMSL, F. **The Binding of Isaac**. 2011. Accessed in: 2020-07-25. Available: <https://bindingofisaac.com/.> Citations on pages 55 and 57.

MELHART, D.; AZADVAR, A.; CANOSSA, A.; LIAPIS, A.; YANNAKAKIS, G. N. Your gameplay says it all: modelling motivation in tom clancy's the division. In: IEEE. **2019 IEEE Conference on Games (CoG)**. [S.l.], 2019. p. 1–8. Citations on pages 43 and 45.

MELOTTI, A. S.; MORAES, C. H. V. de. Evolving roguelike dungeons with deluged novelty search local competition. **IEEE Transactions on Games**, IEEE, v. 11, n. 2, p. 173–182, 2018. Citations on pages 48 and 49.

MIGKOTZIDIS, P.; LIAPIS, A. Susketch: Surrogate models of gameplay as a design assistant. **IEEE Transactions on Games**, IEEE, 2021. Citation on page 54.

MOHER, D.; LIBERATI, A.; TETZLAFF, J.; ALTMAN, D. G. Preferred reporting items for systematic reviews and meta-analyses: the prisma statement. **Annals of internal medicine**, Am Coll Physicians, v. 151, n. 4, p. 264–269, 2009. Citation on page 46.

Monolith Productions. **Middle-earth: Shadow of Mordor**. 2014. Acessado em: 2021-02-11. Available: <https://store.steampowered.com/app/241930/Middleearth_Shadow_of_Mordor/.> Citations on pages 50 and 51.

Motion Twin. **Dead Cells**. 2018. Accessed in: 2020-07-25. Available: <https://dead-cells.com/.> Citation on page 27.

MOURET, J.-B.; CLUNE, J. Illuminating search spaces by mapping elites. **arXiv preprint arXiv:1504.04909**, 2015. Citation on page 38.

Naughty Dog. **Uncharted**. 2007. Accessed in: 2020-09-11. Available: <https://www.unchartedthegame.com/en-us/.> Citation on page 31.

Nintendo. **The Legend of Zelda - Franchise**. 1986. Accessed in: 2020-09-04. Available: <https://www.zelda.com/.> Citations on pages 31, 43, 55, and 57.

____. **Pokémon - Franchise**. 1996. Accessed in: 2020-09-10. Available: <https://www.pokemon.com/us/.> Citations on pages 15 and 33.

ONO, J.; OGATA, T. Surprise-based narrative generation in an automatic narrative generation game. In: **Content generation through narrative communication and simulation**. [S.l.]: IGI Global, 2018. p. 162–185. Citation on page 34.

ORJI, R.; NACKE, L. E.; MARCO, C. D. Towards personality-driven persuasive health games and gamified systems. In: **Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems**. [S.l.: s.n.], 2017. p. 1015–1027. Citation on page 39.

PEREIRA, L. T. **Adaptative Procedural Generation of Multiple Creative Facets in Video Games by Modeling Player's Profiles and Performance**. Phd Thesis (PhD Thesis) — Universidade de São Paulo, 2021. Citations on pages 28, 59, 71, and 91.

PEREIRA, L. T.; PRADO, P. V.; TOLEDO, C. Evolving dungeon maps with locked door missions. In: IEEE. **2018 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.], 2018. p. 1–8. Citations on pages 31, 32, 46, 49, 52, and 54.

PEREIRA, L. T.; PRADO, P. V. de S.; LOPES, R. M.; TOLEDO, C. F. M. Procedural generation of dungeons' maps and locked-door missions through an evolutionary algorithm validated with players. **Expert Systems with Applications**, Elsevier, v. 180, p. 115009, 2021. Citations on pages 34, 46, 49, 52, 54, 56, 61, 64, 79, and 91.

PEREIRA, L. T.; VIANA, B. M. F.; TOLEDO, C. F. M. Procedural enemy generation through parallel evolutionary algorithm. In: IEEE. **2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)**. [S.l.], 2021. p. 126–135. Citations on pages 50, 51, 65, 66, 67, 78, and 91.

PRAGER, R. P.; TROOST, L.; BRÜGGENJÜRGEN, S.; MELHART, D.; YANNAKAKIS, G.; PREUSS, M. An experiment on game facet combination\. In: IEEE. **2019 IEEE Conference on Games (CoG)**. [S.l.], 2019. p. 1–8. Citations on pages 53 and 54.

PUGH, J. K.; SOROS, L. B.; STANLEY, K. O. Quality diversity: A new frontier for evolutionary computation. **Frontiers in Robotics and AI**, Frontiers, v. 3, p. 40, 2016. Citation on page 38.

REEVES, C. R. Genetic algorithms. In: ____. **Handbook of Metaheuristics**. Boston, MA: Springer US, 2010. p. 109–139. ISBN 978-1-4419-1665-5. Available: <https://doi.org/10.1007/978-1-4419-1665-5_5>. Citation on page 37.

RICHTER, F. **Gaming: The Most Lucrative Entertainment Industry By Far**. 2020. Available: <https://www.statista.com/chart/22392/global-revenue-of-selected-entertainment-industry-sectors/>. Citation on page 27.

RIVERA-VILLICANA, J.; ZAMBETTA, F.; HARLAND, J.; BERRY, M. Informing a bdi player model for an interactive narrative. In: **Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play**. [S.l.: s.n.], 2018. p. 417–428. Citations on pages 44, 45, and 59.

Rockstar Games. **Grand Theft Auto V**. 2013. Accessed in: 2020-08-15. Available: <https://www.rockstargames.com/V/restricted-content/agegate/form?redirect=https%3A%2F%2Fwww.rockstargames.com%2FV%2F&options=&locale=en_us>. Citation on page 27.

SMITH, T.; PADGET, J.; VIDLER, A. Graph-based generation of action-adventure dungeon levels using answer set programming. In: ACM. **Proceedings of the 13th International Conference on the Foundations of Digital Games**. [S.l.], 2018. p. 52. Citations on pages 31, 32, 34, 46, 47, 49, 50, 51, 52, 54, and 62.

Solarus. **The Legend of Zelda: Mystery of Solarus DX**. 1986. Accessed in: 2020-07-25. Available: <https://www.solarus-games.org/en/games/the-legend-of-zelda-mystery-of-solarus-dx.> Citation on page 46.

Statista. **Leading gaming markets in Latin America as of June 2019, by gaming revenue**. 2019. Available: <https://www.statista.com/statistics/500035/gaming-revenue-countries-latin-america/>. Citation on page 27.

SUMMERVILLE, A.; MARIÑO, J. R.; SNODGRASS, S.; ONTAÑÓN, S.; LELIS, L. H. Understanding mario: an evaluation of design metrics for platformers. In: **Proceedings of the 12th international conference on the foundations of digital games**. [S.l.: s.n.], 2017. p. 1–10. Citation on page 65.

SUMMERVILLE, A. J.; BEHROOZ, M.; MATEAS, M.; JHALA, A. The learning of zelda: Data-driven learning of level topology. In: **Proceedings of the FDG workshop on Procedural Content Generation in Games**. [S.l.: s.n.], 2015. Citations on pages 31 and 32.

SUMMERVILLE, A. J.; MATEAS, M. Sampling hyrule: Multi-technique probabilistic level generation for action role playing games. In: **Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference**. [S.l.: s.n.], 2015. Citations on pages 31 and 32.

TAKAHASHI, D. **SuperData: Games grew 12% to $139.9 billion in 2020 amid pandemic**. 2021. Available: <https://venturebeat.com/2021/01/06/superdata-games-grew-12-to-139-9-billion-in-2020-amid-pandemic/>. Citation on page 27.

TOGELIUS, J.; SHAKER, N.; NELSON, M. J. Introduction. In: SHAKER, N.; TOGELIUS, J.; NELSON, M. J. (Ed.). **Procedural Content Generation in Games: A Textbook and an Overview of Current Research**. [S.l.]: Springer, 2016. p. 1–15. Citations on pages 28, 32, 34, and 35.

TOY, M.; WICHMAN, G. **Rogue**. 1980. Accessed in: 2020-07-25. Available: <https://www.mobygames.com/game/rogue.> Citations on pages 27 and 33.

TREANOR, M.; SCHWEIZER, B.; BOGOST, I.; MATEAS, M. The micro-rhetorics of game-o-matic. In: **Proceedings of the International Conference on the Foundations of Digital Games**. [S.l.: s.n.], 2012. p. 18–25. Citations on pages 52 and 54.

Undead Labs. **State of Decay**. 2018. Accessed in: 2021-02-11. Available: <https://state-of-decay-2.fandom.com/wiki/State_of_Decay_2_Wiki.> Citations on pages 50 and 51.

VAHLO, J.; KAAKINEN, J. K.; HOLM, S. K.; KOPONEN, A. Digital game dynamics preferences and player types. **Journal of Computer-Mediated Communication**, Oxford University Press Oxford, UK, v. 22, n. 2, p. 88–103, 2017. Citations on pages 39, 42, 45, and 59.

VALENTE, A. **The 10 Most Expensive Games Ever Made, Ranked**. 2019. Accessed in: 2020-08-15. Available: <https://www.thegamer.com/most-expensive-games-ever-made/.> Citation on page 27.

VALTCHANOV, V.; BROWN, J. A. Evolving dungeon crawler levels with relative placement. In: ACM. **Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering**. [S.l.], 2012. p. 27–35. Citations on pages 31 and 32.

Valve. **Left 4 Dead 2**. 2009. Accessed in: 2021-02-11. Available: <https://store.steampowered.com/app/550/Left_4_Dead_2/.> Citations on pages 50 and 51.

VIANA, B. M.; SANTOS, S. R. dos. Procedural dungeon generation: A survey. **Journal on Interactive Systems**, v. 12, n. 1, p. 83–101, 2021. Citations on pages 28, 36, and 46.

VIANA, B. M. F.; PEREIRA, L. T.; ; TOLEDO, C. F. M. Illuminating the space of dungeon maps, locked-door missions and enemy placement through map-elites. Submitted. 2022. Citation on page 61.

____. Illuminating the space of enemies through map-elites. Submitted. 2022. Citation on page 65.

Video Game History Wiki. **Action-adventure game**. 2009. Acessado em: 2020-09-11. Available: <https://videogamehistory.fandom.com/wiki/Action-adventure_game.> Citation on page 31.

WIJMAN, T. **Global Games Market to Generate $175.8 Billion in 2021; Despite a Slight Decline, the Market Is on Track to Surpass $200 Billion in 2023**. 2021. Available: <https://newzoo.com/insights/articles/global-games-market-to-generate-175-8-billion-in-2021-despite-a-slight-decline-the-market-is-on-track-to-surpass-200-billion-in-2023/>. Citation on page 27.

Wild Card Games. **Dwarf Quest**. 2012. Accessed in: 2020-07-25. Available: <https://steamcommunity.com/sharedfiles/filedetails/?id=114241031.> Citation on page 46.

YEE, N. Motivations for play in online games. **CyberPsychology & behavior**, Mary Ann Liebert, Inc. 2 Madison Avenue Larchmont, NY 10538 USA, v. 9, n. 6, p. 772–775, 2006. Citations on pages 41, 42, and 45.

YEE, N.; DUCHENEAUT, N. Gamer motivation profiling: Uses and applications. **Games User Research**, Oxford University Press, Oxford, UK, p. 485–490, 2018. Citation on page 59.

YU, D. **Spelunky**. 2008. Accessed in: 2020-08-10. Available: <https://spelunkyworld.com/.>
Citation on page 32.


ICMC
USP
SÃO CARLOS