A System for Orchestrating Multiple Procedurally Generated Content for Different Player Profiles

Leonardo Tortoro Pereira, *Graduate Student Member, IEEE*, Breno M. F. Viana, and Claudio F. M. Toledo

***Abstract*—We propose a system able to orchestrate the procedural generation of three creative facets (levels, narrative, and rules), using quality-diversity algorithms to promote diversity while converging to the desired solutions and able to satisfy different player's profiles. We aim to facilitate the project's reproducibility and extension by presenting the system's architecture and high-level design decisions and showcasing how we decoupled it enough to change algorithms. Our generation starts with a grammar-based quest generator that works as a frame for two quality-diversity algorithms: a level generator, which also creates locked-door puzzles, and an enemy generator, which selects their visuals based on their weapons. A rule-based profile analysis uses a pretest questionnaire to classify the player into four profiles, and the quest generator creates different content for each of them. We use a game prototype to test our system with over 83 players, playing over 204 levels and answering a posttest questionnaire after each. Results show the game and contents were well received, and there was a difference in perception between profiles. Players who played content based on their profile enjoyed it more than those playing content from another profile.**

***Index Terms*—Content orchestration, games, player classification, procedural content generation, software architecture.**

I. Introduction

PROCEDURAL content generation (PCG) is a technique that has become progressively more common in the game industry, whose revenue is predicted to pass the US$200 billion mark by the end of 2022.[1] A relevant example is Hades, that won two *Game of the Year* awards in 2020 and applies PCG in its levels [1]. PCG has also become a commonplace study field in the academy, giving birth to artificial intelligence algorithms and applications in serious games, e.g., for educational purposes [2], [3]. Although mainstream, the grand vision of PCG: the generation of complete games, according to Liapis et al. [4]; is still out of reach. Even further is the ability to adapt these games to different players: most PCG research until the early 20 s focused on generating specific content, and few could support different types of players. Liapis et al. [4] pointed out some of these hurdles while presenting an informative review on papers that generated more than one type of content. They also elaborate on the first set of proposals about orchestrating different creative facets.

Our work aims to expand the horizons in the orchestration of procedural content generators, presenting three contributions: a more concrete architectural example of a system orchestrating different contents; the generation of three different facets (levels, narrative, and rules); adapting these contents based on typical player archetypes. The generated facets are levels (dungeons with locked-door puzzles and the placement of enemies), narrative (in the form of quests), and rules (in the form of enemies with different attributes and mechanics). We generate said levels and enemies using a quality diversity (QD) approach.

Our system orchestrates the contents, adds postprocessing to improve quality, and injects said content into a game prototype. After answering a short questionnaire, the game selects the best matching content for players, making it an *offline* content orchestrator able to do *online* gameplay customization for different players. All these characteristics together make the system itself an advance for content orchestration research, especially in our genre of 2-D action-adventure games.

We present the related works in Section II. Section III introduces our system's architecture, offering an abstract view of the process and how each component handles and passes along the data. The implementation details on how we create and orchestrate contents and select profiles are in Section IV. Section V presents the experimental procedure to test the system and Section VI report the collected results. Finally, we write our conclusion in Section VII.

II. Related Work

The basis for this research is Liapis et al.'s review of *Orchestrating Game Generation* [4]. They define six creative facets of games: Visuals, Audio, Narrative, Rules, Levels, and Gameplay. These offer a controlled area to exploit PCG. The orchestration, i.e., harmonizing different facets to create a good quality game, is the new frontier of PCG that researchers should aim to improve.

This orchestration can be done in three different ways [4]:

Top-down: A sequential pipeline where the previous step ensures the content is ready for the game and handles the content for the next generator to add their corresponding content;

Manuscript received 13 March 2022; revised 18 July 2022; accepted 1 October 2022. Date of publication 12 October 2022; date of current version 19 March 2024. *(Corresponding author: Leonardo Tortoro Pereira.)*

This work involved human subjects or animals in its research. The author(s) confirm(s) that all human/animal subject research procedures and protocols are exempt from review board approval.

The authors are with the Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, São Paulo 13566-550, Brazil (e-mail: leonardop@usp.br; brenov@usp.br; claudio@icmc.usp.br).

Digital Object Identifier 10.1109/TG.2022.3213781

[1][Online]. Available: https://newzoo.com/insights/articles/games-market-revenues-will-pass-200-billion-for-the-first-time-in-2022-as-the-u-s-overtakes-China

2475-1510 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

Bottom-up: Each generator creates its content, compares to the others, and uses the feedback to improve toward a common goal;
In-between: Uses different proportions from both.

As for works related to the orchestration of contents, the review presented in Liapis et al. [4] covers interesting ones. The projects of *Angelina* [5], *Game-O-Matic* [6], and *Rogue Dream* [7] are the ones handling a total of four creative facets. As interesting as the reviewed works are, we raise some questions over their real-world applicability for games. They depend majorly on content scraped from the web, without a means of validation both from algorithms or human designers. Furthermore, they are somewhat limited in their flexibility to adapt to different players' and designers' needs. In general, their algorithms do not have metrics or objective functions for evaluation, nor a machine learning (ML) algorithm that somehow learned to validate good contents.

In Cook et al.'s *Angelina* [5], the system evaluates online sources' moods to select background images, sounds, and contents, placing them in a *metroidvania* game and evolving its levels. Neither content, mood, nor player preferences affect the level generator algorithm, and they evaluate the content through an algorithm, measuring only the creativity with exciting results reported.

Game-O-Matic [6] shared a similar context. It creates relationships between entities through text and uses its content to get the visuals from the internet. Predetermined rules transform the verbs into game mechanics, and postprocessing eliminates unfeasible rules. There is a simple-level generation using rules as a foundation. It also does not adapt to different players nor evaluate content with users, as it is a cocreative system.

Rogue Dream [7] used the player's name as a random seed to generate an association between objects and the visuals through Google's autocomplete and matches the results with a prescripted list of rules for mechanics, enemies, and game rules. The level generator has a more computer-driven approach and lacks user adaptation and evaluation.

AudioInSpace [8] generated audio via a Compositional Pattern Producing Networks and the position and color of players' bullets with another. Both use each other's output as feedback. The player's selection of their favorite weapon also becomes an input. Therefore, the work generates rules (bullet patterns) and visuals (bullet color) at a high level. This system allows some adaptation to different players, although limited to the player's bullets and audio contents.

The last example we take from Liapis et al. review is *Game Forge* [9]. They orchestrate two facets (narrative and levels) to reach good gameplay quality metrics that match the user's desired settings. However, they did not conduct experiments with players to validate. Our system has features from *Game Forge*, using an algorithm to generate quests as the base for the other generators.

We also use quests generated by grammar as a frame for the other generators (dungeon and enemies) to work. This approach resembles Montfort et al.'s Slant [10], which creates a story based on multiple systems working on an initial story's "blackboard." When concluded, the story goes to the Harvester, that checks the story's completion and augments it. Next, the GRIOT-Gen system realizes symbolic representations, and the final system (Curveship-Gen) selects the content (microplans) and produces the final text.

More recent research with content orchestration is Karavolos et al.'s system [11] that generates both levels and rule sets by evaluating a gameplay model trained via deep learning. They make levels, the attributes for each player class, and, through a convolutional neural network, adapt the classes' data to enable fair and balanced matches. Although not tested with players, the computational results are promising.

We believe the research in [11] to be a peer to ours, although with different goals and advancing different areas of knowledge regarding orchestration of game facets and adaptation to players. We lack a feedback system. However, our level generation evolves whole dungeons with locked-door puzzles and enemy placement, considering diverse metrics and accounting for QD. Our enemy generation is similar to their classes' generator, but we search for a set of promising solutions while covering a space defined by behavior metrics, also focusing on QD. Many recent PCG problems presented good results using QD, as in Gravina et al.'s review [12]. Moreover, we also create quests.

A fundamental pillar to harmonizing different facets in a quality game is creating an architecture that allows better developers' productivity, accounting for improved software quality. Content orchestration projects demand many functionalities to be added to the system, benefiting from the Design Stamina Hypothesis,[2] which guides the fields of software design and architecture, refactoring, test-driven development, DevOps and alike [13], [14], [15], [16].

A software design and architecture guidance may facilitate the reuse, replacement, and addition of generators to other game projects with relative ease. For example, we used a simple Evolutionary Algorithm (EA) for level generation and changed the approach to MAP-Elites without changing the data representation. Game developers often set aside this software architecture and design area, as reported in Mizutani et al.'s review on the subject [17], which, from 512 studies, found only 36 representative ones with sufficient architectural description and, between them, most were tailored to specific games or genres. The authors report studies presenting software architectures benefit from reuse, providing better systems, even if employed for a specific game application.

Although Mizutani et al.'s review [17] approached game-related research in general, PCG research is also affected. It is still rare to find discussions on what techniques, patterns, and designs the developers employed so that others could easily extend or replicate following a software architecture design. One example that does so in recent literature is Kreminski et al.'s [18], where they created an architecture for a cocreative storytelling game. Albeit in a container-level depth,[3] they present the major

[2][Online]. Available: https://martinfowler.com/bliki/DesignStaminaHypothesis.html
[3]Using C4's model classification. [Online]. Available: https://c4model.com/

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

communications between entities as well as with other containers.

We apply MAP-Elites for our level generation, used for similar purposes in Alvarez et al.'s work [19]. The authors use Interactive Constrained MAP-Elites to design a mixed-initiative tool, able to generate a matrix of rooms based on suggestions from the designer on a previous iteration. Charity et al. [20] proposed another method inspired by Constrained MAP-Elites to create rooms with mechanics based on various games, using the General Video Game Artificial Intelligence framework. In this case, the authors set the MAP-Elites matrix according to the games' mechanics, while the fitness function evaluates the agents' survival and conclusion time.

The works in [19] and [20] had enemies distributed through the levels. However, we apply another MAP-Elites to generate enemies, following Khalifa et al.'s approach [21] to create bullet hell's levels. Their method generates bullets (a type of enemy in this genre) with different speeds, damage values, movement patterns, and spawning points and places them on the level. Therefore, a single run of our algorithm allows us to place diverse enemies in a level.

Quests are game objects intended for a player that include tasks and rewards, integrating the game's narrative [22]. Our quest approach is similar to Thue et al.'s work [23] by creating quests that try to fit player profiles. However, they select the best fit quest (or encounter) from a pool of hand-made ones, branching them following events. In our case, the grammar creates a questline in run-time, prioritizing quests that match a given profile.

Gellel and Sweetser [24] created levels and quests by combining generative grammar and cellular automata. They use the grammar to codify locked-door missions and rooms, while the automata create play spaces inside the level. Our grammar structure is more related to the work of Doran and Parberry [25] whose quests have a *Motivation* (our profile). Each motivation has a strategy (our non-terminal types), which presents a sequence of actions (our terminals) with rules stated after analyzing over 750 quests from four popular RPGs.

Thus, this article aims to show advances on multifacet PCG, creating contents for different player profiles and providing system architecture insights that may enable better results and reproducibility for other researchers in this area.

III. System Architecture

Our approach is an *In-between* orchestration. It has some sequential control, where content generators interpret a high-level frame as input to create their contents with some freedom, similar to the *Creative Maestro* from Liapis et al. [4]. However, we have some facets being able to create content in parallel with a *Fake-Sheet*, a frame which generators agree to work into and expand, also from Liapis et al. [4]. Thus, we can fit the content as requested while slightly diverging. We also apply some postprocessing methods in the resulting content to improve its final quality.

To guide the generation toward the liking of different users, we divided them into different groups, or profiles, using data from the player through our Profile Analyst (PA). Currently, we generate the content before the player starts the gameplay (*offline* PCG scenario [26]), doing next an *online* selection of which generated content fits best their profile and loading such content in the game.

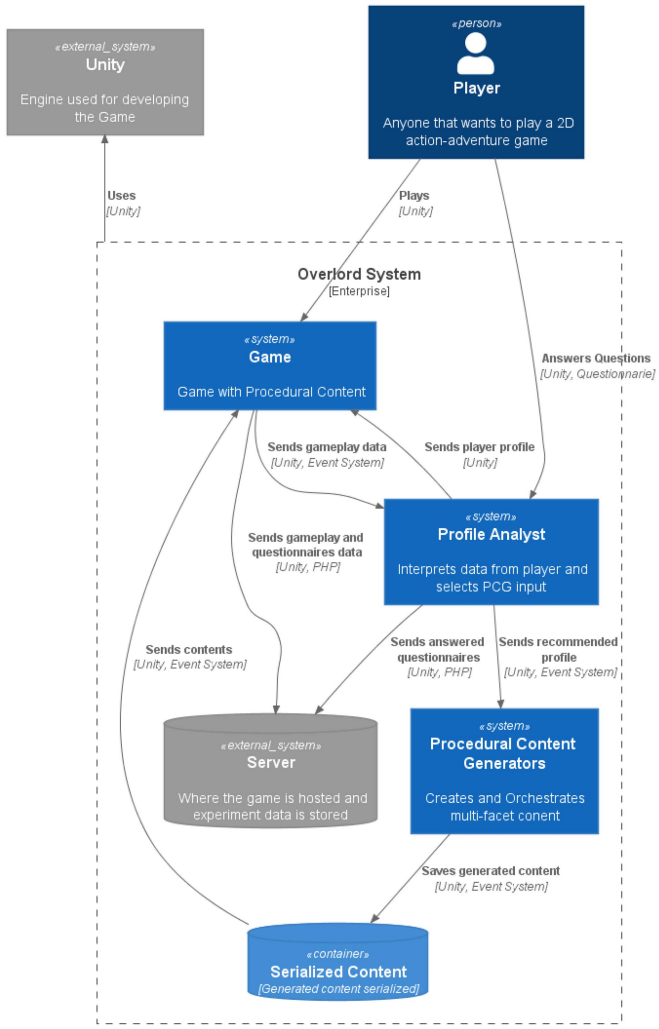

Fig. 1. System context diagram for the overlord architecture.

Fig. 1 shows the system Context diagram. The system is publicly available at GitHub,[4] with the release used for the experiments in this article in tag v0.2.0.[5] We structured our system, which we named Overlord,[6] as a collection of three different systems: the Game; the Profile Analyst; and the Procedural Content Generation. An external server and a Serialized Content container to save generated data for *offline* loading support these systems. We upload a WebGL version of the game and conduct our experiments in the server.[7]

[4][Online]. Available: https://github.com/LeonardoTPereira/Overlord-Project

[5][Online]. Available: https://github.com/LeonardoTPereira/Overlord-Project/releases/tag/v0.2.0

[6]A little wordplay as it rules over the maestro (master, or lord) and others.

[7][Online]. Available: https://overlord-pcg-en.web.app/

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

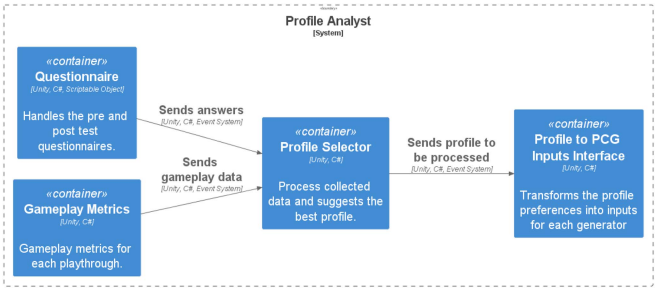

Fig. 2. Profile Analyst Container Diagram. The player answers a pretest questionnaire, the Profile Selector analyses the data, suggests a profile, and translates into inputs for the PCG System via an interface. The PA also collects metrics from the Game System and feed it to the Profile Selector container.

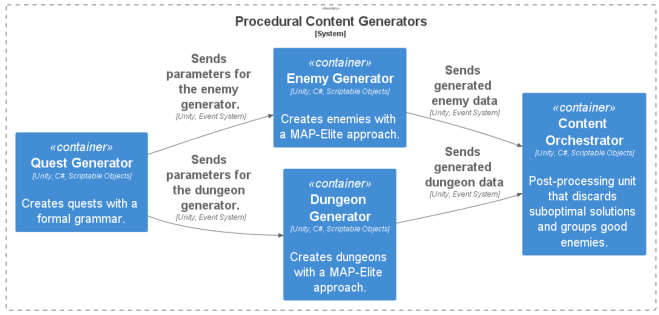

Fig. 3. PCG Container Diagram. The Quest Generator creates a quest line based on the player profile, processes the results, and passes it as input parameters for both Dungeon and Enemy generators. They create their contents and pass them to the Content Orchestrator's postprocessing unit. This unit discards undesired contents and selects each room's best enemy groups, according to the amount needed and available types.

The Game system, developed using Unity,[8] uses the content generated by the PCG system and stored in the Serialized Content container, providing fun to the player. It collects gameplay data and sends it to the Server and Profile Analyst.

The latter is a system that collects data from the player, processes it, suggests a profile, and feeds it to the PCG system. The data comes from a pretest questionnaire answered once per session, a posttest questionnaire answered after each level, and gameplay metrics sent by the Game system for each level. It also stores data on the Server. The PCG system generates different game facets procedurally and orchestrates their content. It saves the results in the Serialized Content container, so it can be used *offline* by the Game system.

In this context, we focus on separating the three systems' logic as much as possible. The main idea is to simplify modifications, additions, or replacements of a whole algorithm for another, avoiding reworking a massive portion of the code. In an experimental setting, this may allow testing more algorithms and comparing them against each other. A considerable step in such a process is to define interfaces between systems, so they have a common language to understand parameters and data without coupling. In a broad sense, this means following the Open-Closed Principle of SOLID [27]. Liapis et al. [4] make a similar suggestion when presenting the *blackboard system*.

The Profile Analyst processes the player data and sends the results to the PCG system via a data class with a constructor, which demands inputting the data needed for the PCG algorithms to work. Moreover, the PA's code requires a change only if a new PCG algorithm demands new parameters as input, which can be done via inheritance or parameter changes, requiring a new implementation for the interface in the latter. The Serialized Content container holds a similar purpose, where we store data for each content in a serialized object. In *Unity*, the *Scriptable Object*[9] is a useful data container useful for this situation. We have a *Scriptable Object* class for each content type, whose objects save the high-level data created by the generator. The game reads and processes the content into *Game Objects*, with visuals, sounds, physics, and other game-related aspects ready to be used in-game.

[8][Online]. Available: https://unity.com/

[9][Online]. Available: https://docs.unity3d.com/Manual/class-ScriptableObject.html

By creating this interface, we can use different algorithms to make the same content without changing the Game System. Moreover, the Game System can change aspects of a *Game Object* without changes in the generator. For example, a 2-D dungeon can become 3-D, but the generator only provides information about its layout and what rooms have which objects. The following sections explain how each subsystem works and the interfaces that connect them.

A. Profile Analyst System

Fig. 2 shows the Container Diagram of the PA System. It gets explicit and implicit data from the player. The former is via the Questionnaire Container, and the latter is via the Gameplay Metrics container, which receives data from the Game System. Both save their data in the Server and send them to the PA, which processes it to generate input for the PCG System, providing the contents that should fit the player's profile (Profile Selector container). In theory, this container can have a decision tree, an ML algorithm, etc. In our case, we tested it with a simple set of rules, as presented in Section V. Finally, our Profile to PCG Inputs Interface container processes the resulting data and transforms it so the PCG System can understand and manipulate it, creating the required content.

B. Procedural Content Generator System

The PCG System can generate and orchestrate different contents for the game based on the input provided by the Profile Analyst. The Serialized Content container saves the resulting contents to be loaded later by the Game System. We have three different generators, shown in their containers in Fig. 3: Quest, Dungeon, and Enemy Generator. The Quest Generator receives data from the Profile Analyst and creates a series of simple quests (questlines). The system serializes and processes the questlines to extract inputs for the Dungeon and Enemy Generator containers, using the quests as a frame for the succeeding contents.

The Dungeon Generator container creates dungeons with locked-door puzzles and defines how many enemies occupy each room. It guarantees to provide all items, enemies, and exploration needed from the originating questline. The generation

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

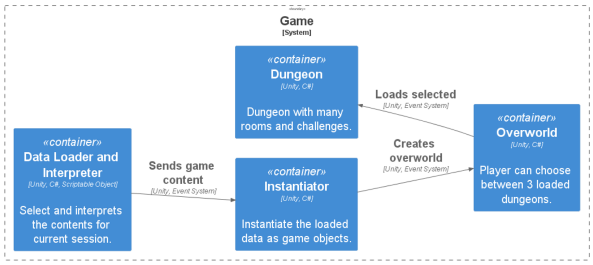

Fig. 4. Game Container Diagram. Loads the previously stored contents for the current player's profile. Processes and converts them to Game Objects with specific data and methods that the game needs, instantiates the objects, initializes and runs the game.

runs parallel to the one in the Enemy Generator container, in a *Fake-Sheets* step. The other one creates enemies with different attributes (e.g., health, damage, speed), weapons, and movement patterns. It guarantees to provide all enemy types needed for the originating questline. Our postprocessing module is the Content Orchestrator container, which works as a standalone orchestrator. It arranges which enemies (from the ones the quest-frame demands) occupy each dungeon room, based on the Enemy and Dungeon Generator contents. At last, the Serialized Content container serializes the resulting dungeon and enemies, and the generation step is over.

C. The Game System

The Game System processes the Serialized Content container's data for the current profile, selects which goes into the ongoing game session via the Profile Analyst System's information, and processes them into Game Objects for each content. The Instantiator container embodies the content when the player selects a *Dungeon* from the *Overworld*. After the player finishes a *Dungeon* (by completing or dying), the data goes to the server, and they may play again. The Data Loader and Interpreter container read the content data, e.g., how many rooms are in the *Dungeon*, their placement, and what content they have inside. It also creates Game Objects with the visual representation, sounds, and scripts, that the game demands from that content. For instance, the game can require what sprites/textures/shaders each dungeon's tile must have, their collision areas, their location in the game space, types of sound effects, etc. Fig. 4 shows the Game System container diagram.

IV. System Implementation

In this section, we discuss some details on how we implemented the containers from the systems in Section III for our specific setting on the experiment presented in this article.

A. Profile Analysis

Our Profile Selector container from the Profile Analyst system, described in Section III-A, uses a simple weighted rule to give the player one of four possible profile types based on their answers to our pretest questionnaire. They influence the quests' generation, presented in Section IV-B1, and we employ these quests as a canvas for the other generators. The first profile is

Q1 I am an experienced player;
Q2 I am an experienced player in the action-adventure genre;
Q3 In which difficulty do you usually play? (Options - Very easy, Easy, Medium, Hard, Very Hard);
Q4 I like playing games where I can explode, crush, destroy, shoot and kill;
Q5 I like playing games where I can fight using close combat skills and evade fast attacks;
Q6 I like playing games where I can explore the game world and uncover secrets and mysteries;
Q7 I explore all the places, elements, and characters of the virtual world;
Q8 I complete all quests, including those that aren't necessary to finish the game;
Q9 I like playing games where I can collect rare items and hidden treasures;
Q10 I like playing games where I can build friendships between game characters and work toward a common goal;
Q11 I like playing games where I can immerse myself in the role of the character and make meaningful decisions;
Q12 I usually only do what is necessary to pass a level or complete a quest;

Fig. 5. Pretest questionnaire.

Achievement, aimed at collectors and influencing the creation of more rooms with rewards. The second is the **C**reativity profile, which focused on explorers and generating more rooms, branched dungeons, and locked-door missions.

For the **I**mmersion profile, we focused on allowing players to explore the game with more leisure. It influences the generators to create fewer enemies, fewer locked-door missions, and fewer branching in the dungeons, but a few more rooms to visit. The last one is **M**astery, for those that seek action. It enables more enemies per room and with a more challenging difficulty. We adapted these profiles from Yee and Ducheneaut's [28] player motivation, selecting the ones that our current game best allowed players to experience.

We asked players to answer 12 questions in our pretest questionnaire. They answered them only once when starting the game to define their profile. We adapted the questions from Rivera-Villicana et al. [29], Vahlo et al. [30], and Yee and Ducheneaut's [28] works on player profiles to extract the most crucial gameplay features that each player prefers. We focused on keeping the number of questions as low as possible to increase retention. Fig. 5 shows the questions whose answers are on a 5-point Likert scale, ranging from Strongly Disagree to Strongly Agree, except for Q3.

Each profile category is a sum of the answers [as shown in (1)], ranked in descending order: the one with greater value is the profile most fitting to the player, and the one with the lesser value, the least compatible one. Based on this, we give a descending weight to each profile that will affect the chance to continue creating quests of a specific type in the Quest Generator. Moreover, the descending weight impacts the content demanded by the dungeon and enemy generators. In case of a draw, we use the order of the question enumeration, as in (1): Mastery has preference over Creativity, then Achievement and last Immersion. For Q3, we subtract three from the answer to make the medium difficulty equal to 0. In this case, easier difficulties will reduce the Mastery weight, and harder ones increase it. Q12 has an inverse weight: it penalizes all profiles, but Mastery the more the player agrees that they only like to do what's necessary (which translates to not caring enough for exploring, interacting

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

1) $S \rightarrow W|G|K|E$
2) $W \rightarrow wanderW|wander$
3) $G \rightarrow getTG|get$
4) $K \rightarrow killTK|kill$
5) $E \rightarrow exploreTE|explore$

Fig. 6. Quest generating grammar Gr.

with the world, and collecting items)

$$M = Q_3 - 3 + Q_4 + Q_5$$
$$C = Q_6 + Q_7 + 5 - Q_{12}$$
$$A = Q_8 + Q_9 + 5 - Q_{12}$$
$$I = Q_{10} + Q_{11} + 5 - Q_{12}. \quad (1)$$

Section IV-B1 explains the quest generation algorithm and how the weights affect the quest-generating grammar. For our experiment, we standardize the weights: the most fitting profile gets the weight 7, and the weights of the others are 5, 3, and 1, by descending preference. These values come from empirical experiments on balancing the number of quests for a run and the generated content for each generator. The interface provides this map of profile and weight to the PCG system.

B. Procedural Content Generation

This section presents the three PCG containers presented in our PCG system as well as works they were based upon.

1) Quest Generation—Narrative Facet: We generate quests using simple formal grammar as a proof of concept for the system's ability to create content from quests as a starting frame. These quests are based on the works of Hartsook et al. [9] and Doran and Parberry [25], selecting the ones that our current game could handle best while trying to make it representative for each profile. Its terminals consist of game mechanics related to completing the quest. The player must either *W*ander (leisurely travel the dungeon), *G*et (something), *K*ill (enemies), or *E*xplore (places and secrets). They are created through grammar Gr where $N = \{S, W, G, K, E\}$, $\Sigma = \{wander, get, kill, explore\}$. Fig. 6 presents our production rules, with S being the start symbol.

To start the grammar, we use the weights the Profile Analyst system provides, as described in Section IV-A. A loop begins for each profile weight and creates at least one quest of each type, generating the nonterminal for that quest. Each nonterminal has 50% chance of selecting its two rules; therefore, we can create a chain of quests at each iteration. Fig. 7 contains the algorithm that processes the grammar and generates quests given the profiles' weights. The system allows a maximum of four iterations of that loop per profile weight.

The Quest Generator processes the generated quests and transforms them into inputs to the Dungeon and Enemy Generator containers. For the latter, the profile dictates the difficulty level (hard, medium, and easy) based on the player's Mastery weight. Next, it saves the amount of each enemy type (e.g., slime) in a dictionary. The Quest Generator also passes the difficulty to the Enemy Generator as input (as shown in Section IV-B2) and uses

$newQuestChance, chainCost \leftarrow 0$
repeat
  for all $profileWeights > newQuestChance$ **do**
    process non-terminal of corresponding quest
  $chainCost \leftarrow chainCost + 2$
  $newQuestChance \leftarrow Random(7) + chainCost$
until $chainCost < 7$

Fig. 7. Quest-generating algorithm. The weights for the four profiles, from worst to best matching, are 1, 3, 5, and 7.

the dictionary to select and serialize the necessary amount for each type of enemy. As our Enemy Generator generates different enemies for a single type (due to the MAP-Elites implementation, described in Section IV-B2), we can have various enemies from the same class.

For the Dungeon Generator, the algorithm needs the number of rooms, keys, locks, and linearity. It also demands the number of enemies to be placed, the same as the one given to the Enemy Generator. We guarantee a minimum configuration, independent of the Quest Generator data: 16 rooms, three keys, three locks, and 1.0 for linearity. Then, we create four other discrete possible values for each Dungeon Generator input. We use discrete values for better control in the experiments; thus, a group of similar users would play very similar levels, reducing bias from the randomness. The possible values are as follows:

1) Number of rooms: 16, 20, 24, 28, 32.
2) Number of keys and locks: 3, 4, 5, 6, 7.
3) Linearity: 1.0, 1.2, 1.4, 1.6, 1.8.

Moreover, $exploration$, get, and $wander$ quests add a counter parameter that specifies linearity and the number of rooms. The $exploration$ and get quests add another counter to dictate the number of locks and keys. Therefore, the higher the counter, the more of that element.

2) Enemy Generation—Rule Facet: A previous system configuration used an EA implementation for enemy generation, presented in Pereira et al. [31]. But the flexibility of our system allowed us to change this algorithm to another with relative ease. We follow Viana et al.'s [32] MAP-Elites approach for the current Enemy Generator container, where they encode the enemies as a set of parameters: health, damage, weapon used, and projectile speed (for ranged weapons), attack speed (as well for ranged weapons), movement speed, movement pattern, duration of move cycle, duration of rest cycle.

We evolve these enemies through MAP-Elites that converge to a single input: the desired difficulty, coded as a single float input. We calculate individuals' fitness as a weighted average based on the enemies' attributes and synergy with different weapons and movement patterns. As MAP-Elites is applied, the enemies converge to the desired difficulty while attempting to preserve the diversity of having an enemy of each movement pattern for each weapon type. Other approaches could be used, like running an EA for each pair of movement and weapon combination. However, we used MAP-Elites as the algorithm's purpose is to bring said diversity while converging, which is what our problem requested. The same is valid for the dungeon generator.

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

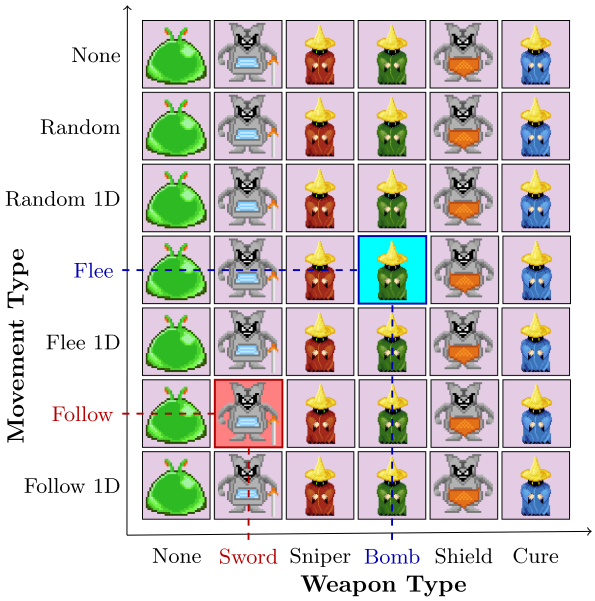

Fig. 8. Map of MAP-Elites population. The red cell represents a melee enemy that follows the player to hit with a sword. The blue cell represents a ranged enemy that flees from the player while throwing bombs toward them.

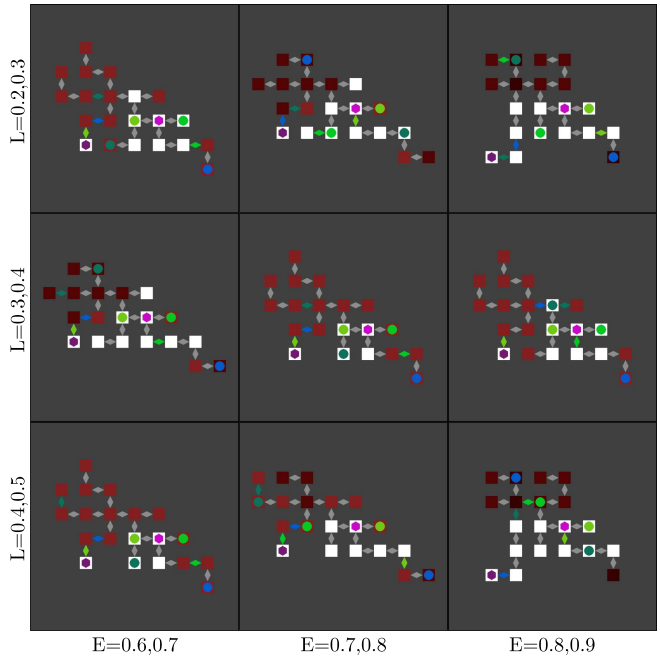

Fig. 9. 3× 3 center most MAP-Elites population of levels with 20 rooms, 4 keys, 4 locks, 30 enemies, and linear coefficient equal to 2 as input. Each cell corresponds to an Elite. Squares are rooms. The pink hexagon marks the start and the purple one, the goal. The number of enemies in a room vary with the red color: white for 0 enemies, dark red for 4 enemies (normalized for shown Elites). Circles are a key. Diamonds are corridors, and colored ones are locks with the color of the respective key.

The weapon used determines the enemy type, e.g., no weapons are slimes, and enemies that shoot simple projectiles are fire mages. We save the enemy's type and genotype values in a dictionary and use them in the Content Orchestrator and the Serialized Content containers. We present the MAP-Elites mapping and which enemy corresponds to each weapon in Fig. 8. The movement patterns consist of following or fleeing from the player, staying still, or moving in random directions each time interval. By generating all these different enemies in a single execution through QD, we guarantee enough enemies with the desired difficulty to populate our levels and keep diversity to entertain players.

As the enemy generation is not as common in PCG works as in other contents, we present our reasoning as to why we create different mechanics with the algorithm, and hence they are part of the rule facet. We follow the definition used by Liapis et al.'s work [4], which groups rules and mechanics as the *rule* facet. They made this decision based on Sicart's work [33]: "... game mechanics are methods invoked by agents, designed for interaction with the game state.... Game mechanics can be invoked by any agent, be that human or part of the computer system.... for any agent in a game, the mechanics is everything that affords agency in the game world."

3) Dungeon Generation—Level Facet: Like in the Enemy Generator container, we have used a previous EA implementation for dungeon generation on experiments with our system, presented in Pereira et al. [34]. And the system's flexibility allowed the use of a new algorithm with little integration effort. For the dungeon generation, we follow Viana et al.'s MAP-Elites approach [35]. We represent the dungeons by a tree genotype, where the rooms are nodes, and the edges are connections between rooms. Keys can be inside nodes, and locks can block edges until a specific key is found. Furthermore, the algorithm distributes the total desired enemies across nodes. They convert the tree representation into a 2-D grid containing the phenotype's dungeon structure.

The solutions converge to the game designer's desired inputs: number of rooms, locks, keys, and enemies, and desired linearity. But the MAP-Elites approach also attempts to create them with a variety in exploration (based on Liapis et al. [36]) and leniency (based on Smith et al. [37]) measures [35]. We serialize each level's width, length, and a list of rooms and corridors. For each room and corridor, we save their coordinates on the grid, a list of keys and locks it may contain/need to be opened, the number of enemies the room may have, and its type (e.g., if it is a room, a corridor, etc.). The Dungeon Generator hands this data object to the Content Orchestrator and, later, the Serialized Content containers.

Fig. 9 presents the MAP-Elites mapping for the dungeons and examples for the 3×3 centermost elites instead of all 5×5, for better visualization. As the exploration increases (dungeons in the same row, from left to right), the minimum obligatory path increases, forcing the player to explore more and as the leniency increases (dungeon in the same column, from bottom to top), the number of white rooms (no enemies) tends to increase. By generating all these different levels in a single execution through a QD approach, we guarantee to have diverse levels that attend to the same criteria for players to play during our experiments, reducing bias. In direct game applications, it allows for settings that enhance replayability, like in roguelike scenarios, where a

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

```
for all enemy in generatedEnemies do
  if movement = none and (weapon = none or weapon = sword
  or weapon = shield) then
    generatedEnemies.Remove(enemy)
for all dungeon in genereatedDungeons do
  if dungeon.Fitness > threshold then
    genereatedDungeons.Remove(dungeon)
melees ← Union(none, sword, shield)
ranged ← Union(sniper, bomb)
for all room in dungeon do
  while room.NeededEnemies > 0 do
    if room.HasMeleeOrRanged() then
      if !room.HasHealer() and healer.Count() > 0 then
        room.Enemies.Add(healer.Pop())
      else
        room.Enemies.Add(Union(melees, ranged, healer).Pop())
    else
      if Union(melees, ranged).Count() > 0 then
        room.Enemies.Add(Union(melees, ranged).Pop())
      else
        room.Enemies.Add(healer.Pop())
    room.NeededEnemies − −
```

Fig. 10. Content Orchestration algorithm.

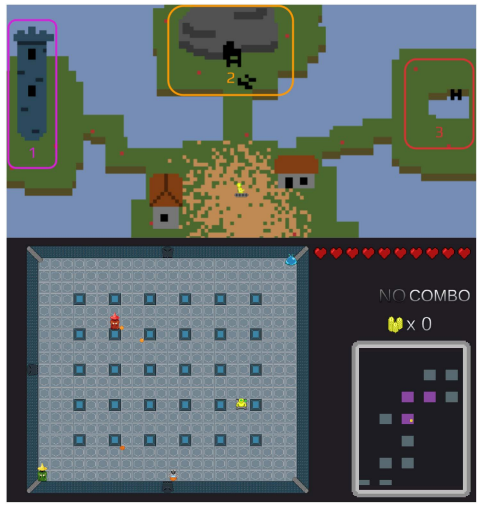

Fig. 11. Game overworld (upper) and dungeon (bottom).

player that lost may play a very similar level or create enough of them to populate a selection hub.

4) Content Orchestrator: Content Orchestrator is mainly a postprocessing unit of the contents created, altering contents following a predetermined set of rules, created after design decisions and preliminary tests conducted with a smaller sample of players. For the levels' contents, Content Orchestrator discards those with fitness values too distant from desired, based on the levels returned by the MAP-Elites. For the enemies, Content Orchestrator first removes some undesirable combinations, but the MAP-Elites algorithm does not have a fix function to remove, like melee enemies (sword, shield, and slimes) that do not move.

The next step is the placement of enemies in each room inside the level, attempting to create different gameplay combinations. First, Content Orchestrator checks the number of enemies the room must have, then tries to add healers only in rooms with two other enemies or if no other option remains. The different types have equal chances of being selected until the room is full or if the Content Orchestrator placed all needed enemies of that type.

The algorithm also places treasures, but in a deterministic approach, as follows. Behind every locked door and every leaf node of the tree, the algorithm puts some treasure until it achieves the maximum amount of treasure defined by the *get* quests. Next, the Content Orchestrator's algorithm serializes the level and enemy data, and the Game System can load them and start the game for the player. Fig. 10 presents pseudocode summarizing the orchestrator's operations.

C. Game Prototype

Our game prototype is a 2-D action-adventure game, inspired majorly by The Legend of Zelda [38] and The Binding of Isaac [39]. The player must select a dungeon from three options presented in the overworld scene, in Fig. 11. It also shows our player avatar (yellow robot) inside a dungeon room, in which there are three enemies that the player must kill.

Players must kill enemies to advance and find keys to open locked doors aiming at the dungeons' exit. As displayed in the UI, players collect coins and chain consecutive hits to a combo counter. The player has ten health points, shown in the upper-right corner. The game displays a mini-map of the dungeon in the bottom-right, highlighting already-visited rooms. We added two random-influenced details for diversity. First, each room layout comes from a poll of premade configurations. Since the enemy placement is dynamic, once creating a room, a set of spawn points avoids blocking the spawning of enemies. The other random factor is the color scheme of the dungeon. Each dungeon receives its color palette randomly from a set of premade palettes at the start.

V. Experimental Procedure

The overlord system created a collection of levels and enemies from the four different profiles described in Section IV-A. The system selects these for the players based on their profiles, chosen after the volunteers answer the pretest questionnaire in Fig. 5. We created one questline for each of the six possible combinations of preferences for each main profile. Therefore, we had six complete questlines for each of the four profiles, totaling 24. For each questline, the system saves all feasible dungeons' and enemies' contents from the MAP-Elites from Sections IV-B3 and IV-B2. Thus, there is a total of 42 possible enemies and 25 dungeons. We chose this offline approach, using a set of previously created content, to reduce bias due to variations in content generated from the same input, as our method is stochastic.

When a new session begins, the player answers the pretest questionnaire, and the Profile Analyst selects their profile. The system then randomly gets one of the six questlines for the profile. From this questline, the game randomly selects three sets of dungeon-enemy pairs from the poll, and assigns them to each of the dungeon entrances in the overworld, from Fig. 11. If the player dies in a dungeon and wants to select another, a new random dungeon-enemy combination of that questline

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

Q1 The level was fun to play;
Q2 The level was difficult to complete;
Q3 The enemies were difficult to defeat;
Q4 The challenge was just right;
Q5 The rewards were on the proper amount;
Q6 I liked the amount of exploration available on this level;
Q7 I liked the challenge of finding the keys to this level;
Q8 It was difficult to find the exit/goal of this level;

Fig. 12. Posttest questionnaire.

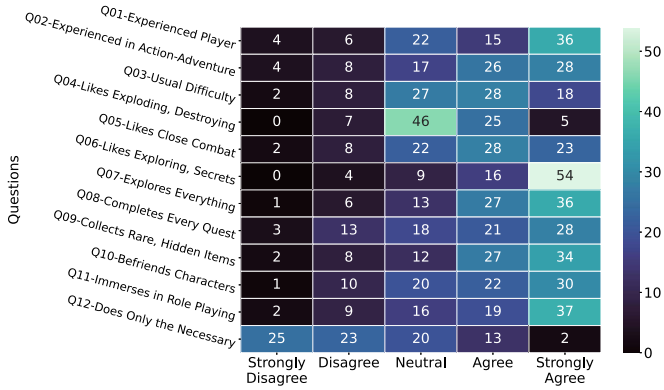

Fig. 13. Heatmap grouping the answer count for each point in the 5-point Likert scale for the 12 questions in the pre-test questionnaire.

is selected until no more dungeons are available. If the player succeeds in their adventure, completing the dungeon, or if they change dungeons after dying, we ask them to answer the posttest questionnaire from Fig. 12 and ask if they want to continue playing. If positive, the game selects a new set of three dungeon-enemy pairs to replace the ones in the overworld.

There is no repetition in the draws, so the player always plays a new dungeon for the first time and avoids bias from knowing it from a previous playthrough. We execute this loop until all dungeons are removed from the poll, or the player decides to exit the game. If no dungeons are left, a message informs the player that the game ended. Thus, we are mainly evaluating if the system can create the contents to be fun and diverse enough for players. These are grouped in four profiles, based on their answers of the pretest questionnaire. Each profile had different inputs to the content generators, that create content *offline*, before the experiment itself.

We created a WebGL Build of our game and hosted it on a public server; anyone could play anonymously, as mentioned in Section III. We invited people over social networks and email lists to play our game for two weeks. We collected no personal information, identifying players by a random ID. Therefore, we cannot guarantee that all players were different, only there were different sessions of gameplay. The system divides players into a Test Group and a Control Group. After the Profile Analyst System selects their profile, there is a 50% probability of playing levels for the actual player profile (Test Group member) or playing ones of a different profile (Control Group member).

VI. Results

A total of 83 players answered the pretest questionnaire and jointly played and evaluated (with the posttest questionnaire) 204 levels. Fig. 5 details the questions and Fig. 13 shows the bar plot of the answers. Each bar shows the number of answers for an item on the 5-point Likert scale. Most players had experience playing games in the action-adventure genre (Q1 and Q2). They played games mostly on medium difficulty, with a slight tendency for harder than easier ones (Q3).

Players were primarily neutral or liked a little the action elements on games that could drive them to a Mastery profile (Q4 and Q5). However, they generally had a greater tendency for the Creativity profile, which is about exploration (Q6 and Q7). Achievement and Immersion profiles also fit most players, showing they enjoyed collecting stuff, completing quests (the former, through Q8 and Q9), interacting with the story of the game world, their NPCs, and enjoying the world (the latter, through Q10 and Q11). The last question was negative for players, showing they mostly liked to spend time in the game world and not rush everything looking for action.

After each played level, the post-test questionnaire popped up, collecting the player's opinion about the game. As the respondents could play many levels, we had 204 responses from this questionnaire, after discarding the ones with unanswered items. We have the following answers: 30 for Achievement, 29 for Creativity, 97 for Immersion, and 48 for Mastery.

Although the pretest answers pointed out that players had an overall preference for Creativity, the ones from the Immersion profile played and answered more levels (97 in total), followed by Mastery players (48). Achievement and Creativity answered fewer (30 and 29, respectively). Probing for an expected difference in answers for each profile group, we present a violin plot for each Posttest question (from Fig. 12), grouped by the profile of the respondent, in Fig. 14. We can observe that different profiles had slightly different average answers and standard deviations for most responses.

Q1 shows that, on average, players from the Creativity group had less fun than the other groups, while the others had roughly the same. Immersion players perceived the level's difficulty (Q2) as harder, while Creativity players as easier to play. For the difficulty against enemies (Q3), the Mastery players report easier when defeating enemies, which is expected for players that prefer combat to other motivations.

Creativity and Mastery respondents felt the challenge (Q4) slightly better than Achievement and Immersion ones. Mastery players perceive the rewards (Q5) as insufficient, being such rewards more on average for Creativity and Immersion. Achievement players found a sufficient reward, possibly because they are primarily motivated by collecting rewards. Except for the Creativity one, all profiles enjoyed the exploration (Q6). Such a tendency continues when asked about the challenge of finding locks and keys (Q7). In this case, Achievement profile players also found exploration slightly less challenging than Mastery and Immersion. For the difficulty in finding the exit (Q8), Achievement players found it easier, followed by the Creativity ones, and immersion and Mastery respondents reported an average difficulty level.

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

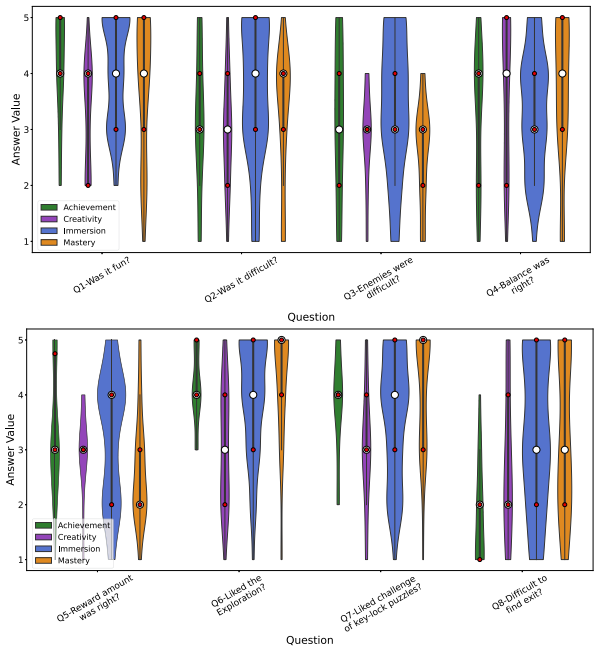

Fig. 14. Violin plot, grouping answers for each post-test question by profile. The plot width increases with the total answers for each group. The white dot is the median, red ones the quartiles.

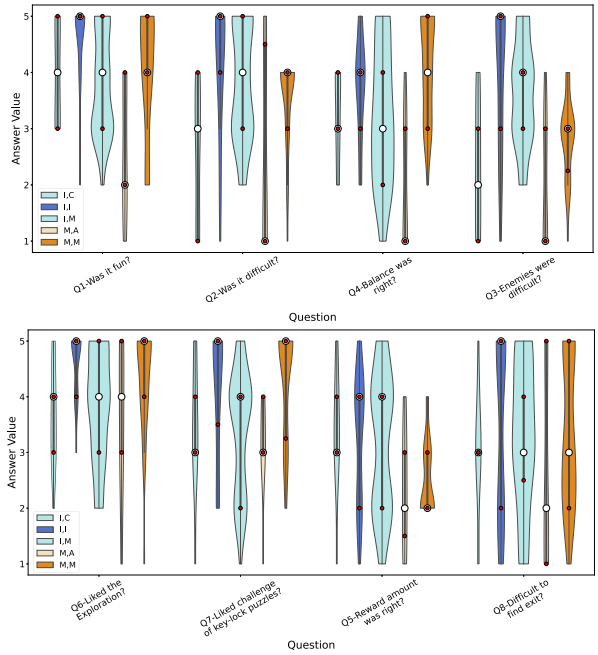

Fig. 15. Violin plot, grouping answers by the Control Group (lighter hues) and Test Group (darker hues) of Immersion and Mastery players with 10 or more answers for each posttest question. The plot width increases with the total answers. The white dot is the median, red ones the quartiles.

TABLE I
POSTTEST ANSWERS COUNT, GROUPING PLAYERS BY THE PROFILE THEY PLAYED AS VERSUS TRUE PROFILE. CONTROL GROUP IS COLORED

Played as	**(A)chievement**				**(C)reativity**			
Profile	**A**	C	I	M	A	**C**	I	M
Answers	**2**	1	7	20	5	**2**	5	17
Played as	**(I)mmersion**				**(M)astery**			
Profile	A	C	**I**	M	A	C	I	**M**
Answers	6	13	**23**	55	11	3	0	**34**

Table I shows the distribution of respondents by their profiles and the profile they played as. The Control Group is highlighted (those that played content for their accurate profile). The Achievement and Creativity players gave only two answers for the Test Group. The reduced number happens since these profiles were fewer compared to the others. Therefore, we do not have enough data to make this comparative analysis for those profiles. Instead, we analyze the answers only for the Mastery and Immersion profiles and only the groups with at least ten answers for more robust results (underscored in Table I). Fig. 15 is the same Violin Plot as Fig. 14, but now considering the player profile and the profile of the played content. A lighter hue is for the Control Group, and a darker is for the Test Group.

Test Groups had more fun playing the game than Control Groups. They also found the game more challenging (Q2), especially the enemies (Q3), but both groups perceived the exploration to find the exit as more or less the same (Q8). Although thinking the game was more difficult, the Test Group for the Immersion had a similar positive opinion about the balance as the Control Group. The Mastery profile's Test Group found it more balanced (Q4). Groups that were given content from the same profile had more or less the same opinion about the rewards (Q5). Both Test Groups liked the exploration compared to the Control Groups (Q6). They also wanted the challenge for the key-lock puzzles (Q7).

VII. CONCLUSION

After analyzing our results in Fig. 14, we can conclude that our *Overlord* system was able to create entertaining content (Median on Q1 was 4 and average above 3 for all profiles) and had an average balance on the challenge (Q4). The other questions show that the individual generators provided exciting and challenging enemies and dungeons, and the postprocessing Content Orchestrator delivered interesting content. Not only that, but the game prototype was able to represent these contents entertainingly, as the players' opinions show.

Over Fig. 15 data, we conclude that our Profile Analyst had positive results for players: in a game environment, the classification was accurate (at least for the profiles with sufficient data for the analysis). Moreover, simple profiling guiding the PCG (Test Group) resulted in players who were more entertained while playing, felt the game was more balanced, and liked the contents and the challenges compared to CG. With this data, we can also conclude that our quest generation guided the enemy and dungeon generators to create content that each profile desired.

These results also show that our system is robust enough to orchestrate the procedural generation of multiple game facets with enough diversity to please different player profiles. In addition, these results reinforce our system's flexibility, as other papers presented similar experiments with other enemy and dungeon-generating algorithms with good results. In a future work, we plan to expand the system architecture with new PCG

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.

containers (e.g., generation of puzzles, music, NPCs) and a new quest system to allow users to interact effectively with the NPCs through the quests. We also intend to enhance the Profile Analyst system by integrating an ML module by learning players' preferences while they play the game. Thus, we can provide an individual recommendation of PCG parameters and the online generation of the contents.

Acknowledgment

This work was developed using computational resources and financial support of Centro de Ciências Matemáticas Aplicadas à Indústria (CeMEAI) supported by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) - CEPID-CeMEAI (FAPESP 2013/07375-0) and also supported by Conselho Nacional de Desenvolvimento Cientìfico e Tecnològico—CNPq.

References

[1] "Hades," *Supergiant Games* (Video Game), 2020.
[2] S. Risi and J. Togelius, "Increasing generality in machine learning through procedural content generation," *Nature Mach. Intell.*, vol. 2, no. 8, pp. 428–436, 2020.
[3] L. Rodrigues and J. Brancher, "Procedurally generating a digital math game's levels: Does it impact players' in-game behavior?," *Entertainment Comput.*, vol. 32, 2019, Art. no. 100325. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1875952119300308
[4] A. Liapis, G. N. Yannakakis, M. J. Nelson, M. Preuss, and R. Bidarra, "Orchestrating game generation," *IEEE Trans. Games*, vol. 11, no. 1, pp. 48–68, Mar. 2019.
[5] M. Cook, S. Colton, and A. Pease, "Aesthetic considerations for automated platformer design," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2012, vol. 8, pp. 124–129. [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/12524
[6] M. Treanor, B. Blackford, M. Mateas, and I. Bogost, "Game-o-matic: Generating videogames that represent ideas," in *Proc. 3rd Workshop Procedural Content Gener. Games*, 2012, pp. 1–8, doi: 10.1145/2538528.2538537.
[7] M. Cook and S. Colton, "A rogue dream: Automatically generating meaningful content for games," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2014, vol. 10, pp. 2–7. [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/12745
[8] A. K. Hoover, W. Cachia, A. Liapis, and G. N. Yannakakis, "Audioinspace: Exploring the creative fusion of generative audio, visuals and gameplay," in *Proc. Int. Conf. Evolutionary Biologically Inspired Music Art*, 2015, pp. 101–112.
[9] K. Hartsook, A. Zook, S. Das, and M. O. Riedl, "Toward supporting stories with procedurally generated game worlds," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011, pp. 297–304.
[10] N. Montfort, R. P. y Pérez, D. F. Harrell, and A. Campana, "Slant: A blackboard system to generate plot, figuration, and narrative discourse aspects of stories," in *Proc. Int. Conf. Commun. China*, 2013, pp. 168–175.
[11] D. Karavolos, A. Liapis, and G. Yannakakis, "A multifaceted surrogate model for search-based procedural content generation," *IEEE Trans. Games*, vol. 13, no. 1, pp. 11–22, Mar. 2021.
[12] D. Gravina, A. Khalifa, A. Liapis, J. Togelius, and G. N. Yannakakis, "Procedural content generation through quality diversity," in *Proc. IEEE Conf. Games*, 2019, pp. 1–8.
[13] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley Professional, 2018.
[14] R. C. Martin, J. Grenning, S. Brown, K. Henney, and J. Gorman, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Hoboken, NJ, USA: Prentice Hall, 2018.
[15] K. Beck, *Test-Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Professional, 2003.
[16] G. Kim, J. Humble, P. Debois, J. Willis, and N. Forsgren, *The DevOps handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. Portland, USA: IT Revolution, 2021.
[17] W. K. Mizutani, V. K. Daros, and F. Kon, "Software architecture for digital game mechanics: A systematic literature review," *Entertainment Comput.*, vol. 38, 2021, Art. no. 100421.
[18] M. Kreminski, M. Dickinson, M. Mateas, and N. Wardrip-Fruin, "Why are we like this?: The AI architecture of a co-creative storytelling game," in *Proc. Int. Conf. Found. Digit. Games*, New York, NY, USA, 2020, pp. 1–4, doi: 10.1145/3402942.3402953.
[19] A. Alvarez, S. Dahlskog, J. Font, and J. Togelius, "Empowering quality diversity in dungeon design with interactive constrained map-elites," in *Proc. IEEE Conf. Games*, 2019, pp. 1–8.
[20] M. Charity, M. C. Green, A. Khalifa, and J. Togelius, "Mech-elites: Illuminating the mechanic space of GVG-AI," in *Proc. Int. Conf. Found. Digit. Games*, New York, NY, USA, 2020, pp. 1–10, doi: 10.1145/3402942.3402954.
[21] A. Khalifa, S. Lee, A. Nealen, and J. Togelius, "Talakat: Bullet hell generation through constrained map-elites," in *Proc. Genet. Evol. Comput. Conf.*, 2018, pp. 1047–1054.
[22] K. K. Yu, M. Guzdial, and N. Sturtevant, "The definition-context-purpose paradigm and other insights from industry professionals about the definition of a quest," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2021, vol. 17, pp. 107–114. [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/18897
[23] D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen, "Interactive storytelling: A player modelling approach," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, Sep. 2021, vol. 3, pp. 43–48. [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/18780
[24] A. Gellel and P. Sweetser, "A hybrid approach to procedural generation of roguelike video game levels," in *Proc. Int. Conf. Found. Digit. Games*, 2020, pp. 1–10.
[25] J. Doran and I. Parberry, "A prototype quest generator based on a structural analysis of quests from four mmorpgs," in *Proc. 2nd Int. Workshop Procedural Content Gener. Games*, New York, NY, USA, 2011, pp. 1–8, doi: 10.1145/2000919.2000920.
[26] J. Togelius, N. Shaker, and M. J. Nelson, *Introduction in Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. N. Shaker, J. Togelius, and M. J. Nelson, Eds., Amsterdam, The Netherlands: Springer, 2016, pp. 1–15.
[27] R. C. Martin, J. Newkirk, and R. S. Koss, *Agile Software Development: Principles, Patterns, and Practices*, vol. 2, Hoboken, NJ, USA: Prentice Hall, 2003.
[28] N. Yee and N. Ducheneaut, "Gamer motivation profiling: Uses and applications," *Games User Res.*, Oxford University Press, pp. 485–490, 2018, doi: 10.1093/oso/9780198794844.003.0028.
[29] J. Rivera-Villicana, F. Zambetta, J. Harland, and M. Berry, "Informing a BDI player model for an interactive narrative," in *Proc. Annu. Symp. Comput.-Human Interaction Play*, 2018, pp. 417–428, doi: 10.1145/3242671.3242700.
[30] J. Vahlo, J. Kaakinen, S. Holm, and A. Koponen, "Digital game dynamics preferences and player types: Preferences in game dynamics," *J. Comput.-Mediated Commun.*, vol. 2, pp. 88–103, 2017.
[31] L. T. Pereira, B. M. F. Viana, and C. F. M. Toledo, "Procedural enemy generation through parallel evolutionary algorithm," in *Proc. 20th Braz. Symp. Comput. Games Digit. Entertainment*, 2021, pp. 126–135.
[32] B. M. F. Viana, L. T. Pereira, and C. F. M. Toledo, "Illuminating the space of enemies through map-elites," in *Proc. IEEE Conf. Games (CoG)*, 2022, pp. 17–24, doi: 10.1109/CoG51982.2022.9893621.
[33] M. Sicart, "Defining game mechanics," *Game Stud.*, vol. 8, no. 2, pp. 1–14, 2008.
[34] L. T. Pereira, P. V. de Souza Prado, R. M. Lopes, and C. F. M. Toledo, "Procedural generation of dungeons' maps and locked-door missions through an evolutionary algorithm validated with players," *Expert Syst. Appl.*, vol. 180, 2021, Art. no. 115009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421004504
[35] B. M. F. Viana, L. T. Pereira, and C. F. M. Toledo, "Illuminating the space of dungeon maps, locked-door missions and enemy placement through map-elites," 2022, doi: 10.48550/ARXIV.2202.09301.
[36] A. Liapis, G. Yannakakis, and J. Togelius, "Towards a generic method of evaluating game levels," *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, vol. 9, no. 1, pp. 30–36, 2013.
[37] T. Smith, J. Padget, and A. Vidler, "Graph-based generation of action-adventure dungeon levels using answer set programming," in *Proc. 13th Int. Conf. Found. Digit. Games*. 2018, pp. 1–10.
[38] *The Legend of Zelda* (Video Game). Nintendo R&D4, 1986.
[39] E. McMillen and F. Himsl, *The Binding of Isaac* (Video Game). 2011.

Authorized licensed use limited to: Malta College of ArtsScience&Tech. Downloaded on May 05,2025 at 09:30:20 UTC from IEEE Xplore. Restrictions apply.