UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

# Architecture and Development of a Real-Time Multiple Content Generator System for Video Games

**Leonardo Tórtoro Pereira**

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

ICMC USP
SÃO CARLOS

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: ______________________

**Leonardo Tórtoro Pereira**

# Architecture and Development of a Real-Time Multiple Content Generator System for Video Games

Thesis 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 Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Claudio Fabiano Motta Toledo

**USP – São Carlos**
**February 2023**

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

P436a

Pereira, Leonardo Tortoro
Architecture and development of a real-time multiple content generator system for video games / Leonardo Tortoro Pereira; orientador Claudio Fabiano Motta Toledo. -- São Carlos, 2023.
212 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2023.

1. Realtime Procedural Content Generation. 2. Software Architecture. 3. Evolutionary Algorithm. 4. Formal Grammar. 5. Content Adaptation. 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

**Leonardo Tórtoro Pereira**

# Arquitetura e Desenvolvimento de um Sistema de Geração Procedural de Múltiplos Conteúdos para Jogos Eletrônicos em Tempo Real

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor 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**
**Fevereiro de 2023**

*This work is dedicated to my family, advisor, girlfriend and friends, who always helped me to become a better person.*
*And for all of those who thirst for knowledge and fight for science.*

# ACKNOWLEDGEMENTS

The main thanks are to my advisor, Claudio Fabiano Motta Toledo, who has been a friend and inspiration since my early undergraduate years. And also to my parents, who always supported me in every step of my academic career.

I would also like to thank the other students from our research group, specially those that helped us develop this project: Breno Maurício de Freitas Viana, who assisted in the development of the MAP-Elites generator of both dungeons and enemies; Luiza Torello Vieira, who aided in the development of both grammar for quest generation; Luana Terra do Couto and Paolo Victor Hitoshi Scassa, who worked in the initial formulation of the quest generator and its early development; Ângelo Antônio Bertoli Guido, who assisted in the development of the room generator.

Furthermore, I would like to thank everyone who has participated in our experiments and supported us during this research.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - PROEX-7573621/D. This study was also financed in part by The Brazilian National Council for Scientific and Technological Development (CNPq), grant 141857/2019-6.

*"It's not about whether it's impossible or not,*
*I'm doing it because I want to."*
*(Eiichiro Oda, One Piece)*

# RESUMO

PEREIRA, L. T. **Arquitetura e Desenvolvimento de um Sistema de Geração Procedural de Múltiplos Conteúdos para Jogos Eletrônicos em Tempo Real**. 2023. 212 p. Thesis (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, Brazil, 2023.

É apresentado um sistema capaz de orquestrar a geração procedural de múltiplos conteúdos, usando diferentes técnicas de criatividade computacional, e capaz de gerar os conteúdos em tempo real. Os conteúdos gerados são calabouços, o posicionamento de inimigos, chaves e fechaduras em suas salas, os inimigos que serão colocados no calabouço, missões para o jogador completar, e as salas do calabouço. São usados algoritmos evolutivos, MAP-Elites, Gramáticas Formais, Cadeias de Markov e Autômatos Celulares para criar tais conteúdos. Eles são organizados por um orquestrador e colocados em um protótipo de jogo. No último estágio do sistema, o conteúdo pode ser gerado em tempo real, tão logo quanto são coletados os dados do jogador para usar de entrada, através de uma análise de perfis baseada em regras. Também é descrita uma arquitetura e design de sistema, que permitiram a adição de novos geradores e a troca de algoritmos para cada gerador com certa facilidade. Diferentes experimentos com jogadores foram realizados para testar diferentes algoritmos e configurações do sistema, todos com resultados positivos, e com os jogadores divertindo-se ao jogar os conteúdos resultantes. Jogadores que jogaram o conteúdo baseado em seus perfis divertiram-se mais do que aqueles que jogaram conteúdo de outro perfil. Isso mostra que nosso identificador de perfis com base em regras pode guiar os geradores para bons resultados.

**Palavras-chave:** Geração Procedural de Conteúdo, Arquitetura de Software, Design de Sistemas, Algoritmo Evolutivo, Geração de Conteúdo em Tempo Real.

# ABSTRACT

PEREIRA, L. T. **Architecture and Development of a Real-Time Multiple Content Generator System for Video Games**. 2023. 212 p. Thesis (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, Brazil, 2023.

We present a system able to orchestrate the procedural generation of multiple contents, using different computational creativity techniques, and able to generate the contents in real time. The generated contents are the dungeons, the placement of enemies, locks and keys in its rooms, the enemies that will be placed in said dungeons, quests for the player to complete, and the dungeon's rooms. We use evolutionary algorithms, MAP-Elites, Formal Grammars, Markov Chains and Cellular Automata to create these contents. They are organized by an orchestrator and placed in a game prototype. In the late stage of the system, the content can be generated in real time, as soon as it collects data from the player to use as input, using a rule-based profile analysis. We also describe our architecture and system design, which allowed us to add new generators and change the algorithms for each generator with relative easy. Different experiments with players were made to test different algorithms and system's setups, all with positive results, and the players having fun playing the resulting contents. Players who played content based on their profile enjoyed it more than those playing content from another profile. Showing that our rule-based profiler can guide the generators to good results.

**Keywords:** Procedural Content Generation, Software Architecture, System Design, Evolutionary Algorithm, Real-Time Content Generation.

# LIST OF FIGURES

Figure 1 – *The Legend of Zelda: Breath of the Wild* (top) and *Horizon Zero Dawn* (bottom), two award-winning action-adventure games released in 2017 . . . 41

Figure 2 – Graph abstraction of 2 levels from *The Legend of Zelda: The Minish Cap* game. Only rooms containing items which are essential for the player's progression are shown. Each color identifies a different type of key-lock pair. Keys are represented as diamonds and locks as squares. The vertical lines show a sequence of rooms in the same path, while the horizontal ones identify a path's branching. Many rooms in the same horizontal line may be visited by players, if they wish so. Extracted from: <https://www.youtube.com/watch?v=KEVJXqV7XMc> . . . . . . . . . . . . . . . . . . . . . . . 43

Figure 3 – A possible solution for the puzzle in the upper dungeon from Figure 2. In the upper left image, the first key is found and used in the first lock. In the upper right one, two levers are used to open the green door. In the lower left image, the player finds the bow and opens the first door requiring its use. At last, in the lower right image, the player gets the boss key and opens the last lock. Extracted from: <https://www.youtube.com/watch?v=KEVJXqV7XMc> . . 44

Figure 4 – General representation of an Evolutionary Algorithm. Extracted from: (EIBEN; SMITH, 2003) (p. 17). . . . . . . . . . . . . . . . . . . . . . . . . . . 49

Figure 5 – A* execution example. Extracted from: (LESTER, 2005). . . . . . . . . . 53

Figure 6 – Markov Chain example from a hypothetical market as seen in <https://en.wikipedia.org/wiki/Examples_of_Markov_chains>. The directed graph's nodes are states of the stock market in a given week, and the edges are the probability of what the next week will be. The sum of the edges' weights exiting a given node is 1.0. . . . . . . . . . . . . . . . . . . . . . . . . . 55

Figure 7 – Example of the rewriting system $M = (\Sigma, R)$, $\Sigma = \{a,b,x,y\}$ and $R = \{r_1 : a \to x, r_2 : b \to y\}$ translating the string “aab”. . . . . . . . . . . . . . . . . . 56

Figure 8 – A system context diagram from a bank system, as shown in C4 model's website <https://c4model.com/>. The Internet Banking System is the system of interest, while the customer is the main actor, but the banking system also interacts with the E-mail System and the Mainframe Banking System. . . . 58

Figure 9 – A container diagram from a bank system, as shown in C4 model's website <https://c4model.com/>. The system consists of a web application, visited by the user, a single-page application with the functionalities, or a mobile app that the user may also visit. Both applications call the API Application, which sends emails using the E-mail System, makes API calls to the Mainframe Banking System, and uses data from the Database. . . . . . . . . . . . . . . . 59

Figure 10 – A component diagram from a bank system's API Application, as shown in C4 model's website <https://c4model.com/>. The container consists of a Sign In Controller, which uses data from the mobile or single page Applications to sign in a user. The Security Component check the data, as well as data from the Reset Passwords Controller component, which also gets data from both applications. The Reset Password Controller also uses the E-mail Component to send e-mail to users. The applications can make API calls to the Accounts Summary Controller component, which, in turn, uses the Mainframe Banking System Facade, that calls the Mainframe Banking System. . . . . . . . . . . 59

Figure 11 – A code diagram (UML's class diagram) for the Mainframe Banking System Facade class, as shown in C4 model's website <https://c4model.com/>. The class has an implementation (which shows it is an interface, and not a class). This implementation may throw exceptions, may parse the balance response from the banking system, may create a balance request, and, finally, use the banking system connection. . . . . . . . . . . . . . . . . . . . . . . . . . . 60

Figure 12 – Bartle player types chart. Source: <https://en.wikipedia.org/wiki/File:Character_theory_chart.svg> . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

Figure 13 – PCG works which generate multiple creative facets. As in (Liapis *et al.*, 2019). 80

Figure 14 – AI-Based Game Design Patterns. As in (TREANOR *et al.*, 2015a). . . . . . 84

Figure 15 – PCG-Based Game Design Patterns. As in (COOK *et al.*, 2016). . . . . . . . 85

Figure 16 – System context diagram for the *Overlord* architecture, considering the experiments with offline content generation. . . . . . . . . . . . . . . . . . . . . . 91

Figure 17 – System context diagram for the *Overlord* architecture, considering the experiments with online content generation. . . . . . . . . . . . . . . . . . . . . . 92

Figure 18 – *Profile Analyst* container diagram. The player answers a pre-test questionnaire, the *Profile Selector* analyses the data, suggests a profile and translates into inputs for the *PCG* system via an interface. The *Profile Analyst* also collects metrics from the *Game* system and feed it to the *Profile Selector* container. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93

Figure 19 – *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 post-processing unit. This unit discards undesired contents and selects each room's best enemy groups, according to the amount needed and available types. A *Room Generator* container creates the rooms as demanded by the dungeon. . . . . . . . . . . . . . . . . . . . . 94
Figure 20 – *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. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95
Figure 21 – Highlight of the Procedural Content Generator System inside the Overlord Context Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
Figure 22 – Genotype-phenotype translation. Dungeon Parameters: 20 Rooms, 4 Keys, 4 Locks and 1.5 Linear Coefficient . . . . . . . . . . . . . . . . . . . . . . 99
Figure 23 – Next step of genotype-phenotype translation . . . . . . . . . . . . . . . . . . 100
Figure 24 – Example of overlapping nodes and a fixed tree. . . . . . . . . . . . . . . . . . 101
Figure 25 – EA – As seen in (PEREIRA *et al.*, 2021) . . . . . . . . . . . . . . . . . . . 102
Figure 26 – Crossover, as seen in (PEREIRA *et al.*, 2021), but accounting for enemies . 103
Figure 27 – 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 besides the red one, and it also has more rooms with enemies. . . . . . . . . . . . . . . . . . . . . . . . . . . . 109
Figure 28 – Flowchart with an overview of our approach. Pop abbreviates population. . 110
Figure 29 – Example of a MAP-Elites population evolved for levels with 20 rooms, 4 keys, 4 locks, 30 enemies, and linear coefficient equal to 2. 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. . . . . . . . . . . . . . . . . . . . . . . 111
Figure 30 – List of enemies of our game prototype. Slimes have no weapon. Swordsmen use swords. Bower mages shoot arrows. Bomber mages throw bombs. Shieldmen hold shields. Healers use cure spell to heal other enemies. . . . . 115
Figure 31 – 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. . . . 116

Figure 32 – The map of MAP-Elites population for the online experimental setup. 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. . . . . . . . . . . . . . . . . . . . . . . . . 119
Figure 33 – Quest generating grammar *Gr*. . . . . . . . . . . . . . . . . . . . . . . . 120
Figure 34 – Quest generating grammar $Gr_2$. . . . . . . . . . . . . . . . . . . . . . . . 122
Figure 35 – All achievable derivations. . . . . . . . . . . . . . . . . . . . . . . . . . . 122
Figure 36 – Example of rooms generated with fixed patterns. . . . . . . . . . . . . . . . 124
Figure 37 – Cross-shaped structuring element matrix applied for the erosion. . . . . . . 125
Figure 38 – Example of rooms generated with the Cellular Automata-based algorithm. . 126
Figure 39 – Workflow of the PCG system, a part of the diagram shown in Figure 59. After receiving the weight of the player's preference for each profile, the generation starts and proceeds to generate all contents, send to the Game System at the end. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 128
Figure 40 – A possible solution of the orchestrator on how to divide the enemies, items and NPCs through the dungeon's rooms . . . . . . . . . . . . . . . . . . . . 131
Figure 41 – Highlight of the Game System inside the Overlord Context Diagram . . . . 131
Figure 42 – Screenshots showing differences from the early version of the game from the most current version. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132
Figure 43 – Screenshots showing some major gameplay elements. . . . . . . . . . . . . 133
Figure 44 – Mapping of each game's action to their corresponding key in the keyboard. . 134
Figure 45 – Map of a dungeon as shown in the game's UI when in the full-screen map mode. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 135
Figure 46 – Menu that shows the completed and currently open quests for the player. . . 135
Figure 47 – Dialogue UI, where an NPC is requesting the player to kill a certain number of enemies from a specific type. . . . . . . . . . . . . . . . . . . . . . . . . 136
Figure 48 – A red skeleton, an enemy, in a room with the player. . . . . . . . . . . . . . 137
Figure 49 – An NPC is requesting the player to collect some items and trade for another, with a fellow NPC. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
Figure 50 – An NPC is congratulating the player for completing an exchange quest. When the dialogue appears on the screen, an event is broadcast and the quest controller, which is listening, closes the quest. . . . . . . . . . . . . . . . . . 140
Figure 51 – An NPC is exchanging items with the player, after all necessary items were collected. When the dialogue appears on-screen, an event is broadcast and the quest controller, which is listening, completes the quest. The inventory controller and inventory UI also listens to the event, updating their data accordingly. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 141

Figure 52 – An example of a dialogue inside the game. The NPC portrait and background color varies according to the speaker. The dialogue can be divided in different lines, which are displayed one after the other, after the player press the "confirm" button (X key). . . . . . . . . . . . . . . . . . . . . . . . . . . 142
Figure 53 – A room marked in red showing where the player must go. This mark is made after an event is called by the dialogue controller when an opening dialogue for a *go somewhere* quest is spoken. . . . . . . . . . . . . . . . . . . . . . . 143
Figure 54 – Enemies' spawn points represented as yellow circles. . . . . . . . . . . . . 144
Figure 55 – Highlight of the Profile Analyst System inside the Overlord Context Diagram 144
Figure 56 – Example of a pre-test questionnaire inside the Unity engine, as the players in our experiments has seen and responded. They may click in one of the radio buttons, indicating their agreement to the statement from 1 to 5, or ignore, which will save the question as not-answered, or 0. . . . . . . . . . . . . . . . 145
Figure 57 – Example of a post-test questionnaire inside the Unity engine, as the players in our experiments has seen and responded. They may click in one of the radio buttons, indicating their agreement to the statement from 1 to 5, or ignore, which will save the question as not-answered, or 0. . . . . . . . . . . . . . . . 145
Figure 58 – Example of data collected from a player's gameplay in a dungeon named as *Classic* in the Firestore Cloud database. The picture shows the given weight the player scored for each profile, how they answered the pre-test, and the number of rooms they entered in that dungeon. . . . . . . . . . . . . . . . . . 146
Figure 59 – An overview of our system, showing the flow of data during its execution and summarizing our methodology. The PCG strategy for each content may vary according to the experiment, as well as when the content is generated (offline vs. online), but the structure is the same for all experiments. . . . . . . . . 148
Figure 60 – Data workflow for the experimental setup about the enemy generator EA. . . 151
Figure 61 – Parallel coordinates visualization of the numeric attributes for the top 10 best enemies for each difficulty setting, used in the experiments with players. The charts present normalized parameters. . . . . . . . . . . . . . . . . . . . . . 153
Figure 62 – Experience questionnaire on a 5-point Likert scale. Q1 through Q3 are answered only once per player. . . . . . . . . . . . . . . . . . . . . . . . . 154
Figure 63 – Total number of players by difficulty setting. . . . . . . . . . . . . . . . . . 155
Figure 64 – Results from the demographic questions. . . . . . . . . . . . . . . . . . . . 156
Figure 65 – Results from how fun the game was for each difficulty. . . . . . . . . . . . 157
Figure 66 – Results from the questions about the combat difficulty (left column) and difficulty adequacy (right column) for each setting. . . . . . . . . . . . . . . . 158

Figure 67 – Total attempts players gave to each difficulty, how many times they succeeded and how many times they failed. Each series is normalized by the number of players on said difficulty: 10 for the easy, 11 for the medium and 7 for the hard, as shown in Figure 63. . . . . . . . . . . . . . . . . . . . . . . . . 159
Figure 68 – Data workflow for the offline experimental setup. . . . . . . . . . . . . . . 160
Figure 69 – Game overworld (upper) and dungeon (bottom). . . . . . . . . . . . . . . . 161
Figure 70 – Pre-Test Questionnaire. . . . . . . . . . . . . . . . . . . . . . . . . . . . 162
Figure 71 – Post-Test Questionnaire for the Offline Experiment. . . . . . . . . . . . . . 163
Figure 72 – Heatmap grouping the answer count for each point in the 5-point Likert scale for the 12 questions in the pre-test questionnaire. . . . . . . . . . . . . . . . 164
Figure 73 – 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. . . . . . . . . . . . . . . . . . . . . . . . . 166
Figure 74 – Data workflow for the online experimental setup. . . . . . . . . . . . . . . 169
Figure 75 – Pre-Test questionnaire for the online experiment. The questions are directly translated to the player's preference for the achievement, creativity, immersion and mastery profiles, respectively. . . . . . . . . . . . . . . . . . . . . . 170
Figure 76 – Loading screen for the real-time level generation game prototype. The player can select any weapon they wish, and play after the content is created. . . . 170
Figure 77 – Loading screen for the real-time level generation game prototype. The player can select any weapon they wish, and play after the content is created. . . . 171
Figure 78 – Post-test questionnaire for the online experiment, answered after completing (or giving up) each level. . . . . . . . . . . . . . . . . . . . . . . . . . . 179
Figure 79 – Heatmap showing the mean value of each profile's answer to Q1 of the post-test questionnaire: how much fun they felt playing each dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 180
Figure 80 – Heatmap showing the mean value of each profile's answer to Q2 of the post-test questionnaire: how difficult was the dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . 181
Figure 81 – Heatmap showing the mean value of each profile's answer to Q3 of the post-test questionnaire: how much challenge they felt playing each dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 181
Figure 82 – Heatmap showing the mean value of each profile's answer to Q4 of the post-test questionnaire: how difficult were the enemies in each dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 182

Figure 83 – Heatmap showing the mean value of each profile's answer to Q5 of the post-test questionnaire: how difficult was to navigate through each dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 183
Figure 84 – Heatmap showing the mean value of each profile's answer to Q6 of the post-test questionnaire: how good they felt was the dungeon size. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. 183
Figure 85 – Heatmap showing the mean value of each profile's answer to Q7 of the post-test questionnaire: how good they felt was the dungeon's exploration. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 184
Figure 86 – Heatmap showing the mean value of each profile's answer to Q8 of the post-test questionnaire: how good they felt was the challenge to open the locked doors. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . . . . . . . . . . . . . . . . . . . . 184
Figure 87 – Heatmap showing the mean value of each profile's answer to Q9 of the post-test questionnaire: how good they felt was the rewards (items) collected. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 185
Figure 88 – Heatmap showing the mean value of each profile's answer to Q10 of the post-test questionnaire: how much fun they had completing the quests. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 186
Figure 89 – Heatmap showing the mean value of each profile's answer to Q11 of the post-test questionnaire: how much they felt the quests were designed by humans. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon. . . . . . . . . . . . . . . . . . . . . . . . 186
Figure 90 – Heatmap plot, grouping the success rate for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator. 187
Figure 91 – Heatmap plot, grouping the percentage of enemies killed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator. . . . . . . . . . . . . . . . . . . . . . . . . . . . 187
Figure 92 – Heatmap plot, grouping the percentage of map completion for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator. . . . . . . . . . . . . . . . . . . . . . . . . . . . 188
Figure 93 – Heatmap plot, grouping the percentage of mastery quests completed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator. . . . . . . . . . . . . . . . . . . . . . . . . . . 188

Figure 94 – Heatmap plot, grouping the percentage of achievement quests completed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator. . . . . . . . . . . . . . . . . . . . . . . . . . 189
Figure 95 – Heatmap plot, grouping the percentage of creativity quests completed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator. . . . . . . . . . . . . . . . . . . . . . . . . . 190
Figure 96 – Heatmap plot, grouping the percentage of immersion quests completed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator. . . . . . . . . . . . . . . . . . . . . . . . . . 190
Figure 97 – 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 post-test question. The plot width increases with the total answers. The white dot is the median, red ones the quartiles. . . . . . . . . 212

# LIST OF ALGORITHMS

Algorithm 1 – Basic Evolutionary Algorithm. Adapted from: (EIBEN; SMITH, 2003) (p. 16) . . . . . 49
Algorithm 2 – MAP-Elites Algorithm, as seen in (CHATZILYGEROUDIS *et al.*, 2021) 51
Algorithm 3 – Recursive DFS. Adapted from: (CORMEN, 2008) . . . . . 54
Algorithm 4 – Tree Structure-generation algorithm (PEREIRA *et al.*, 2021) . . . . . 98
Algorithm 5 – Quest-generating algorithm. The weights for the four profiles, from worst to best matching, are 1, 3, 5, and 7. . . . . . 120
Algorithm 6 – Derivation algorithm. . . . . . 123
Algorithm 7 – Algorithm that selects the next derived symbol. This function is invoked as *SetNextSymbol()* in Algorithm 6. . . . . . 123
Algorithm 8 – Room-generating algorithm. . . . . . 125
Algorithm 9 – Content Orchestration algorithm. . . . . . 127
Algorithm 10 – Conditions to transform the player's mastery weight to a numeric difficulty value for the enemy generator algorithm. . . . . . 130

# LIST OF TABLES


Table 1 – Comparison between our work and the closely related ones in reviewed literature about enemy generation. . . . . . . . . . . . . . . . . . . . . . . . . . 74

Table 2 – Comparison between our work and the closely related ones in reviewed literature. The creative facets have been abbreviated to their first letters to fit the page, in the following order: **V**isuals, **A**udio, **N**arrative, **L**evel, **R**ules and **G**ameplay. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

Table 3 – Parameters, variable type, range, and their descriptions for the enemy's genotype in the EA. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 113

Table 4 – Each movement type that the enemy can have, its weight in the fitness function and the working details. . . . . . . . . . . . . . . . . . . . . . . . . . . 114

Table 5 – Each weapon type available, its weight in the fitness function and the details. The projectiles' weights(*) are lower as they are multiplied by the projectile speed and attack speed. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 114

Table 6 – Data collected from averaging 200 executions of the EA with different input parameters. The input is abbreviated with D-P-G, D is the difficulty, P is the population size, and G is the number of generations. Regarding difficulty, E, M and H represents Easy = 14, Medium = 17.5, and Hard = 22.5 difficulties, respectively. Finally, *AVG Best* and *STD Best* are collected from the average fitness of the 20 best individuals. . . . . . . . . . . . . . . . . . . . . . . . . . 152

Table 7 – Post-test answers count, grouping players by the profile they played as vs. true profile. Control Group is colored. . . . . . . . . . . . . . . . . . . . . . . . . 166

Table 8 – Results of the Mann-Whitney U test, considering that rejecting the null hypothesis means that the answers from players with a matching profile are greater than those playing a different profile. We reject the hypothesis with a p-value equal or smaller than 0.05, meaning less than 5% chance of a type I error. . . 168

Table 9 – The 12 inputs used to create the content for the online-generation experiment, as well as what are the inputs for the pre-test questionnaire they represent and what weights each answer is converted when normalized as inputs for the quest generator. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171

Table 10 – Dunn's test result for the normalized distance from the desired input for the levels generated for each quest generator's input. . . . . . . . . . . . . . . . 173

Table 11 – Dunn's test result for the normalized sparsity of enemies in the levels generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . 173

Table 12 – Dunn's test result for the normalized standard deviation of enemies' position in the levels generated for each quest generator's input. . . . . . . . . . . . . 174
Table 13 – Dunn's test result for the resulting fitness of the levels generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 174
Table 14 – Dunn's test result for the total rooms in the levels generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 175
Table 15 – Dunn's test result for the total keys in the levels generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 175
Table 16 – Dunn's test result for the total locks in the levels generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 176
Table 17 – Dunn's test result for the linearity in the levels generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 176
Table 18 – Dunn's test result for the total enemies in the levels generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 176
Table 19 – Dunn's test result for the total quests generated for each quest generator's input. 177
Table 20 – Dunn's test result for the number of achievement quests generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . 177
Table 21 – Dunn's test result for the number of creativity quests generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 178
Table 22 – Dunn's test result for the number of immersion quests generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 178
Table 23 – Dunn's test result for the number of mastery quests generated for each quest generator's input. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 178
Table 24 – Total players by profile according to their answers on the pre-test questionnaire during the experiment with the online generators. . . . . . . . . . . . . . . . 179
Table 25 – Results of the Mann-Whitney U test, considering that rejecting the null hypothesis means that the answers from players with a matching profile are greater than those playing a different profile. We reject the hypothesis with a p-value equal or smaller than 0.05, meaning less than 5% chance of a type I error. . . 191

# LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| AI | Artificial Intelligence |
| BDI | Belief-Desire-Intention |
| CNN | Convolutional Neural Network |
| DFS | Depth-First Search |
| EA | Evolutionary Algorithm |
| GA | Genetic Algorithm |
| HCI | Human Computer Interaction |
| KR | Key Room |
| LR | Locked Room |
| MAP-Elites | Multidimensional Archive of Phenotypic Elites |
| ML | Machine Learning |
| MMO | Massive Multiplayer Online |
| MMORPG | Massive Multiplayer Online Role Playing Game |
| MUD | Multi-User Dungeon |
| NPC | Non-Player Character |
| NR | Normal Room |
| Null Hypothesis | H0 |
| PCG | Procedural Content Generation |
| QD | Quality-Diversity |
| RPG | Role-Playing Game |

# CONTENTS


1 INTRODUCTION . . . . . 33
1.1 Motivation . . . . . 33
1.2 Research Questions . . . . . 36
1.3 Thesis Structure . . . . . 37

2 CONCEPTS, TECHNIQUES, AND TECHNOLOGIES . . . . . 39
2.1 Action-Adventure Games . . . . . 39
2.2 Dungeons . . . . . 41
2.3 Game Engine . . . . . 43
2.4 Procedural Content Generation . . . . . 46
2.5 Evolutionary Algorithm . . . . . 48
2.6 MAP-Elites . . . . . 49
2.7 Path-Finding Algorithms . . . . . 51
*2.7.1 A* Algorithm . . . . . 51*
*2.7.2 Depth-First Search . . . . . 53*
2.8 Markov Chain . . . . . 54
2.9 Formal Grammar . . . . . 55
2.10 C4 Model . . . . . 57
2.11 Final Remarks . . . . . 60

3 LITERATURE REVIEW . . . . . 63
3.1 Player profiling and content adaptation . . . . . 63
3.2 Dungeon Generation . . . . . 70
3.3 Enemy Generation . . . . . 71
3.4 Quest Generation . . . . . 73
3.5 Generation of multiple contents . . . . . 78
3.6 System Architecture . . . . . 83
3.7 Summary . . . . . 86

4 METHODOLOGY . . . . . 89
4.1 System Context . . . . . 89
4.2 Systems' Containers . . . . . 92
*4.2.1 The Profile Analyst System . . . . . 93*
*4.2.2 The Procedural Content Generator System . . . . . 93*

*4.2.3 The Game System . . . 94*
4.3 Procedural Generation . . . 96
*4.3.1 Evolutionary Algorithm for Dungeon Generation . . . 97*
*4.3.2 MAP-Elites for Dungeon Generation . . . 108*
*4.3.3 Evolutionary Algorithm for Enemy Generation . . . 112*
*4.3.4 MAP-Elites for Enemy Generation . . . 114*
*4.3.5 Formal Grammar Quest Generator . . . 120*
*4.3.6 Markov Chain Stochastic Grammar Quest Generator . . . 121*
*4.3.7 Cellular Automata Room Generator . . . 124*
*4.3.8 Content Orchestrator . . . 125*
*4.3.9 PCG Workflow Example . . . 126*
4.4 Game Prototype . . . 130
*4.4.1 Gameplay . . . 134*
*4.4.2 Enemies . . . 137*
*4.4.3 Quest System . . . 139*
*4.4.4 Dialogue System . . . 141*
*4.4.5 Content Placement . . . 142*
4.5 Player Profile . . . 144
4.6 Final Remarks . . . 146

5 RESULTS . . . 149
5.1 Enemy Generator Results . . . 150
*5.1.1 Enemy Generator Evolutionary Algorithm's Performance . . . 150*
*5.1.2 Enemy Generator Evolutionary Algorithm's Diversity . . . 152*
*5.1.3 Enemy Generator Evolutionary Algorithm's Player Feedback . . . 152*
5.2 Offline Generator Results . . . 159
*5.2.1 Experimental Setup . . . 159*
*5.2.2 Experimental Results . . . 164*
5.3 Online Generator Results . . . 168
*5.3.1 Experimental Setup . . . 168*
*5.3.2 Computational Experiments . . . 171*
*5.3.3 Player Feedback from Online Generation . . . 178*
*5.3.3.1 Visual Analysis . . . 180*
*5.3.3.2 Statistical Analysis . . . 189*
*5.3.3.3 Final Remarks . . . 192*

6 FINAL REMARKS . . . 193

BIBLIOGRAPHY . . . 197

APPENDIX A OFFLINE EXPERIMENTAL SETUP EXTENDED RESULTS . . . . . . . . . . . . . . . . . . . . . . . . . . 211

CHAPTER

# 1 INTRODUCTION

Procedural Content Generation (PCG) can be defined as "*the algorithmic creation of game content with limited or indirect user input*", according to (TOGELIUS *et al.*, 2011). What started in the 80s in games like Rogue and Elite to solve storage shortage problems is now pivotal to game development by increasing replayability, reducing authoring burden, and enabling particular aesthetics (RISI; TOGELIUS, 2020a). A relevant example is Hades, which won two *Game of the Year* awards in 2020 and applies PCG in its levels (SUPERGIANT GAMES, 2020). PCG is also heavily researched in the academy, both as a study subject by itself and giving birth to artificial intelligence algorithms and applications in serious games, e.g., for educational purposes (RISI; TOGELIUS, 2020b; RODRIGUES; BRANCHER, 2019).

Research in fields of Computational Intelligence seeks methods to enhance the PCG used in the game industry nowadays, be it through deep learning(LIU *et al.*, 2021), machine learning(SUMMERVILLE *et al.*, 2017), or search-based algorithms(TOGELIUS *et al.*, 2011). We also have publications in Software Engineering, using the mechanics-dynamics-aesthetics framework combined with an agile methodology based on SCRUM, as in (ATMAJA; PARLIKA *et al.*, 2019), or documenting PCG-based design patterns(TREANOR *et al.*, 2015b). PCG research benefits not only the game industry, but also the field of Artificial Intelligence (AI) research in a more generic sense (RISI; TOGELIUS, 2020a).

## 1.1 Motivation

PCG algorithms may generate a wide range of content using different algorithms. For example, levels have been created by a multitude of algorithms, especially using search-based approaches, mainly with the Evolutionary Algorithm (EA), but also with formal grammars and Machine Learning (ML) (VIANA; SANTOS, 2021). The same goes for puzzles created using grammar-based methods, search-based methods, Cellular Automata, Markov Chains, and other techniques (KEGEL; HAAHR, 2019).

Music is often generated using Neural Networks, ranging from Generative Adversarial Networks, Recurrent Neural Networks, Variational Autoencoders, and the like, but also with Markov Chains and EAs (CIVIT *et al.*, 2022). On the other hand, Terrain generation focuses on fractals, grammars, tiling, simulations, EAs, and so forth (VALENCIA-ROSADO; STAROSTENKO, 2019). Stories are generated using graph and grammar-based approaches, AI planners, Genetic Algorithms, Monte Carlo Tree Search, ML, etc. (ALHUSSAIN; AZMI, 2021). Although not much explored in the literature, we can also generate enemies, usually through EAs (PEREIRA; VIANA; TOLEDO, 2021), as well as visual assets, like textures (DONG *et al.*, 2020), and other assets for games.

Although we have all these different researches in the area, the grand vision of PCG is the generation of complete games, according to Liapis et al. (Liapis *et al.*, 2019); which still needs to be achieved. We also are distant from adapting these games to different players: most of the PCG research until the early 20s focused on generating specific content, and only some could support several players' profile. Liapis et al.(Liapis *et al.*, 2019) show some challenges we as researchers must surpass while presenting an informative review on works generating multiple contents simultaneously. They also elaborate a set of proposals on how we can create and align cohesively or orchestrate different creative facets.

These creative facets are a simplification of the six core creative elements of a game procedurally generated: *visuals*, *audio*, *narrative*, *rules*, *levels* and *gameplay*. However, the same facet may be generated by different algorithms in the same game: e.g., dungeons and rooms created by various algorithms, but both are levels' content and dialogues, locked-door puzzles, Sokoban puzzles, and quests can be generated each by a different solution, but all of them are narrative components. Therefore, some works might create two or three facets but end up being a collection of four or five algorithms working in harmony. The review provides many insights from relevant pioneering works and describes the different strategies to cohesively manipulate the generation of these many contents.

The authors call the strategy of this multiple content generation an *orchestration* of such contents, while the algorithm that controls the different generators is called a *maestro* algorithm. We point out that most algorithms in (Liapis *et al.*, 2019) cannot generate more than a pair of contents simultaneously, especially in a way that they can adapt themselves independently and with great variety.

Some recent works are approaching the generation and orchestration of multiple contents while learning from players, although AI-controlled players, as in the case of (KARAVOLOS; LIAPIS; YANNAKAKIS, 2021). They use a shooter game with levels and rules generated via algorithms and AI agents to play the game. Levels, character class parameters, and match outcomes are input to train a convolutional neural network that predicts important gameplay data.

In this doctorate research, we advance on the concept of multiple content orchestration

by focusing on creating the narrative, rules, and levels facets. More specifically, we work toward combining the systems that create levels, the configuration of elements in each room, enemies to populate the levels (and how to distribute them), and narrative (especially quests, but also locked-door puzzles inside dungeons). We also provide a concrete architectural example of a system orchestrating different contents and adapting these contents based on players' preferences. Said architecture provides enough flexibility to employ different algorithms when generating content, as well as to add new algorithms with relative ease.

One of our main inspirations was the work of (HEIJNE, 2016), where a procedural clone from *The Legend of Zelda: A Link to the Past* can generate levels, enemies that populate them, and puzzles for some rooms. However, their algorithms are rule-oriented, providing low variability. They can also select the setting that best matches each player's preferences, although only for a predefined set of options and using a fixed set of rules.

To create content based on the feedback gathered from users, we need to collect and process data from them. The game industry collects and studies the data to find preferences towards gameplay elements such as mechanics, balancing aesthetics, etc. They do so usually thanks to A/B tests, with statistical analysis and Bayesian approaches (COLLINS, 2014). More recent applications apply Data Mining algorithms and Artificial Neural Networks to handle even greater data sets, with a high success rate. An example is OpenAI's AlphaStar, an artificial intelligence capable of defeating professional Starcraft II and Dota 2 players. The AI analyzed the data gathered from thousands of players (OPENAI *et al.*, 2019; VINYALS *et al.*, 2019).

In this context, it is relevant to understand the users' motivations and preferences when creating content. We can employ such information to determine player classes or types, creating clusters of players with similar tastes, play styles, and skills. Many attempts have been made to understand players by creating theoretical categories or determining different gameplay motivations. Such understanding achievement can employ data collection explicitly (surveys and questionnaires) or implicitly (via gameplay data) for one or several game genres (BARTLE, 2003; YEE, 2007; VAHLO *et al.*, 2017a; MELHART *et al.*, 2019; HEIJNE, 2016; BONTCHEV; GEORGIEVA, 2018; LORIA; MARCONI, 2018; COWLEY; CHARLES, 2016). However, each one produced different results, and there needs to be more consensus on collecting and handling this data to learn about players accurately.

Experiments in the Human Computer Interaction (HCI) area show that it is possible to use preferences and personality data to enhance software use. In a study where 660 participants had their data collected, it was possible to identify that some personality traits were significant to maintain people using a given software (ORJI; NACKE; MARCO, 2017). Other studies showed promising related results in games: Bicho and Martinho evolve an infinite runner's difficulty procedurally to match its player's skill better. They always presented obstacles to be avoided with two different game mechanics. Every time the player chooses one, it gets harder for that single mechanic. Therefore, a simple player profile (preference between the two mechanics)

was defined and used to create a more enjoyable game (BICHO; MARTINHO, 2018). Heijne also adapted content depending on collected data from the player's performance in an action-adventure game. There is a procedural generator able to choose between some minor variants of level configuration, placement of enemies, and puzzles' layout sorted by the difficulty level of each content (HEIJNE, 2016).

There are recent works, as in (GRAY; ZHU; ONTAñóN, 2021), applying ML-based approaches to model players' preferences. The methods collect and process data from previous interactions of the player with the game (as well as simulated data to enhance the database). The output suggests the best content for the player in their current gameplay state, being able to change the recommendation over time. Their work uses the Multi-Armed Bandits problem to model the player and be the AI that adapts the game to enhance their experience.

For our project, we use explicit data from questionnaires to define our players' preference as well as evaluate our system's quality. There are successful cases in literature (RIVERA-VILLICANA *et al.*, 2018; VAHLO *et al.*, 2017b; YEE; DUCHENEAUT, 2018; HEIJNE, 2016) applying questionnaires to identify players' profile and evaluate the gameplay. However, we also collect implicit data from users' gameplay to check their performance and whether it matches the questionnaire's input profile.

## 1.2 Research Questions

The main research question of this doctorate is summarized next:

*How can procedural content generation be effectively applied to simultaneously create multiple and coherent contents in a real-time environment and adapt them to different users' needs?*

Thus, some specific research questions are listed next, aiming to state better the context of the desired contributions:

- What type of architecture can effectively coordinate the generation of narrative, levels, and rules facets?
- How to build a modular architecture which can support different generating algorithms with as few changes as possible to other containers?
- How can an orchestrator algorithm process the generated content to make them feasible and more enjoyable when placed together?
- How can the orchestrator deal with simultaneous content generation, considering or learning from player or game designer profiles?
- How to adapt the simultaneous content generation within online environments?

The answer to those questions will lead us to push the computational boundaries when approaching simultaneous content generation.

## 1.3 Thesis Structure

The doctorate thesis is organized in the next chapters as follows:

- Chapter 2 discuss some essential concepts, techniques, and technologies for the clearer understanding of the topics brought in this thesis. Specially these related to procedural content generation, software architecture, machine learning and game development.
- Chapter 3 presents in finer detail the most important works in literature that support this work, specially the ones cited in this chapter.
- Chapter 4 we discuss the methods we use to create every different game facet, how our architecture manages the different algorithms, how the maestro interlaces content and tries to enhance the quality and feasibility, how we collect and analyze data from players and details on the game prototype in which we test the results.
- Chapter 5 presents the experimental setups produced through this doctorate research and their respective results. We present both computational results from running the algorithms and comparing against metrics in the literature and from data we collect and analyze from users using the game prototype.
- Chapter 6 has our final remarks on the doctorate project.

CHAPTER

2

# CONCEPTS, TECHNIQUES, AND TECHNOLOGIES

This chapter describes the concepts behind the main themes approached in the current research. First, we introduce the action-adventure game genre (Section 2.1), the one to which the game prototype developed for this research belongs. The concepts of dungeons in games follow (Section 2.2), the main challenges the player must overcome inside them, and a model to represent said dungeons. Then, we present the game engines and the one used to develop our game prototype, used as a test-bed: the Unity engine (Section 2.3).

Next, we state the main ideas about Procedural Content Generation (Section 2.4) followed by the concept of Evolutionary Algorithm (Section 2.5), the core algorithm behind the dungeon and enemy generators used in this project. We describe the MAP-Elites algorithm, an algorithm based on the Evolutionary Algorithm that searches for diverse solutions while preserving quality (Section 2.6). We also mention some path-finding algorithms (Section 2.7), especially the A* and Depth-First Search, applied to evaluate this work's levels. Furthermore, concepts of Markov Chain (Section 2.8) and Formal Grammar (Section 2.9), both used to generate our quests, are also introduced. Finally, we describe the C4 model used for our system's diagrams (Section 2.10).

## 2.1 Action-Adventure Games

The action-adventure game genre demands great focus on physical challenges, including reflex and motor coordination, similar to action games, but also includes many elements from adventure games. Some elements are item gathering, exploration, and interaction with the environment (and the people inhabiting it). The story usually takes place in a large world connecting different essential areas, and the game requires puzzle solving of different types (ROLLINGS, 2003; RYAN, 2002; LUBAN, 2002). For most games, the player controls a single

avatar, the story's protagonist. This mix of elements brings the genre close to the Role-Playing Game (RPG), although the latter generally focuses more on story-telling, level, and abilities system, which are not present in the majority of action-adventure games (ADAMS; ROLLINGS, 2006).

It is essential to highlight that the genre definition for games is not exact and varies between authors and players. Moreover, many definitions valid for overall genre classification may conflict with the gameplay of specific games, demanding a case-by-case study. The most referred game as the progenitor of the action-adventure genre is the *Adventure* game (ROBINETT, 1979), an adaptation of a text adventure game that allowed the player to fight battles as in action games. Another landmark for the genre was the game *The Legend of Zelda* from 1986, which was a huge commercial success and one of the first to combine the main elements of the genre skillfully. *The Legend of Zelda* brought exploration of a world much larger than usual at that time with transport and inventory puzzles, combats based on action mechanics, a currency system, and a level system close to the ones in RPGs but without experience points. It was the first title of one of the most famous game franchises, currently consisting of 19 leading titles, all in the action-adventure genre.

Nowadays, the genre has a roster of many successful games, with many nominated for The Game Awards in 2017. Some of them were: *The Legend of Zelda: Breath of the Wild* (Figure 1, top) (FUJIBAYASHI, 2017), *Horizon Zero Dawn* (Figure 1, bottom)(JONGE, 2017) and *Super Mario Odyssey* (MOTOKURA, 2017), all in the Game of the Year category (along with others), with *The Legend of Zelda: Breath of the Wild* being the winner. Furthermore, *Hellblade: Senua's Sacrifice* (ANTONIADES, 2017), *Uncharted: The Lost Legacy* (ESCAYG, 2017), *Metroid: Samus Returns* (HOSOKAWA, 2017) and *Nier: Automata* (TARO, 2017), all action-adventure games, were nominated for other categories.

Aside from their remarkable commercial success, the games from the genre also hold an important place in the research area, being a test platform for many PCG algorithms, as we will report in Chapter 3. The many types of puzzles coming from action-adventure games bring varied and exciting challenges for researchers, which use the most different approaches ranging from evolutionary algorithms to statistical learning and grammar to solve them (SUMMERVILLE; MATEAS, 2015; SUMMERVILLE *et al.*, 2015; LIAPIS, 2017; VALTCHANOV; BROWN, 2012; DORMANS; BAKKES, 2011).

One of the genre's most common challenges is eliminating all enemies in a given area. This task turns out to be a logical puzzle in many situations, as the majority of enemies in the games are weak to specific weapons or items or need to be attacked in a specific time window. Other more straight logical challenges are: solving spatial puzzles by pushing objects around the level to proper locations, sometimes with movement restriction; solving logic puzzles varying from common logical and mathematical problems recreated in the play space, such as using a specific item's attribute to access a new area; and the spatial awareness puzzle, or locked-doors

Figure 1 – *The Legend of Zelda: Breath of the Wild* (top) and *Horizon Zero Dawn* (bottom), two award-winning action-adventure games released in 2017

mission.

## 2.2 Dungeons

Dungeons are the main play space for most adventure, action-adventure, and Role-Playing Game (RPG) projects. They have an entrance, many intermediate rooms, and a final room with a challenging boss monster, which may protect a rare item or key used for game progression and/or to reach new areas. The intermediate rooms commonly hold different challenges, varying with the individual mechanics of each game. The challenges usually are:

- Random meetings with monsters, which is the case for RPGs.

- Puzzle challenges of many types, e.g., moving objects to open a passage or using specific items in particular objects.

- Fighting with enemies inside the level for adventure and action-adventure games.

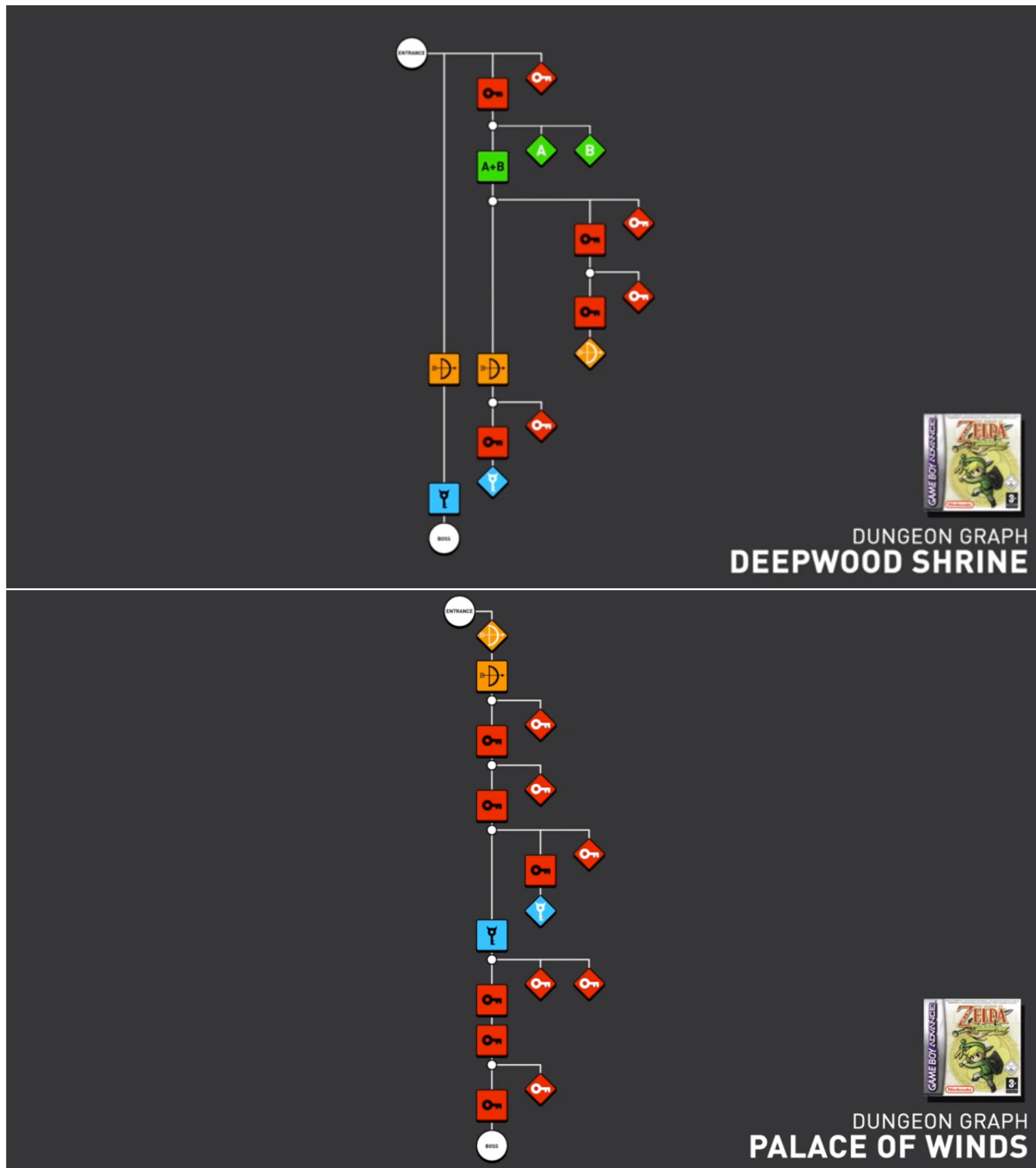
Furthermore, it is a commonplace to find items scattered throughout the rooms and, in some cases, switches that allow access to other parts of the dungeon. The items spread across rooms can be keys, consumable items, or equipable items used to unlock passages to other rooms in some games. Our project focuses on procedurally generating content for this last set of challenges: finding items scattered throughout the dungeon's rooms. *The Legend of Zelda* game franchise is based on this kind of mission, along with puzzles and enemies spread inside some rooms. However, the puzzles and enemies' placement are beyond the scope of the current research. As already stated, we intend to create dungeons with different features related to the positioning of rooms and the distribution of items able to open locked paths and their locks. The *Boss Keys* series of videos analyzes this type of play space and missions, including the creation of appealing visual representation through graphs to represent the different shapes of dungeons, their keys, and locks. Figure 2 shows two dungeons from *The Legend of Zelda: The Minish Cap* game (FUJIBAYASHI, 2004), converted into the graphs suggested by the creator of *Boss Keys* series.

In the upper graph of Figure 2, we can see the dungeon with many branches and wider path choices for the player. The branches and wider path choices demand the player to backtrack in the dungeon, that is, go back to an already visited room to reach the goal. The lower graph in Figure 2, however, has a linear structure that becomes unnecessary for the player to backtrack for most parts of the dungeon. The dungeons from the first kind usually have more puzzle-based challenges, while the second one holds more action-based challenges by hosting more and stronger enemies in its rooms [1].

The ability to allow diversity in the content of rooms, and their placement in the dungeon, is essential to enable a PCG method to be general enough when creating both types of a dungeon. This characteristic is considered by the algorithm presented in our research, as described in Chapter 4. The graph representation also allows us to easily observe another goal for our algorithm to be viable in a commercial game: the existence of unique pairs for lock-key combinations. Figure 3 shows different pairs with different colors (diamonds are keys and squares are locks) and the order for the player to solve the challenges until the completion of the dungeon, which is the upper one in Figure 2.

[1] <https://www.youtube.com/watch?v=KEVJXqV7XMc>

Figure 2 – Graph abstraction of 2 levels from *The Legend of Zelda: The Minish Cap* game. Only rooms containing items which are essential for the player's progression are shown. Each color identifies a different type of key-lock pair. Keys are represented as diamonds and locks as squares. The vertical lines show a sequence of rooms in the same path, while the horizontal ones identify a path's branching. Many rooms in the same horizontal line may be visited by players, if they wish so. Extracted from: <https://www.youtube.com/watch?v=KEVJXqV7XMc>

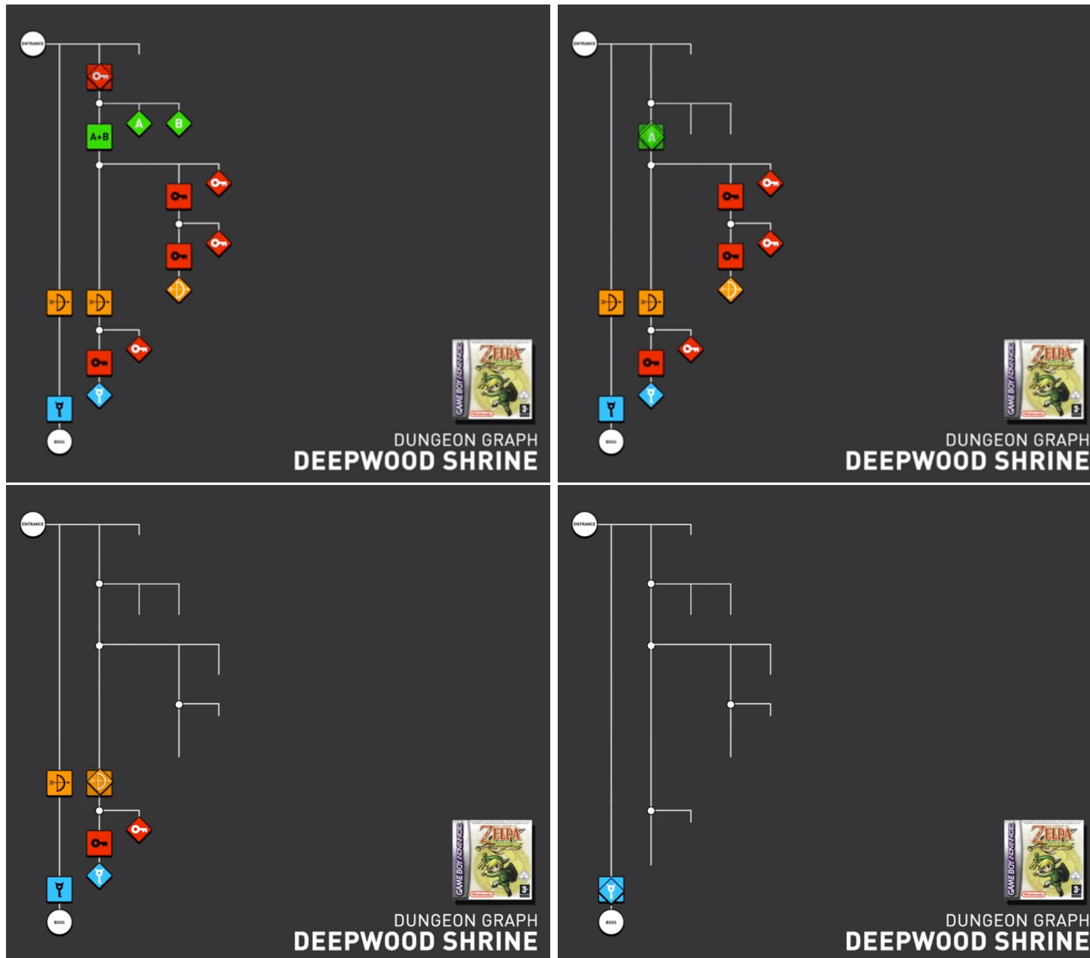



## 2.3 Game Engine

A game engine is a software framework designed to aid programmers, artists, and game designers in creating, implementing, and building video games. It is mainly composed of a render engine for 2D and/or 3D graphics, a physics engine or a collision detection and response engine, a sound engine, script language, animation, artificial intelligence, network connection, data streaming, memory manipulation, thread generation, localization support, and scenes graph.

However, the complexity of engines has increased over the years, giving even more

Figure 3 – A possible solution for the puzzle in the upper dungeon from Figure 2. In the upper left image, the first key is found and used in the first lock. In the upper right one, two levers are used to open the green door. In the lower left image, the player finds the bow and opens the first door requiring its use. At last, in the lower right image, the player gets the boss key and opens the last lock. Extracted from: <https://www.youtube.com/watch?v=KEVJXqV7XMc>




functionality to them (GREGORY, 2014, Chapter 1). The ability to use a single game engine to create many games saved plenty of resources in time and money for game development companies, instead of the old need to develop an engine (or heavily alter a previous one) for each new game. Most engines nowadays allow the user to build games for many platforms with just a few (or even without) alterations in the project, saving a lot of time (PEDERSEN, 2009).

The game engine originated in the middle of the 90s due to the software architecture from the game *Doom* (HALL, 1993). This game architecture successfully separated its main software components, art assets, game worlds, and rules which affected the player's experience.

Separating the components makes it faster and easier to license new games and rebuild previous ones as new products. Thus, it facilitates the creation of new art assets, world layouts, weapons, characters, vehicles, and game rules, with only minor changes in the engine's software. At the end of the 90s, some games like *Quake III Arena* (JAQUAYS, 1999) and *Unreal* (BLESZINSKI, 1998) were made with re-usability in mind. Their engines were highly cus-

tomizable through script languages, and the licensing of such motors turned out to be a viable secondary means of profit for their creators (GREGORY, 2014, Page 11).

Nowadays, many game engines are available in the market, each with its own business model, solutions, architectures, script languages, functionality, and approaches to enable usage by non-programmers. The two main engines will be presented briefly, together with the reasons why the *Unity* engine was chosen by our project.

The *Unreal Engine*, now in version 4, is one of the most powerful engines and one of the oldest (since 1998, when the game *Unreal* was released). As its most relevant business competitor, *Unity*, both are used for applications beyond gaming as apps for enterprises, simulations, cinematic experiences, and movies. However, their focus is on creating high-quality games for many platforms, including virtual and augmented reality. Some unique selling points for *Unreal* are the real-time photorealistic rendering; the complete C++ source code being available, allowing for customization; a *Blueprint* system for visual scripting, allowing designers to make many contents without knowing how to code; robust multiplayer framework; and many others. Its business model allows free use of the software with all its functionality. However, 5% of all revenue from the games made with the engine goes to the company. The engine is aimed at small and large development teams; therefore, the engine is used in many independent games developed by smaller studios with a handful of developers and usually lower development costs. But it is also used to develop *Triple A* games made by large studios, which spend a large amount of investment and employ many developers (GAMES, 2018).

The *Unity* engine allows handling both 2D and 3D games, teams, and projects of different sizes, and it has many state-of-the-art functionalities to help create *triple A* games. There is no native visual script in *Unity*, but there are third-party alternatives in its asset store. Unity is well-known for trying to bond deeply with its users and attract new users with the *Unity User Groups*[2], the *Unite* event [3] and the *Unity Developer Day*[4]. Those heavy marketing strategies made it, currently, the most used game engine in the world and with the largest number of players for their games. The games made with unity had 5 billion downloads in the 3rd quarter of 2016; a total of 2.4 billion different mobile devices running those games; 34% from the 1000 biggest free mobile games made with *Unity*; and 770 million people playing *Unity* games. Roughly 90% of *Samsung VR* and 53% of *Oculus Rift* games were made with *Unity*.

Another important feature that makes it so attractive is the direct integration of many secondary tools essential for game development in the engine. For instance, the *Unity Analytics* allows developers to easily obtain gameplay and behavior data from players without needing third-party solutions. There is also the *Anima2D* tool, which helps the in-engine creation of 2D animation, and the level design tool *ProBuilder*. The current programming language in *Unity* is

[2] <https://unity3d.com/pt/community/user-groups>
[3] <https://unite.unity.com/>
[4] <https://unity3d.com/pt/events/unity-developer-day-brasil-2017>

C#. Its business model is based in selling licenses based on the user's revenue: the free license gives all the core features of the engine and allows companies with less than U$ 100,000 a year revenue to publish their games for free. For yearly revenues ranging from 100,000 to 200,000 dollars, it is necessary to buy the *Unity Plus* license, which costs U$35 monthly and offers some advantages as additional features for the *Analytics*. The *Unity Pro* license is a need for companies with revenue greater than $200,000 a year, and it provides support from *Unity*'s employees. Its current pricing is a monthly fee of U$125 [5]. Another advantage of *Unity* is its learning process that is considered more accessible compared to other market leaders, according to developers and game journalists (RASHID, 2016; DEALS, 2016; PLURALSIGHT, 2015).

In comparison, *Unreal Engine* build for 15 different platforms [6] and *Unity*, 28[7], making them a better choice for developers who want to reach the largest consumer market possible.

Therefore, we chose *Unity* as the engine to develop our game prototype based on the following reasons:

- Faster learning rate and possibility to publish in many platforms, being the most used engine in the industry.
- The use of C# language (preferred by the author) and many integrated tools that become development easier in small teams.
- Collecting data from the players' gameplay through *Unity Analytics*. enditemize

## 2.4 Procedural Content Generation

The technique known as Procedural Content Generation was originally developed to reduce memory usage by game content, allowing the insertion of more content since many of them were limited by the capacity of cartridges and floppy disks. By the time of its creation, in the 80s, computers had little memory and processing power, so this was a must for improving game content. Trying to break the limits from the other games, the developers from *Rogue* and *Elite* were one of the pioneers using PCG. *Rogue*, a dungeon crawler game, generated levels only when the player was about to enter them. Their creation applied algorithms that defined features such as the size of the current room, the connections between rooms, the location of items and enemies, and placement of the level's entry and exit (DESIGN, 1984). *Elite* is a game about space combat and commercial trade between planets. Its 256 planets for each of the eight galaxies were procedurally generated from a random seed (BELL; BRABEN, 1984).

The original motivations to use PCG turned obsolete with the improvement of memory, processing speed, and overall capacity of computers. However, the advances in technology

[5] <https://store.unity.com/>
[6] <https://www.unrealengine.com/en-US/what-is-unreal-engine-4>
[7] <https://unity3d.com/pt/unity/features/multiplatform>

brought a massive expansion of the number of players, an increase in games' complexity, the improvement of their contents, and, accordingly, the increase in the cost and production time of games. The new challenges made the game developers look back to PCG, but with novel goals: create large amounts of diverse content as fast as possible and, in some cases, adapt them to different players. These goals aim to add more game replay value by creating different missions, and levels in each new playthrough (IOSUP, 2011; HENDRIKX *et al.*, 2013). Some recent games that use PCG are *No Man's Sky* (GAMES, 2016b), *The Binding of Isaac* (MCMILLEN; HIMSL, 2011b), *Elite:Dangerous* (DEVELOPMENTS, 2014) and *Civilization VI* (GAMES, 2016a).

The use of PCG in the game industry is usually made possible by applying random algorithms without adapting directly to the user's profile or by randomly selecting elements from a database of assets (levels, models, textures, or any other type) previously created by human developers. They can also use available algorithms for procedural art and animation generation, which differ from the other types of PCG approaches. The developers generally create algorithms tailored to their specific game, making it quite difficult to reuse in other projects.

The game *Civilization VI* applies procedural generation of models to create different contents, including farms. Moreover, the sea coasts from the maps are created automatically using spline curves, but with their randomness limited by artists. The developers made the visual effects procedurally from cracking ice next to the coast [8]. *Elite:Dangerous*, on the other hand, uses PCG in two different ways. The first is generating the planet's ecosystems. Artists created a set of rules, together with programmers, that, when applied to planets, provided a visually pleasing and realistic ecosystem [9]. The second one creates star systems, called *Stellar Forge*. The developers implemented many rules based on astronomy to simulate a wide range of realistic physical and chemical reactions to create unique systems for each combination of parameters extracted from random seeds [10].

The game *No Man's Sky* was acclaimed for an ambitious PCG approach: it was able to procedurally generate an entire universe, with almost 18 quintillion planets, each with flora and fauna also created through algorithms. Its music was also procedurally generated. For each creature to be unique and believable, the developers created a set of feasible resources for each part of the body (e.g., different heads, legs, bodies, etc.). Considering what elements were already part of the body, some elements of the set of yet-uncreated features were marked as non-selectable. After selecting such an element, its child components were chosen. For example, after the selection of a type of head, a mouth must be chosen for the creature. However, only those compatible with the current head remain in the selection pool for the algorithm [11].

[8] <http://firaxis.com/?/blog/single/procedural-terrain-generation-in-sid-meiers-civilization-v>
[9] <https://www.youtube.com/watch?v=GEVutbSqBI0&t=466>
[10] <https://us2.campaign-archive.com/?u=dcbf6b86b4b0c7d1c21b73b1e&id=76df98203b#StellarForge>
[11] <http://3dgamedevblog.com/wordpress/?p=836>

Finally, the game *The Binding of Isaac* applies PCG to create its levels in the shape of dungeons, as the ones in this research. Although relatively older than previous examples, the game was remade and launched in 2014 as *The Binding of Isaac: Rebirth* (MCMILLEN, 2014), and it had two expansions as downloadable content: one at 2015 and another at 2017. It was also ported for consoles of the current generation, having a life cycle that lasted until 2017. Part of this longevity and success is thanks to the high challenge and replayability factors provided by PCG. Its algorithm depends a lot on human work: more than a thousand rooms were created by the game designer, ordered according to difficulty and chapter to which they belong (an aesthetic factor), and randomly chosen to be placed in the dungeon. They draw the content features of each room from more than 50 enemies and items. There are different probabilities to be placed in each room according to their relevance. Enemies vary according to luck: the same enemy can be created, with lower probability, as a stronger specimen that can have different colors (also drawn randomly), each with a unique ability [12]. The game development still demands a lot of effort from the creators by generating the initial content, but the PCG algorithm of the game, even though being simple, can return significant variability and helped in the commercial success: the original title sold more than 3 million copies in the *Steam* platform, and its remake more than 2 million copies [13].

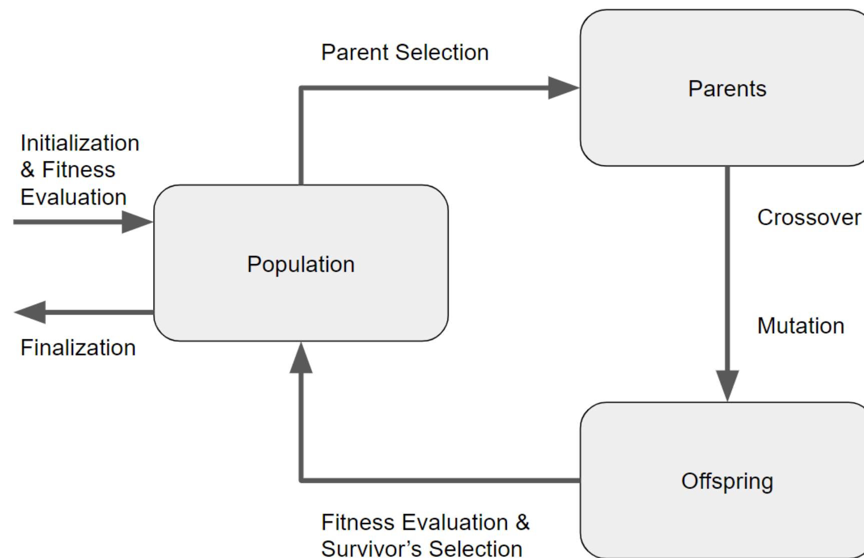
## 2.5 Evolutionary Algorithm

Evolutionary Algorithms (EAs) are stochastic meta-heuristics for search and optimization based on Darwin's theory of evolution. Although there are many subgroups and branches of the main concept of the algorithm, all follow the same *modus operandi*: a population consisting of different individuals is created by a set of rules (usually a uniform random distribution). Then, individuals' selection for breeding and survival is simulated for many generations until a specific stop criterion has been satisfied. The individuals are representations of possible solutions for the problem, coded in myriad ways. The individuals are evaluated through some metric, mostly known as a fitness function. The theory of evolution-based features of the EA arises in survival and breeding selection, in which the best individuals have a greater chance to breed and/or be kept for the next generation. Figure 4 shows a diagram representing the basic procedures of an EA.

For the general case, the parent selection for breeding happens by comparing the fitness of individuals from the population, with the better ones having more chances to participate in the recombination process. A function explicitly defined for the problem calculates the fitness value. After several parents have been chosen (in most cases, a couple), crossover and mutation breeding procedures are executed since a probability condition is met. The crossover of individuals will shuffle pieces of the parent individuals to produce one or more new individuals. Next, the

[12] <http://edmundmcmillen.blogspot.com.br/2011/09/binding-of-isaac-gameplay-explained.html>
[13] <http://steamspy.com/search.php?s=binding+of+isaac>

Figure 4 – General representation of an Evolutionary Algorithm. Extracted from: (EIBEN; SMITH, 2003) (p. 17).




novel individuals may be mutated, which usually means an unexpected change adding further variability to the representation of the individual.

After the creation of offspring individuals, the selection of the survivors begins by choosing a balanced mix of old and new individuals, where the priority is the survival of the fittest individual. The algorithm ends after a stop criterion which can be the execution of a fixed number of generations; the best possible solution found, no improvement in the best solution for a fixed number of generations, among others. The Algorithm 1 exemplifies the execution of an EA with the most common procedures in literature (EIBEN; SMITH, 2003).

**Algorithm 1** – Basic Evolutionary Algorithm. Adapted from: (EIBEN; SMITH, 2003) (p. 16)

```
1: INITIALIZE population with random solutions
2: EVALUATE each individual
3: while STOP CRITERION not satisfied do
4:     SELECT parents
5:     CROSSOVER pairs of parents
6:     MUTATE resulting children solution
7:     EVALUATE new individuals
8:     SELECT individuals for the new generation
9: end while
```

## 2.6 MAP-Elites

Multidimensional Archive of Phenotypic Elites (MAP-Elites) is an optimization algorithm; that is, it searches the space of candidate solutions for a problem, seeking to optimize the result. It is also part of the Quality-Diversity (QD) algorithms: optimization algorithms that

search not for the optimum of the cost function, but provide a set of high-quality solutions, differing according to user-defined features of interest. These algorithms assume not only that the objective function returns the fitness value $f\theta$, but also that it returns a feature vector (or behavioral descriptor) $b\theta$. The feature vector usually describes how the solution solves the problem, and the value quantifies how well it solves it. This feature vector may be the trajectory of a robot, and the value measured is the distance to the target, for example.

By defining $B$ as the feature space, the goal of QD optimization is finding for each point $b \in B$ the parameters $\theta$ with the maximum (or minimum) fitness value. The QD problem may be viewed as optimizations constrained by each feature vector. The hypothesis is that solving said problems together is likely faster than independent constrained optimizations. We can assume that high-performing solutions for close feature descriptors will likely be near each other. thus, sharing said information may be beneficial.

QD algorithms return a set of solutions, sometimes called collection, archive, or map. They are expanded, improved, and refined during the optimization process, as each point in them is a different solution type or species. We consider two similar solutions with similar feature vectors as belonging to the same solution type, thus competing to be maintained in the collection. This similarity is defined using a hyperparameter, which sets the tolerance used to determine whether two features are different. They define a "resolution" in $B$, the feature space, as only one solution will occupy a particular region in this space.

We evaluate a QD algorithm by the solution's performance in each type and the coverage of such behavior space. The former is usually computed by the mean, median, or sum of the fitness in the collection. At the same time, the latter may vary according to the feature space, especially if it is discretized. If low-dimensional spaces are assumed, we can discretize the feature space arbitrarily and compute the percentage of bins filled. One may use density metrics in high-dimensional space, e.g., average distance between the k-nearest neighbors. Moreover, we may define a distance threshold between descriptions, computing a similar filling percentage as in the low-dimensional space case.

The MAP-Elites approach discretizes the feature vector into a grid or multidimensional array. Each cell in the grid corresponds to one type of solution, a location in the feature vector. The algorithm aims to fill every grid cell with the best possible solution. It takes inspiration from EAs, as in each iteration, the MAP-Elites alter copies of solutions in the grid to form new ones by applying mutation and crossover. New solutions are evaluated and, if better than the one occupying a cell, placed in there. Algorithm 2 show the basic loop for a MAP-Elites approach, as seen in (CHATZILYGEROUDIS *et al.*, 2021). We also recommend this material for further reading into QD approaches and MAP-Elites.

**Algorithm 2** – MAP-Elites Algorithm, as seen in (CHATZILYGEROUDIS *et al.*, 2021)

1: **procedure** MAP-ELITES($[n_1, ..., n_d]$)
2: $A \leftarrow create_empty_archive([n_1, ..., n_d])$
3: **for** $i = 1 \leftarrow G$ **do**
4: $\theta = random_solution()$ $add_to_archive(()\theta, A)$
5: **end for**
6: **for** $i = 1 \leftarrow I$ **do**
7: $\theta = selection(A)$
8: $\theta' = variation(\theta)$ $add_to_archive(()\theta', A)$
9: **end for**
10: **return** $A$
11: **end procedure**
12: **procedure** $add_to_archive(\theta, A)$
13: $(p, b) \leftarrow evaluate(\theta)$
14: $c \leftarrow get_cell_index(b)$
15: **if** $A(c) = null || A(c).p < p$ **then**
16: $A(c) \leftarrow p, \theta$
17: **end if**
18: **end procedure**

## 2.7 Path-Finding Algorithms

Path-finding algorithms find the shortest path between two given points, usually in a graph or tree structure. Its basic routine consists of exploring a graph from an initial vertex until destiny is found, generally searching for the path with the lowest cost. The path-finding problem can be solved using simpler algorithms such as breadth-first search and Depth-First Search (DFS) within complexity $O(|V| + |E|)$, being $V$ the number of vertexes and $E$ the number of edges. There are shortest-path algorithms that demand more computational complexity, e.g., *Bellman-Ford* algorithm with $O(|V||E|)$ complexity. *Dijkstra*'s algorithm takes $O(|E| + |V|log|V|)$ and uses dynamic programming to eliminate paths and Fibonacci stack in its implementation (FREDMAN; TARJAN, 1984).

Next, we describe the $A*$ and the DFS algorithms used to navigate the dungeons generated in our research.

### 2.7.1 A* Algorithm

The $A*$ is a variation of *Dijkstra's* algorithm and widely used in video games. The algorithm gives a weight to each open node equal to the edge leading to that node, plus the approximated distance of the current node to the goal. A heuristic function provides such a measure that allows the algorithm to discard the longest paths after finding an initial route, making the $A*$ more efficient than *Dijkstra* 's algorithm. The choice of the right heuristic function is vital once the closer it is to the real distance, the algorithm will evaluate less not-optimal nodes. Thus, the most efficient heuristic for an $A*$ algorithm is one that can give the actual

distance between nodes. Unfortunately, this demands a lot of computation time (MATHEW, 2015), so an admissible heuristic is chosen. In this case, a function that never overestimates the cost of reaching the goal is applied. The relatively easy implementation and efficiency of A* have become widely used in games. However, the game environments are complex and contain numerous characters that must execute the algorithm simultaneously. Thus, many optimizations for the algorithm, search space, heuristic function, and memory use are needed. The work of Cui and Shi, "A*-based Path-finding in Modern Computer Games" shows some of those optimizations and many benefits of A* in games (CUI; SHI, 2011). The algorithm's time complexity will depend on the heuristic to estimate the remaining distance. In the worst case, it is $O(b^d)$, where $d$ is the depth of the solution, and $b$ is the average number of successors per state, that is, the branching factor.

We now present an example of the $A*$ algorithm to clarify it for the reader. An algorithm implementation that can be executed from the browser and accepts different configuration inputs can be found in GitHub[14]. In Figure 5, it is possible to observe four steps of the $A*$'s execution in the same map. This map consists of a green tile representing the starting point, a red one that is the goal, and blue ones standing for walkable tiles. In the upper left image, all neighboring cells from the starting nodes are highlighted in green, with arrows pointing to the direction of the object's movement from the starting cell to the others. The set of tiles highlighted in green is called "open" tiles, and it is evaluated at every iteration while searching for the best route.

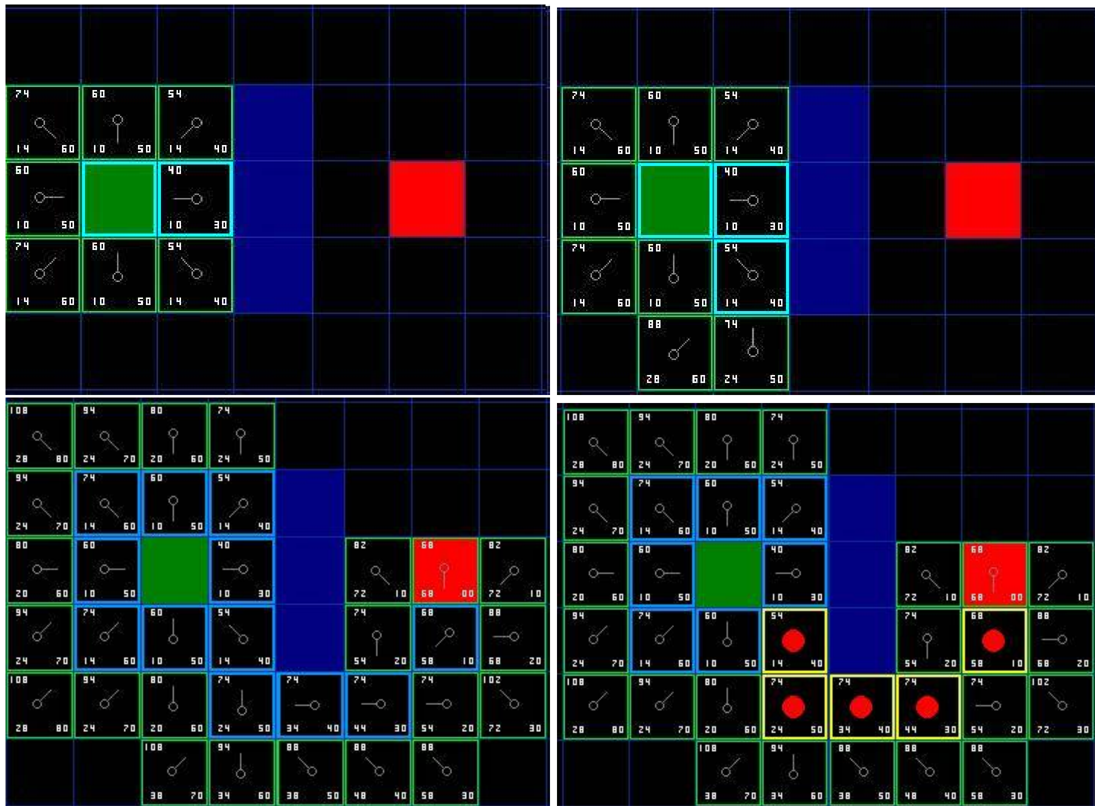
The three values shown in each tile are usually referred to as $F$, $G$, and $H$. The value $G$, shown in the bottom-left corner of each square, is the cost needed to move from the starting node (green) to the current tile, following the route generated by the algorithm to reach the tile. The value in the bottom-right corner, labeled as $H$, is the estimated cost given by the heuristic of moving from the current tile to the objective (red). The value $H$ predicts the remaining distance to reach the goal. At last, the upper-left corner of the tiles contains the value for $F$, which is the sum of $G$ and $H$. The green-highlighted tile, with the lowest value for $F$ in each iteration, is chosen to be part of the current path.

In the upper left image from Figure 5, we can observe the tile standing on the right side of the starting node, highlighted in cyan. Thus, it was chosen the tile with the least distance to the goal (lowest value of $F$), in this case, 40. This tile is added to the "closed" nodes (hence the cyan highlight). In the upper right image, we can observe that the path has been altered: the node closed in the previous step did not connect to any not-opened node. The lack of connection can be seen in the arrows, where none originated. These arrows show the movement direction leading to the shortest path from the starting point, the path which gives the value of $G$.

When moving to one of its neighbors apart from the current $G$, a visited node can hold a lower value of $G$. In this case, we update the neighbor's $G$, and the direction of the arrow in this neighbor node. Following the algorithm's execution, we can see that the goal has been reached

[14] <https://qiao.github.io/PathFinding.js/visual/>

Figure 5 – A* execution example. Extracted from: (LESTER, 2005).



in the bottom left image. The path is shown in the bottom right image, following the direction of the arrows starting from the goal tile (in red) until to reach the starting tile. By doing this, we find the path is composed of five tiles with a red dot in its center, and the shortest distance is 68 units of measurement.

### 2.7.2 Depth-First Search

The Depth-first Search is an algorithm that traverses or searches a tree or graph data structure. It starts at the root node for trees or a selected starting node in graphs and explores as far as possible along each branch before it backtracks. The algorithm takes linear time in the size of the graph $O(|V|+|E|)$. If there are multiple nodes to be chosen in the actual branch, the algorithm may follow a convention or visit a node randomly (CORMEN, 2008). Algorithm 3 describes a recursive implementation of DFS. The algorithm has some interesting applications in the creation and solution of mazes, as it guarantees all nodes are visited (SYEED *et al.*, 2010; KOZLOVA; BROWN; READING, 2015). For huge or infinite trees and graphs, the original DFS may not work properly, but some adaptations, such as the Depth-First Iterative-Deepening, can be used (KORF, 1985).

**Algorithm 3** – Recursive DFS. Adapted from: (CORMEN, 2008)

**Require:** Graph $G$ and vertex $v$ of $G$
1: **function** DFS($G$, $v$)
2: label $v$ as discovered
3: **for** edges from $v$ to $w$ in $G$.adjacentEdges($v$) **do**
4: **if** vertex $w$ is not labeled as discovered **then**
5: DFS($G$, $w$)
6: **end if**
7: **end for**
8: **end function**

## 2.8 Markov Chain

A Markov chain is a stochastic process with a finite number of possible values $X_n$, and whenever the process is in state $i$, there is a fixed probability $P_{ij}$ that the next state will be $j$. Equation 2.1 shows this probability for all states $i_0, i_1, ..., i_{n-1}$ and given that $i, j$ and all $n$ are non-negative. The probability of any future state is independent of past states and depends solely on the present state.

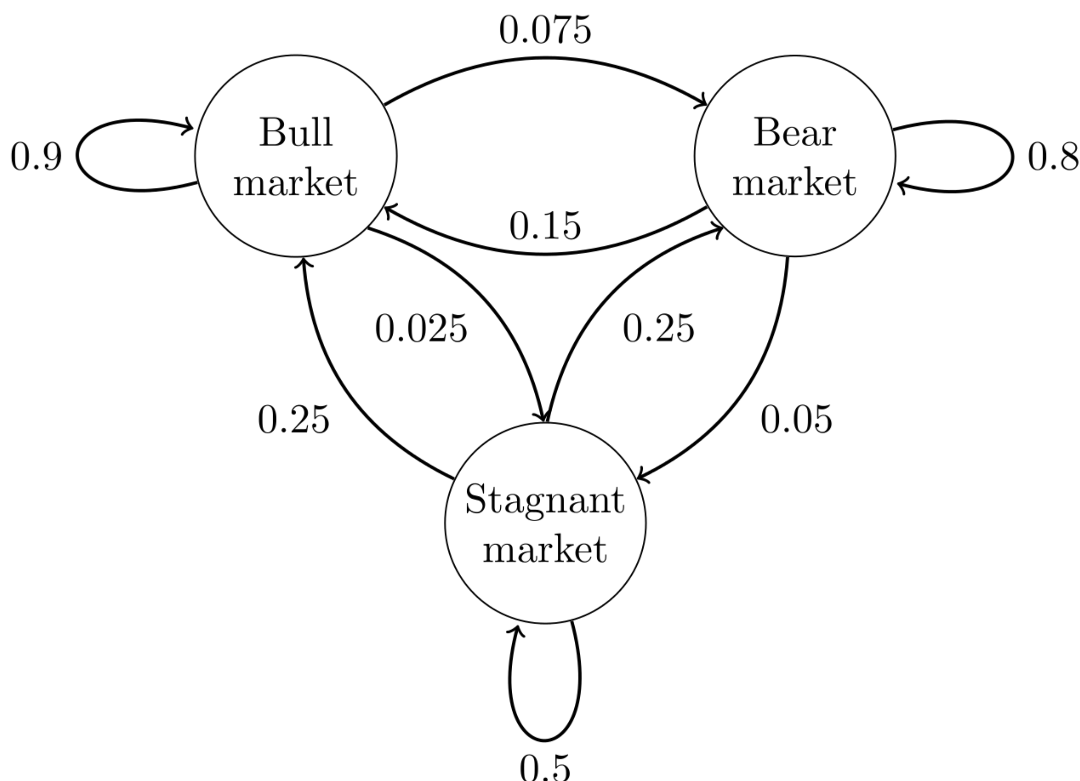
$$P\{X_{n+1} = j | X_n = i, X_{n-1}, ..., X_1 = i_1, X0 = i_0\} = P_{ij} \tag{2.1}$$

The value $P_{ij}$ is the probability that the process will transition into state $j$ when in state $i$. As the probabilities are non-negative and the process must transition into a state, we have $P_{ij} \geq 0$, $i, j \geq 0$, and that $\sum_{j=0}^{\infty} P_{ij} = 1, i = 0, 1, ....$ We refer (ROSS, 2019, Chapter 4) for more details. Figure 6 shows an example of a Markov Chain representing a hypothetical market and the probability for it to transition to the next state $j$ when it is in a particular node in state $i$. For example, we have a 50% chance for the market to remain in the *Stagnant market* when it is already in this state. However, there is an equal probability of 25% for it to transition to either the *Bull market* or *Bear market* states.

The ability to model the transition of states considering the current state (and, sometimes, past states when we have Markov Chains with steps greater than one) has made the Markov Chain attractive to some Procedural Content Generation applications. The authors in (SNODGRASS; ONTAñóN, 2017) create levels for the Super Mario Bros. and Kid Icarus games using Markov chains combined with a series of restrictions. (LI *et al.*, 2021a) uses an ensemble of Markov Chains to create *Mega Man* levels. They are also used to represent stories, where each element is sampled from a distribution of possible bigrams of words that can be in the story (HARRISON; PURDY; RIEDL, 2017).

In our project, Markov Chains provide the probability of specific rules in our grammar being selected over other rules. We next describe formal grammars (Section 2.9) and their stochastic approach, which may use Markov Chains, as we did.

Figure 6 – Markov Chain example from a hypothetical market as seen in <https://en.wikipedia.org/wiki/Examples_of_Markov_chains>. The directed graph's nodes are states of the stock market in a given week, and the edges are the probability of what the next week will be. The sum of the edges' weights exiting a given node is 1.0.




## 2.9 Formal Grammar

Formal grammar, from the formal language theory, describes how to create valid strings following a language's alphabet and syntax. We define grammar through a set of production rules in a language. We briefly describe these grammars in this section, but more details can be seen in (MEDUNA, 2014), as this section is adapted from their work.

An alphabet $\Sigma$ is a finite nonempty set of *symbols*. A nonempty subset of $\Sigma$ is a *subalphabet* of it. And a finite sequence of symbols from $\Sigma$ is a *string*, where $\varepsilon$ is an *empty string*, with zero symbols. These languages can be natural languages (e.g., English, Japanese, Portuguese, etc.) or artificial, like programming languages.

With a pair of an alphabet and a finite relation on the nonempty alphabet ($\Sigma^*$), which we call $R$, or the set of rules, we have a *rewriting system* $M$ such that $M = (\Sigma, R)$. For every $u, v \in \Sigma^*$, if there exist $(x, y) \in R$ and $w, z \in \Sigma^*$ such that $u = wxz$ and $v = wyz$, we say there is a *rewriting relation* over $\Sigma^*$ denoted by $\Rightarrow$ and, in this case, shown as $u \Rightarrow v$. Usually, each rule $(x, y) \in \Sigma^*$ in a rewriting system $M = (\Sigma, R)$ is written as $x \rightarrow y$ for abbreviation purposes. A rule is often labeled with a letter as in $r : x \rightarrow y$. Instead of $r : x \rightarrow y \in R$, we usually abbreviate it as $r \in R$. In the previous example, $x$ is the left-hand side of $r$, and $y$ is the right-hand side of r.

A sequence of rules for a *rewriting system* has each rule named, and when a given rule is used to rewrite a symbol, we denote the rule after the relation inside square brackets: $[r_1]$. We may also underline a substring rewritten during a rewriting step for clarity. Figure 7 translates the string "aab" using a system $M = (\Sigma, R)$, which rewrites from an alphabet $\Sigma = \{a, b, x, y\}$ to another string in the same alphabet, through rules $R = \{r_1 : a \to x, r_2 : b \to y\}$.

Figure 7 – Example of the rewriting system $M = (\Sigma, R)$, $\Sigma = \{a, b, x, y\}$ and $R = \{r_1 : a \to x, r_2 : b \to y\}$ translating the string "aab".

$$\begin{aligned} \underline{a}ab &\to x\underline{a}b\ [r_1] \\ &\to xx\underline{b}\ [r_1] \\ &\to xxy\ [r_2] \end{aligned}$$

We may say that a *rewriting system* $M$ generates or accepts a language $L(M)$, such that $L \subseteq \Sigma^*$ is a language over $\Sigma$, that is, a subset of it. A language generated by $M$ is the set of all strings $y \in F$, being $F$ the final language, such that $x \Rightarrow^* y$ in $M$ for some $x \in S$, with $S$ the starting language, and $\Rightarrow^*$ is the transitive and reflexive closure of $\Rightarrow$. The reflexive closure of a binary relation $R$ on a set $X$ is the smallest reflexive relation (relates every element of $X$ to itself) on $X$, which contains $R$. While the transitive closure of the relation is the smallest relation on $X$ which contains $R$ and is transitive, i.e., given $\{a, b, c\} \in X$, whenever $R$ relates $a$ to $b$ and $b$ to $c$, then $R$ also relates $a$ to $c$.

On the other hand, the language generated by $M$ is the set of all strings $x \in S$ such that $x \Rightarrow^* y$ in $M$ for some $y \in F$. In this context, $M$ is the *language-accepting model*. A parenthesis-generating model can be defined as $M = ({}_M\Sigma, {}_M R), {}_M\Sigma = \{(,)\}$ and ${}_M R = \{\varepsilon \to x\}, x \in_M \Sigma^*$. If $S = \{\varepsilon\}$ and $F =_M \Sigma^*$, the language generated by $M$ is: $L(M) = \{t | s \Rightarrow^* t, s \in S, t \in F\}$, that is: $L(M) = \{t | \varepsilon \Rightarrow^* t, t \in \Sigma^*\}$.

So, the set of all rewriting systems (the alphabets) consists of $\{(,)\}$, and each rule has $\varepsilon$ in the left-hand side. So, we can consider a grammar $X = ({}_X\Sigma, {}_X R)$, able to create the parentheses, with rules ${}_X R = \{\varepsilon \to ()\}$. We may denote grammar properties, such as the alphabet $\Sigma$ or the rules $R$, as belonging to a specific grammar by adding the grammar's name subscripted before it, as shown before with the results from grammar $X$, ${}_X R$. An example of a string generated by such grammar is $\varepsilon \Rightarrow ()\ RightArrow()() \Rightarrow ()(())$, such that $()(()) \in L(X)$. So $L(X)$ consists of all properly nested parentheses.

As for the acceptance of languages, we may create a parenthesis accepting language model as $M = ({}_M\Sigma, {}_M R)$, where ${}_M\Sigma = \{(,)\}$ and ${}_M R = \{x \to \varepsilon\}, x \in \Sigma^*$. If the starting language is $S = \Sigma^*$ and the final language is $F = \{\varepsilon\}$, the language accepted by M is $L(M) = \{s | s \Rightarrow^* \varepsilon, s \in \Sigma^*\}$. The grammar which denote such system can be defined as $Y = ({}_Y\Sigma, {}_Y R)$, such that ${}_Y\Sigma = \{(,)\}$ and ${}_Y R = \{() \to \varepsilon\}$. As an example, $Y$ accepts $(())()$: $(())\underline{()} \Rightarrow (\underline{()}) \Rightarrow \underline{()} \Rightarrow \varepsilon$.

We usually call language-generating models as grammars. Let one of these grammars be denoted as $G = ({}_G\Sigma, {}_G R)$. The symbols in $L(G)$, the language from said grammar, are referred to as *terminal symbols*, or *terminals*, sometimes denoted as ${}_G\Delta$, such that ${}_G\Delta \subseteq_G \Sigma$. The *nonterminal symbols* or *nonterminals* are ${}_G N =_G \Sigma -_G \Delta$. The start language of $G$ always consists of a single symbol, meaning ${}_G N$ (the nonterminals) contains a special *start symbol*, denoted by ${}_G S$. Therefore, ${}_G\Delta \subset_G \Sigma$. In the grammar rules ${}_G R$, at least one nonterminal occurs on the left-hand side of every rule. If $v \Rightarrow^* w$ in $G$, $v, w \in \Sigma^*$, we say that G makes a derivation from $v$ to $w$.

These grammars have many applications in computer science, but our focus is their applications for generating content. We show some applications in Chapter 3. We also define a specific type of generating grammar that uses probabilistic models to select rules. Stochastic grammars add variability to the rules chosen, be it a fixed weighted number generation, as in Pac-Man's ghosts, or an AI-based weighted for the number of generation, which recalibrates the weights using utility theory (LEWIS, 2013).

Probabilistic grammars define a probability distribution over a structured object through a step-by-step stochastic process. Hidden Markov Models are examples of such grammars, a random walk through a probabilistic finite-state network that samples an output symbol at each state. They define the joint probability of a string $x$ and a grammatical derivation $y$, presented in Equation 2.2, as defined in (COHEN; GIMPEL; SMITH, 2008).

$$p(x,y|\theta) = \prod_{k=1}^{K}\prod_{i=1}^{N_k}\theta_{k,i}^{f_{k,i}(x,y)} = exp\sum_{k=1}^{K}\sum_{i=1}^{N_k} f_{k,i}(x,y) log\theta_{k,i} \tag{2.2}$$

In Equation 2.2, $f_{k,i}$ is a function counting the number of times along the derivation the $ith$ event of the $kth$ distribution occurs. $\theta$ is a collection of $K$ multinomials, where the $kth$ includes $N_k$ events.

## 2.10 C4 Model

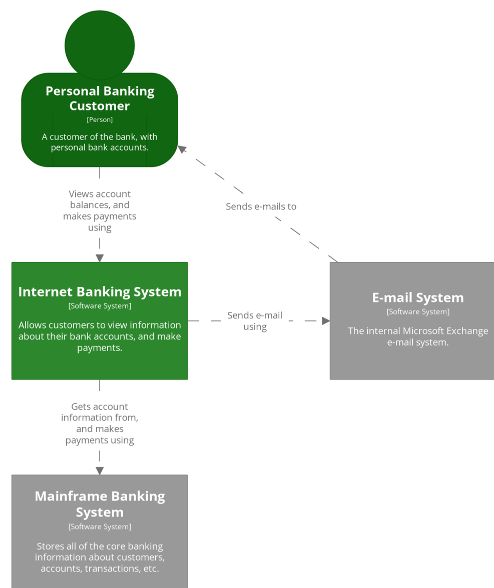
C4 is a model for software architecture created by Simon Brown, using a lean graphical notation technique. Its core principle is decomposing a system into smaller parts so that each person interested in the project can see the model with the correct level of abstraction for their purpose. We summarize its principles here, but more details can be seen in the model's website[15], the author's book on the subject (BROWN, 2022), and other software architecture books, as in (RICHARDS; FORD, 2020, Chapter 21).

Using this model, developers can create four viewpoints of their projects according to their hierarchical level. They are, from most to less abstract: context diagrams, container diagrams, component diagrams, and code diagrams.

[15] <https://c4model.com/>

The most abstract, the system context diagram, shows the system, the users, and other systems interacting. The focus is on what non-technical people want to understand about your system. Figure 8 shows an example of such a diagram, focusing on an internet banking application.

Figure 8 – A system context diagram from a bank system, as shown in C4 model's website <https://c4model.com/>. The Internet Banking System is the system of interest, while the customer is the main actor, but the banking system also interacts with the E-mail System and the Mainframe Banking System.




Then, we can use a container diagram to look inside a system and what containers are there. These containers can be server-side web applications, mobile apps, database schemas, etc. Usually, a separately runnable/deployable unit executes code or stores data. The diagram shows the overall architecture's shape and how the responsibilities are distributed. It also presents core technology choices and communication between containers. Figure 9 shows an example of such a diagram for the internet banking system.

Each container can be decomposed in different component diagrams, identifying the major structural building blocks and their interactions. It shows the container's components, responsibilities, technology, and implementation details. Figure 10 illustrates such a diagram for an internet banking system's API Application container.

The lowest level of abstraction is the code diagram, which is a UML class diagram. These diagrams are usually created via software programs that use the current codebase to build the visualization. This visualization is not recommended except for the most critical or complex components. Figure 11 shows an example for the Mainframe Banking System Facade class.

Given the 4 views of the C4 Model, we will focus in presenting our system through the

Figure 9 – A container diagram from a bank system, as shown in C4 model's website <https://c4model.com/>. The system consists of a web application, visited by the user, a single-page application with the functionalities, or a mobile app that the user may also visit. Both applications call the API Application, which sends emails using the E-mail System, makes API calls to the Mainframe Banking System, and uses data from the Database.

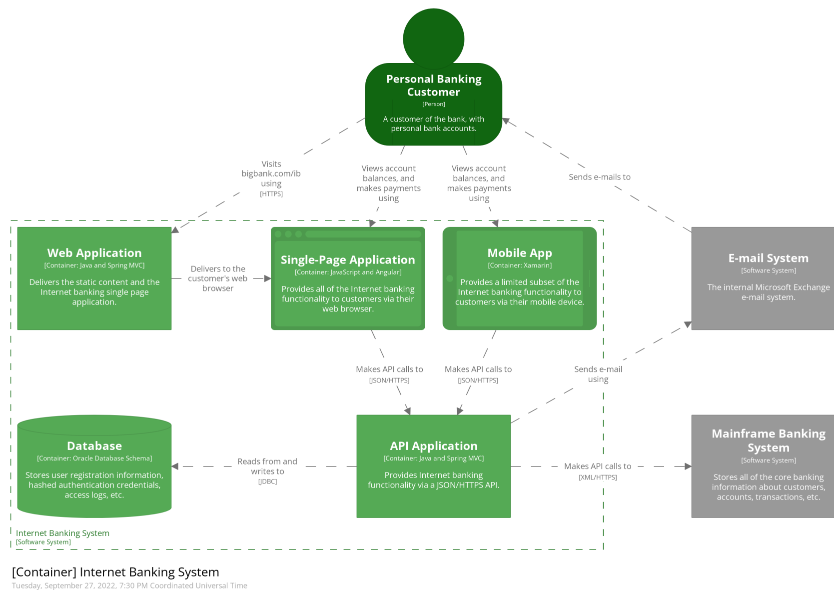
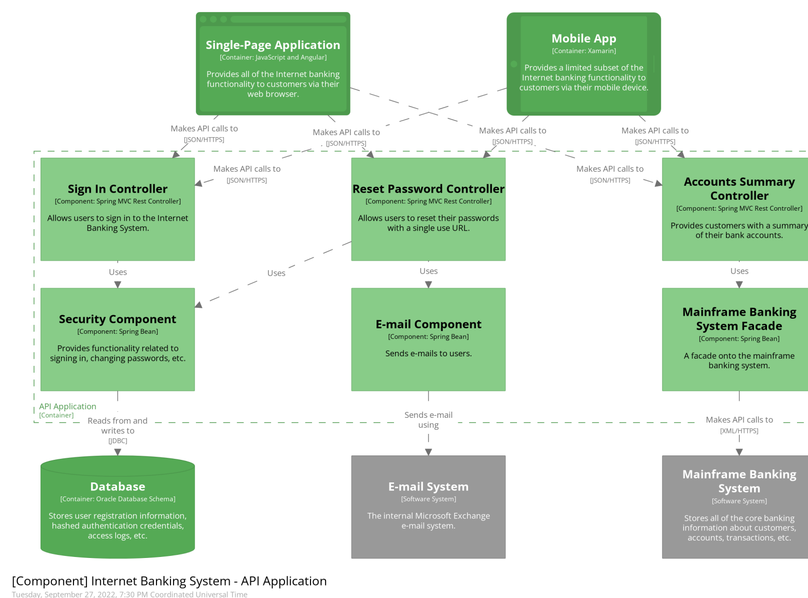
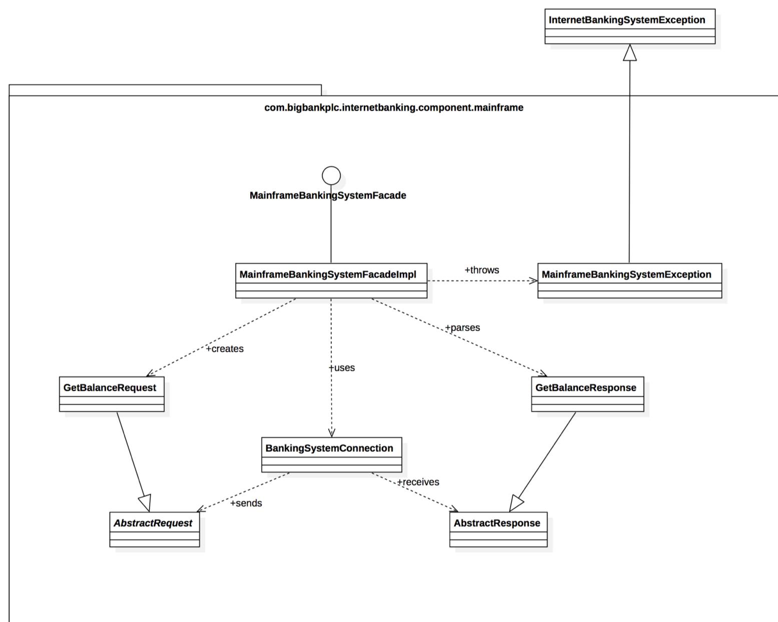



Figure 10 – A component diagram from a bank system's API Application, as shown in C4 model's website <https://c4model.com/>. The container consists of a Sign In Controller, which uses data from the mobile or single page Applications to sign in a user. The Security Component check the data, as well as data from the Reset Passwords Controller component, which also gets data from both applications. The Reset Password Controller also uses the E-mail Component to send e-mail to users. The applications can make API calls to the Accounts Summary Controller component, which, in turn, uses the Mainframe Banking System Facade, that calls the Mainframe Banking System.




view of the 2 most abstract ones: the context and the container diagrams. The context diagram will show how each system interacts with each other and how the users interact with them.

Figure 11 – A code diagram (UML's class diagram) for the Mainframe Banking System Facade class, as shown in C4 model's website <https://c4model.com/>. The class has an implementation (which shows it is an interface, and not a class). This implementation may throw exceptions, may parse the balance response from the banking system, may create a balance request, and, finally, use the banking system connection.




The latter will be used to show how each system is organized and how their containers interact between themselves, specially clarifying the data passage and dependence between them. For brevity, the component and class diagrams will be omitted.

## 2.11 Final Remarks

This chapter presented the main concepts and techniques used in this thesis that are necessary for the reader to understand for a better comprehension of the project. Some of them were related to game development concepts: the definition of the action-adventure genre and some games belonging to it, what are dungeons in the perspective of game design and the main game engines (Section 2.1 through 2.3). Thus, we learned that Unity was used as it is one of the most common, it is easy to learn, free in our usage scope, and has a strong community of developers helping each other. We also learned that the action-adventure genre focuses both in the physical challenges a player must face and the exploration of the virtual world, which is usually divided in different areas that contain dungeons or are the dungeons themselves.

Then, we learned more about the main theme of the research: the procedural generation of content. Section 2.4 covered the origins of the technique, as well as some of their best known applications in the industry, which the reader may be familiar with. The research on PCG will be covered in Chapter 3.

Next, the main algorithms used throughout the thesis were presented: EAs, which were used both for dungeon and enemy generation, is presented in Section 2.5, and are a heuristic technique which uses evolutionary pressure, based on a fitness function, and recombination of solutions to search for a local or global optimal answer for the problem. Section 2.6 showed the MAP-Elites, an algorithm that instead of searching for a single best solutions, provides a set of good solutions, usually using an EA to search for said solutions. As we use path-finding algorithms as part of our fitness function for the dungeon generator, we also provide a brief explanation of these algorithms, specially the ones we used, A* and DFS, in Section 2.7. These algorithms search for the best path between a source and a destination, although the A* uses a heuristic to estimate the best path, while the DFS explores in a predetermined manner, showcasing behaviors of different players traveling a dungeon.

Markov Chains, a stochastic process to calculate the probability of transitioning states in a system, were presented in Section 2.8 and Formal Grammars, a set of production rules in a language, used to form strings that represents the quests from our generator, were presented in 2.9. Markov Chains were used to set the weights of selecting a new rule of our quest-generating grammar, considering the previous state of the grammar. Finally, Section 2.10 shows the C4 model, used for software architecture, specially to represent large systems in different levels of abstractions. The higher levels of abstraction are used in this thesis to show the context in which our systems are used and the containers inside each system, that is, their principal working parts and how they communicate between themselves.

CHAPTER

3

# LITERATURE REVIEW

The present section reviews recent works of PCG in video games and other areas that are related to our research. We first review some works on gathering data from players and adapting content according to different players' preferences in Section 3.1. Then, we start discussing works on the contents we generate in this research: dungeons (Section 3.2), enemies (Section 3.3), and quests (Section 3.4).

Next, as one of our main goals is the orchestration of multiple contents, we discuss related works which performed such orchestration, adapting to different users' needs or not, in Section 3.5. One of our main objectives was to create a reusable, modular, and adaptable system. We then discuss in Section 3.6 works on PCG approaching software architecture, highlighting the importance of suitable architectures. Finally, we present in Section 3.7 a summary of how our work relates to the closest ones in the literature.

## 3.1 Player profiling and content adaptation

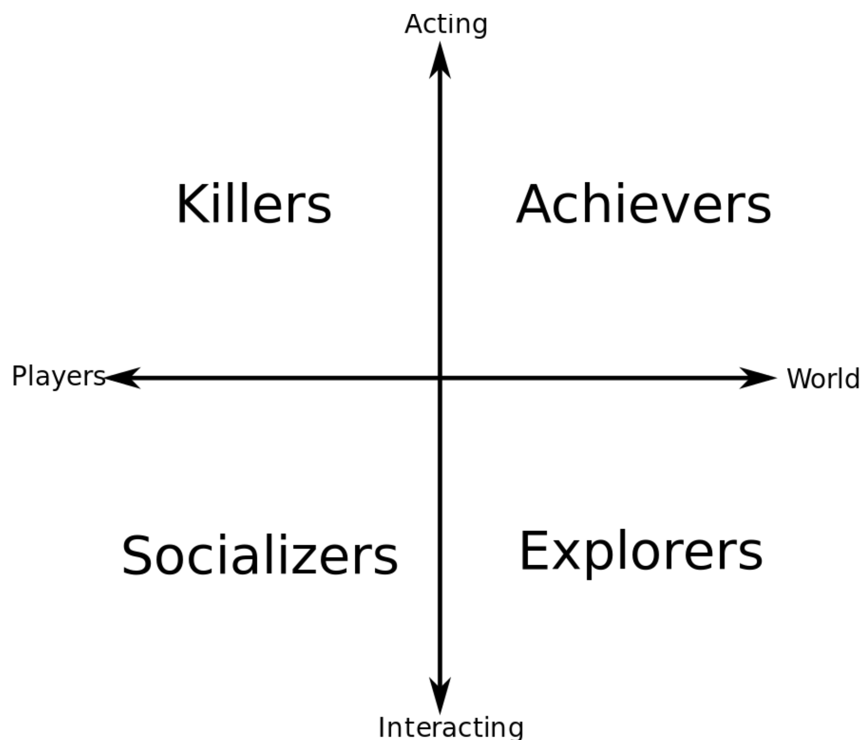
We present and discuss some relevant works in gathering data from players and trying to adapt content for them in this section. When discussing *Adaptive Interfaces and Agents*, (JAMESON, 2002) establishes some essential notions about how we can obtain information from users effectively to understand a system's usability better and adapt it to them. As our main goal is to adapt games to users by collecting usability data, these concepts are essential to obtain critical information. We can gather data using explicit input via self-reports, self-assessments, and non-explicit input.

The self-reports gather objective personal characteristics of the user, such as age and profession. The self-assessments are about gaining insight into how much the user is interested in a topic, their level of knowledge about it, or even the importance they give to some evaluation criterion. Self-reports may also be about specific evaluations, where the user evaluates the items

of a system they interacted with, the actions the system performed, or actions done by others. Finally, the user can also have their skill or knowledge evaluated by responding to test items. The non-explicit input can happen by collecting the user's actions on a system, e.g., using sensors to get data from the user's body or surroundings.

As for works focused on the field of personality traces and profiles in video-game, the first widely known attempt to classify players was done in the 90s by Bartle (BARTLE, 2003). The research is based mostly on Multi-User Dungeon (MUD) and Massive Multiplayer Online Role Playing Game (MMORPG) games. The author divided players into four categories, following two axes. Players may prefer to *Interact* with or to *Act* on the *Game World* or *Other Players*; thus each quadrant produces one archetype. Figure 12 shows the quadrants.

Figure 12 – Bartle player types chart. Source: <https://en.wikipedia.org/wiki/File:Character_theory_chart.svg>




The *Achievers* liked to *Act* in the *Game World* to gain points, levels, equipment, and anything measurable. Another group that liked to *Interact* with the *Game World* is the *Explorers*: they wanted to discover areas and immerse themselves in the world, often disliking time-restricted missions. On the other hand, the *Socializers* liked to *Interact* with *Other Players* and even Non-Player Character (NPC) avatars, playing for the social aspect. Finally, the *Killers* were players who wanted to *Act* with *Other Players*, but through competition and combat, testing their skills. This taxonomy expanded later to include subdivisions of those archetypes, but, in its essence, it remains the same. The data was collected from explicit answers from players about the game.

Since the studies of (BARTLE, 2003), many others have contradicted each other, even when trying to create similar categories in the same game genre, and considering only explicit

data. For instance, Yee's approach on Massive Multiplayer Online (MMO) players' motivations (YEE, 2007) reports a survey with 3200 respondents from players. Their research found that grouping players by their motivations produced better results than dividing them into exclusive categories. They found three main motivation components: achievement, social, and immersion. Each had subcomponents: advancement, mechanics, and competition for achievement; socializing, relationship, and teamwork for social; and discovery, role-play, customization, and escapism for immersion. One exciting discovery was that they were not exclusive, i.e., a player could be motivated by both the achievement and the immersion components by a similar amount.

In more recent approaches, Vahlo et al. (VAHLO *et al.*, 2017a) made a new classification based on thousands of player preferences, first qualitatively analyzing 700 written reviews for digital games. They identified specific contents and keywords and grouped the findings into 33 core game dynamics. Next, they created a questionnaire with these 33 items, where the user had to answer on a 7-point Likert scale how pleasant they found each dynamic. The work clustered 2594 respondents in 7 groups by using a factor analysis that grouped the 33 dynamics in 5 factors:

- *Assault*: preference to kill, destroy, shoot, run, etc.
- *Manage*: likes to acquire resources, expand buildings, manage resources, upgrade items, etc.
- *Journey*: fascinated by exploring, uncovering secrets, making decisions, making in-game friends, etc.
- *Care*: likes taking care of pets and having romantic relationships with characters.
- *Coordinate*: attracted to puzzle-solving, precise platforming, rhythmic actions, and music-related things.

Each of the 7 clusters had factors they liked more, disliked more, or were neutral. Overall, four of them were interested mainly in one of the dynamics (except *Journey*) while the other three preferred mixes of two dynamics (*Journey* and *Assault*, *Journey* and *Coordinate*, *Assault* and *Coordinate*). These findings corroborate Yee's work by showing that players usually have more than one preferred trait in a game, and a combination of motivations can better represent player types. Moreover, it also expands said work, as these archetypes encompass many games and players.

Some profiling strategies are also present in gamification applications, as in the work of (ORJI; TONDELLO; NACKE, 2018). They experimented with 543 volunteers to investigate the relations between user types (in this case, for gamification software) and persuasive strategies. The latter are techniques meant to persuade the user to change their behavior. The authors used the Hexad player types, created for gamification purposes, which captures users' motivations and

styles of interaction, dividing them into six groups. They investigated how each type compares to preferences in persuasive strategies. Following (TONDELLO *et al.*, 2016), the types in Hexad are:

- *Philanthropists*: motivated by purpose, altruistic and do not expect rewards.
- *Socializers*: motivated by relatedness, likes creating social connections and interaction.
- *Free Spirits*: motivated by autonomy and freedom of expression and action, like creating and exploring.
- *Achievers*: motivated by competence, likes progressing through tasks and tackling challenges.
- *Players*: motivated by external rewards or incentives, likes rewards, no matter from what).
- *Disruptors*: (motivated by change, like testing boundaries and disrupting while seeking changes, no matter if positive or negative.

In their experiment, they used ten storyboards to illustrate each of the ten persuasive strategies (*Competition*, *Stimulation*, *Self-monitoring and Feedback*, *Goal setting and Suggestion*, *Customization*, *Reward*, *Social Comparison*, *Cooperation*, *Personalization*, and *Punishment*). The work has a list of questions relating to these strategies, the storyboards, and 24 items of the Hexad types while collecting demographic and drinking behavior information.

The results from (ORJI; TONDELLO; NACKE, 2018) show that *Socializers* are positively affected by any strategy. In contrast, *Disruptors* are negatively affected by most and positively affected by the *Competition* and *Customization* strategies. The *Player* type is positively influenced by *Competition*, *Reward*, *Comparison*, *Cooperation* and *Punishment*. *Achievers* were not influenced by any strategy, while *Free Spirits* were positively influenced only by *Personalization*, and *Philanthropist* only by *Simulation*. While these results point out that different users (players) may respond differently to engagement strategies based on their player profiles (or user types), it also shows that most types were not very useful for discriminating between users' preferences in such strategies.

However, none of the researchers collected in-game data to validate their findings empirically. We suppose that personalized content would have a positive effect when targeted to the right player profile (or type), as all these studies point in this direction. But they need to provide empirical evidence to affirm this. We cannot even report a specific questionnaire or implicit data-collecting strategy as the best one to separate these players into their right types, once no experiment yielded conclusive results.

Trying to test those theories inside a game environment further, we present Melhart et al.'s work(MELHART *et al.*, 2019) with a team composed of both game researchers and

*Ubisoft*'s developers. They analyzed anonymous data collected from 298 players of the game *Tom Clancy's The Division* from 2016 to 2018, trying to predict their motivations and feelings. After collecting the data, they sent Ubisoft's UPEQ questionnaire (AZADVAR; CANOSSA, 2018) to the players, evaluating them according to self-determination theory.

A total of four player archetypes were created using k-means clustering with accuracy ratings neighboring 90%: *Adventurers*, new players which were still figuring out the game's systems and did not find their preferred niche of play; *PvE All-Rounders*, highly skilled players that liked going on cooperative PvE (Player versus Environment) missions; *Social DarkZoners*, players that preferred staying on the PvP (Player versus Player) areas of the game and interacting with each other; and *Elites*, players with very all-rounded preferences in the game.

One can see similarities between their research and Bartle's taxonomy, but there are some key differences, such as the lack of interaction axes and the way those archetypes are laid out. The data fed an artificial neural network with *Radial Basis Function* kernels, which learned some player preferences and behaviors. However, the neural network did not accurately predict more subjective aspects, and had issues with some characteristics in which players diverged in reported preferences and actual gameplay time. Thus, they concluded that it is possible to create player groupings and predict their tastes, but it is not an easy task and requires direct modeling and study. Therefore, we can observe the importance of the data collection and analysis processes when profiling players.

This analysis could lead to significant improvements in game design, especially with recent advances in real-time PCG, as the one presented in Bicho and Martinho's research (BICHO; MARTINHO, 2018). They show an interesting experiment by evolving an infinite runner's difficulty to match its player's skill better. The profiling was simple but ingenious: every obstacle had two ways to be avoided, jumping over it or using a dash skill that could traverse objects. Every time the player uses one to get past an obstacle, the next gets a little more challenging to avoid using the same mechanic. Thus, they could identify each player's favorite mechanic and adapt the content for each. A very interesting finding was that players preferred to continue using their favorite mechanic, even when it kept getting harder to use it, then changing to the other strategy was easier. The reported results can lead us to procedurally generated content for games with more complex mechanics. We can increase the complexity of a mechanic the player likes whenever they succeed, while keeping less liked mechanics simpler, aiming to increase their engagement.

Work with a similar approach is (KANTHARAJU *et al.*, 2022), where they trace player knowledge (the likelihood the player has mastery over a set of concepts or skills) in an educational game that teaches parallel and concurrent programming, collecting data from real users. Their player's model consists of a series of rules used to detect if the player successfully used each skill required to master the game. The model consists of a vector, where each element is the likelihood that the user mastered the corresponding skill. To verify the user's focus on different

gameplay moments, they use time windows to identify the applications of the skills in each one.

The game extracts features from gameplay data for each time window. Next, identify what skills the player was trying to apply using ML and a collection of domain knowledge rules for comparison. Finally, build the knowledge model of the player using the previous outputs. Their results show that the approach performed well, although not as well as using only Machine Learning. However, the system could also construct game levels that improved students' understanding of the taught skills, as the modeling system was able to follow the mastery trend from students.

Another approach, from (GRAY *et al.*, 2020), uses Multi-Armed Bandits to model players in a serious game to help users keep their motivation to exercise often. They first trained via a simulator of players, the Multi-Armed Bandits model, which would measure their social comparison orientation. This metric tried to maximize their motivation to exercise, as it consists of comparing themselves to others to assess their success, plan for future success or view themselves favorably.

The system focused on showing the player how they compared with others in the quantity of steps taken. The Multi-Armed Bandit would have as its "arms", or strategies to follow, one of three types of players: all other players shown to them would have walked less; two players who walked more and two that walked less would be shown for comparison; or only players that walked more would be displayed. The rewards for the model were how many steps were taken in the day and a 5-point Likert scale answer about their motivation. Both measures are compared to the average and standard deviation from previous sessions. Simulations evaluated many strategies. The best was applied in a study with real users. Their results show the model was able to adapt the strategy to the player, as they were more motivated than the control group (which received a random strategy). A follow-up experiment used a similar strategy with positive results when considering a group of players (GRAY; ZHU; ONTAñóN, 2021).

Although these projects show promising results by combining gameplay data with machine learning or specific rule sets, they are application-oriented, making applying their algorithms on different games difficult. We next show some works successfully using more general data collection (implicit and explicit) within games.

Heijne's research, both in (HEIJNE, 2016) and (HEIJNE; BAKKES, 2017) was interesting as they could link the benefits of collecting data to suggest a player profile for the current player and how to use it to guide the generation of multiple contents. They collected implicit and explicit data from people that played a clone from *The Legend of Zelda: a Link to the Past* that heavily relied on PCG techniques and could adapt the generated content based on the player's performance.

The game, called *Procedural Zelda*, was able to procedurally generate levels, place enemies in them and create puzzles, albeit with a somewhat limited extent to the diversity

and difficulty adjustment. The game also had many data logging capabilities: a demographic questionnaire, a personality test 120 questions long (both cases of self-report about objective personal characteristics), implicit data collected from the behaviors of the player when talking to NPCs, exploring levels, solving puzzles and fighting enemies, and a post-game questionnaire that asked about the difficulty overall and of each component, the behavior of the player and preferences over each component (self-reports on specific evaluations).

They collected data from 25 users and divided them into three groups: Hardcores (10 users), which played many hours a week and liked harder difficulty settings, Casuals (8 users), who preferred lower difficulties and spent less time gaming, and Ambiguous (7 players) that were somewhat middle ground between both categories. They also grouped people by the personality test and extensively analyzed correlations between the groups and personalities, as well as their performance. Previous experience in similar games and preferring harder difficulties were the strongest indicators for performance. Furthermore, they observed that performance metrics do not correlate with explicit preferences on mechanics, and overall experience with games correlated with better understanding and use of the mechanics. Finally, players who explored a lot started to lose interested in exploration over time and focused only in the main path. Heijne's work depends on a 120 questions personality test and other explicit data. The work was unable to give very accurate results about how much a player would like each content and what difficulty setting was better in each content for each category of players.

Trying to collect high-quality data from players with as few questions as possible, as massive questionnaires tend to make respondents give up or answer inaccurately ((JAMESON, 2002)), we also turned to Rivera-Villicana et al.'s work. They create a Belief-Desire-Intention (BDI) model of agency for players with only ten questions, semi-structured interviews, and gameplay data. They got data from 23 players and tested the models simulating player behavior in a point-and-click interactive fiction against an uninformed player model that did not try to mimic any player style. They found that the data collected from explicit player input was inaccurate compared to their implicitly gathered gameplay data; relying more on implicit data may be better. However, the model itself, created based on the questionnaire was effective when created using the implicit input, which means the questions themselves were reasonable measures to divide player styles (RIVERA-VILLICANA *et al.*, 2018). Other works that tried to identify profiles with small questionnaires and some in-game data only often present inconclusive statistical analysis (KONERT *et al.*, 2014; LORIA; MARCONI, 2018).

We present the work of Bontchev et al., where their experiment was able to use recognition of players' emotions and data from their gameplay to adapt content for different players after inferring their play-style (BONTCHEV; GEORGIEVA, 2018). However, modeling a player and their personality from their in-game actions is a matter of concern. We need some sensors to read the player's emotions, like cameras filming their faces.

More sophisticated techniques borrowing knowledge from psychology, like identifying

behavioral patterns (*Behavelets*) show a theoretical potential to create player profiles and adapt content adequately, but it is yet unproven. Moreover, it demands help from a specialist to complete the users' model (COWLEY; CHARLES, 2016).

Therefore, we have good evidence that solid profiling of players can lead to good content adaptation. Some have done so through rule sets specific to their games, others through machine learning approaches, and others by analyzing questionnaires and/or gameplay data in a more general approach, to help guide content generation. We use this information to develop some questionnaires to evaluate our content and get data from players to guide our multiple content generators, while also collecting gameplay data to evaluate players further.

## 3.2 Dungeon Generation

Level generation, as mentioned in Section 2.4, was the first application of PCG in games, e.g., Elite and Rogue (BELL; BRABEN, 1984; DESIGN, 1984). In our system, we create our dungeons (our level) using different approaches, all based on the EA first proposed in (PEREIRA *et al.*, 2021) and its MAP-Elites variation proposed in (VIANA; PEREIRA; TOLEDO, 2022). Therefore, we first focus on a dungeon generation overview, based on (VIANA; SANTOS, 2021)'s survey, and highlight some EA and MAP-Elites approaches for level generators, which are closely related to our work.

The survey in (VIANA; SANTOS, 2021) collected data from journals and conferences from 2010 to 2020, selecting 41 papers to analyze. Their findings show that search-based strategies are the most common, present in 17 types of research on the subject, where 16 are EA-based strategies. Some works have hybrid solutions that often use EA and another approach, while others present diverse methods using Machine Learning, solver-based solutions, grammar, and tailored algorithms. Grids usually represent levels (as seen in 23 papers), but sometimes as graphs (4 times), shape grammars (3), and trees (2 times), among others.

The majority of the solutions (30 papers) created 2D levels, and only five could generate 3D levels. There are 31 papers reporting methods that did not yield any levels with barriers (e.g., locked doors puzzles, as in our solutions), and only nine did so. Most papers (27) created top-down cavern-like dungeons like the ones in *Pokémon* games. Other eight works created top-down mansion-like dungeons (like the ones in *The Legend of Zelda*, which is the type of dungeon we make. Only one paper describes a side-scrolling dungeon, as in *Spelunky*.

Another relevant aspect reported in (VIANA; SANTOS, 2021) is that the reviewed papers described methods evaluated in an *offline* environment, which means no generation of levels in real-time following players' feedback. Our level generator can create online content with those features and be tested with players as described in Section 5.3. It is an advance over the surveyed generators. Only one work, (ALVAREZ *et al.*, 2018), had an adaptive generation, but adapted solutions in a mixed-initiative process.

In (Alvarez *et al.*, 2019), a continuation of the aforementioned dungeon generator with adaptive generation, a human designer selects the levels for evolution, and the system improves such levels based on the designer's intended aesthetic. Therefore, the generator does not adapt to different players. The authors use a mixed-initiative QD algorithm called the Constrained MAP-Elites algorithm. It applies a Feasible-Infeasible 2 Population genetic algorithm based on the fulfillment of constraints to create levels with items and enemies inside the rooms. They focused on a framework for designers, allowing input from humans to guide the evolutions.

Two EAs in (FONT *et al.*, 2016) evolve high and low-level representations for a dungeon based on context-free grammar. The high one generates an acyclic graph with interconnected zones and the number and types of objects in each room. The low-level algorithm executes for each zone, generating a cyclic graph containing rooms, their objects, and connections. The algorithm was tested through 50 runs, converging under 10s for all, but not tested with users. (LIAPIS, 2017) also uses an EA approach to create dungeons and missions inside them. A sketch, a high-level description of the dungeon, evolves using a human-provided fitness function, similar to our approach. Each sketch's tile guides the evolution of a segment, combining all segments into a whole dungeon at the end of the execution. The level contains treasures and enemies while fitting the designer's inputs, demanding many constraints to be satisfied.

Charity et al. (CHARITY *et al.*, 2020) propose a 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.

## 3.3 Enemy Generation

There are several works in the literature about the enemy generation problem, by considering their placement and amount. However, our generation will define their amount, status, visuals, and adaptation. By status, we mean features such as health points, damage, speed, and type of movements (e.g., attack behavior). The visuals are about how to portray the enemy in the game, e.g., the NPCs' sprites or models. The adaptation adjusts the enemy difficulty to each player's profile, e.g., easier enemies for levels where the players get stuck.

We found only algorithms generating smarter AIs for existing enemies, focusing only on specific movements. The closest related works are (Baldwin *et al.*, 2017; SHARIF; ZAFAR; MUHAMMAD, 2017; KHALIFA *et al.*, 2018), which approach the placement of enemies on levels and the generation of projectiles in a bullet hell game.

For the generation of enemies in commercial games, we found some examples that are a little closer to our idea. No Man's Sky (GAMES, 2016b), Spore (MAXIS, 2008), and the Creatures game series (GRAND; CLIFF, 1998) are good examples of games that create NPCs

procedurally with different techniques. Other games can change some predefined characteristics in enemies, becoming the challenge more diverse and unique, as Diablo 3 (NORTH, 2012) and Shadow of Mordor (PRODUCTIONS, 2014). Others decide the placement and number of enemies, adapting to the player's performance, as in Left 4 Dead 2 (VALVE, 2009) and State of Decay 2 (LABS, 2018).

The authors in (Baldwin *et al.*, 2017) propose a Genetic Algorithm (GA) to evolve the dungeon's rooms using game design patterns and metrics regarding the placement of enemies and treasures. The room is represented as a grid of tiles, which may contain tiles that represent an enemy. Regarding the enemies' role in the fitness function, it uses the difference between the enemies' density and the target value input by the user. The closest resemblance to our work is the mixed-initiative fitness function, which compares the solution's current fitness to the user's target fitness. Their results were positive, considering the diversity of solutions and convergence, but there was no experimental setup with players within a game environment. Similar work in (SHARIF; ZAFAR; MUHAMMAD, 2017) shows different strategies based on design patterns for the placement of sprites in a level, with some being harmful sprites (enemies or traps). These are taken from a pool of existing ones, but only their placement is approached, not the creation of new ones. The works of (LIAPIS, 2017) and (Baldwin *et al.*, 2017), discussed in Section 3.2, also present enemy placement in the dungeons they generate. In contrast, Baldwin's work defines the number of enemies in each dungeon.

The authors in (KHALIFA *et al.*, 2018) describe levels for a bullet hell game that consist mostly of bullets with different speeds and movement patterns. The bullets can be compared to a simplified version of an enemy for an action-adventure game. Talakat is a description language that creates bullet hell levels with bullet spawners. Each spawner has a parameter for what bullet it spawns, the speed and angle it receives, and the angle and speed at which the spawner rotates, among other features. Each parameter has a value within a range input by the designer as our approach for an enemy generation.

There is also a boss: a spawner with health (time on screen), a position to spawn, and whose spawners will spawn following a script. The authors use the MAP-Elites illumination algorithm combined with a Feasible Infeasible 2 Population Genetic Algorithm (GA) to evolve the level's solution. A* agent is applied to test each solution and provide a fitness value. The computational results show the algorithm creates levels with varying difficulty, but players did not evaluate the levels' quality.

To the best of our knowledge, we did not find an approach in scientific literature creating enemies with as many features as we propose here (see Section 4.3.3 for details). We found a generation of procedural NPCs with many properties in the game No Man's Sky and Spore, with a similar algorithm: a creature is created by randomly putting together different body parts. Although they do not have any input to create them based on the player's performance, they have some constraints as to not putting together body parts that can't match with another already

selected one. However, this constraint is mostly not to break the procedural animations of the NPCs (GAMES, 2016b; MAXIS, 2008).

The game series Creatures, although older, extends this concept, not only physically evolving NPCs, but also making them 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). The enemies in Left 4 Dead 2 and State of Decay 2 are somewhat also adapted to the player, not by changing their characteristics or features. Still, by deciding where to place them and if new enemies should be spawned (if the player is doing well) or if the game should spawn fewer enemies (if the player is not performing well), (VALVE, 2009; LABS, 2018).

With scientific and commercial examples in mind, we propose a new method to create enemies using some mixed-initiative designs from these papers as inspiration. The main idea is to develop diverse enemies, as in Spore and No Man's Sky, and the adaptation of them from Creatures. To do so, we proposed an EA able to evolve a population of diverse enemies that respect the designer's difficulty level. The method also created new enemies with a higher or lower difficulty to fit each player's performance, as reported in (PEREIRA; VIANA; TOLEDO, 2021). Another approach using MAP-Elites was also used for different experiments in this project, and it is reported in (VIANA; PEREIRA; TOLEDO, 2022). Both are better presented in sections 4.3.3 and 4.3.4, respectively.

Table 1 summarizes our findings and how our work compares to what we found in scientific and commercial examples. We present if the feature is procedurally generated (**Y**es or **N**o), and if it is not entirely procedural, we call it **P**artially generated. For example, Spore's, No Man's Sky's PCG is based on a pool of already existing characteristics, not creating any visuals in runtime, nor creating new status, only generating combinations. Creature's visuals are the same, although their status is adapted in real-time using a neural network algorithm. Diablo 3 is another option that draws randomly from an existing set of solutions. (KHALIFA *et al.*, 2018)'s status and visuals are also considered partial once they are only bullets, which are a simplified version of an enemy, but the generation is in runtime. The visuals change depending on the bullet type, which is the same approach proposed by us in Section 4.3.3.

## 3.4 Quest Generation

We also use quests generated by grammar as a frame for the other generators (dungeon and enemies) to work. The first approach applied simple grammar, replaced by stochastic grammar, with a Markov Chain deciding its probabilities in a second approach. Our quest-grammar-as-a-frame resembles Montfort et al.'s Slant (MONTFORT *et al.*, 2013), which creates a story based on multiple systems working on an initial story's "blackboard". When concluded, the story goes to the Harvester, which checks its completion and augments it. Next, the GRIOT-

Table 1 – Comparison between our work and the closely related ones in reviewed literature about enemy generation.

| Work | Placement | Amount | Status | Visuals | Adaptive | MAP-Elites |
|---|---|---|---|---|---|---|
| Spore (MAXIS, 2008) | × | × | × | ✓ | × | × |
| No Man's Sky (GAMES, 2016b) | × | × | P | P | × | × |
| Creatures (GRAND; CLIFF, 1998) | × | × | ✓ | P | ✓ | × |
| Diablo 3 (NORTH, 2012) | × | × | P | × | × | × |
| Left 4 Dead 2 (VALVE, 2009) | ✓ | ✓ | × | × | ✓ | × |
| State of Decay 2 (LABS, 2018) | ✓ | ✓ | × | × | ✓ | × |
| (KHALIFA *et al.*, 2018) | ✓ | ✓ | P | P | × | ✓ |
| (Baldwin *et al.*, 2017) | ✓ | ✓ | × | × | × | × |
| (SHARIF; ZAFAR; MUHAMMAD, 2017) | ✓ | ✓ | × | × | × | × |
| (LIAPIS, 2017) | ✓ | × | × | × | × | × |
| Our: (PEREIRA; VIANA; TOLEDO, 2021) | ✓ | × | ✓ | P | × | × |
| Our: (VIANA; PEREIRA; TOLEDO, 2022) | ✓ | × | ✓ | P | × | ✓ |

Gen system realizes symbolic representations, and the final system (Curveship-Gen) selects the content (microplans) and produces the final text.

Quests are game objects with tasks and rewards, intended for a player, and integrating the game's narrative (YU; GUZDIAL; STURTEVANT, 2021). Our quest approach is similar to Thue et al.'s work (THUE *et al.*, 2021) 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.

Our work uses grammars that were successfully used in other PCG contexts, like when Gellel and Sweetser (GELLEL; SWEETSER, 2020) created levels and quests by combining generative grammar and cellular automata. They applied the grammar to codify locked-door missions and rooms, while the automata created play spaces inside the level. Our grammar structure is more related to the work of Doran, and Parberry (DORAN; PARBERRY, 2011) 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.

Other authors used a Genetic Algorithm mixed with an Automated Planning approach to creating quests, in (LIMA; FEIJó; FURTADO, 2019). In this work, a quest manager (part of their game manager) handles the events and completion of the quest inside the game, interacting with the player. The individual of the Genetic Algorithm encodes a planning problem with a set

of atomic formulas, a set of planning operators, and the initial and goal states of the quest. The atomic formulas and planning operators are schematic and defined as part of the game world (the conceptual scheme of the domain). The operators are possible events during a quest, and the formulas are valid atoms used to describe states, goals, and operators. Therefore, only the initial and goal states may vary between individuals. They are reactive and express different ways of perceiving a situation and the impulse to change it, depending on character preferences.

Their fitness evaluates how well the quest resembles a story arc's tension given by the user. The user provides a vector of values, each corresponding to the desired tension in each moment of the story. The algorithm solves the planning problem for the individual, estimates the story arc tension, and rescales both arcs to the same interval (in case the desired story has a different number of plot points from the generated one). Then, it uses the number of plot points in the arc divided by the mean squared error between the desired and generated arcs as the fitness. The evolved quest is placed in a prototype of a 2D RPG game developed using the Löve 2D framework. First, a game designer evaluated the quests, and 34 students tested them, without knowing if they were generated by humans or by the algorithm. The results show that players had an almost 50% score on selecting the right answer for both cases (algorithm or human-made), meaning the quests were undistinguished.

Another quest generator is presented in (BREAULT; OUELLET; DAVIES, 2021), also using a planning approach. The quests are a set of actions performed to achieve a goal, using as input the world description (a set of facts like characters, locations, and items). The generation of quests follows the state of the world and the characters' preferences. The world's locations are interconnected, and the NPCs are associated with preferences (weights of actions), which are used to evaluate if the quest makes sense to be given by a specific NPC. The designer must define the places, items, and NPCs' preferences, and the planning algorithm tries to find an efficient plan for the goal given each action's costs.

The planning goal is a set of statements that must become true in the world state. These statements are a combination of predicates and an object (an agent, item, or location), e.g., "has baker wheat", meaning that the baker must have wheat for it to become true. Their system depended on the world's setup to provide variety for their quests, but it was able to create diverse quests if this condition was met. We use these predicates as the base of our quests: our grammar creates the quest type or their predicate (e.g., kill, get, talk). In our case, the subject and any extra information required is taken from a fixed set of states provided by the designer. It can be a list of possible enemies to kill, items to gather, etc.

In our approach, although taking inspiration both from (BREAULT; OUELLET; DAVIES, 2021) and (DORAN; PARBERRY, 2011), we use a grammar approach, evolving it later to a stochastic grammar, where the rules' weights are updated via a Markov Chain to reduce repetition of the same quest type in a quest line. Although we could not find any work in the literature using the same approach (be it in PCG or not), we found an interesting work applying Markov

Chains for procedural music generation (ROIG *et al.*, 2018), also for dungeon generation (LI *et al.*, 2021b). We also found other papers with stochastic grammars for diverse applications (LIN *et al.*, 2009), (VO; BOBICK, 2014), gilbert2007probabilistic, (PERCHY; SARRIA, 2009), and (GIRAUD; STAWORKO, 2015), and a few in PCG contexts (CHEN; GUY, 2018) and (MARTINOVIC; GOOL, 2013).

(ROIG *et al.*, 2018) uses a non-homogeneous Markov Chain to create harmonic progressions automatically. They generate a temporal reference of the music's internal beat structure to guide progressions. Their generated harmonics are coherent with the training data's style, considering the musical mid-term and long-term dependencies. Their Markov structure focuses on chords and each chord's beat position, producing chord progressions that are indistinguishable from ones generated by humans.

An ensemble of Markov Chains was used to learn how to create Mega Man levels in (LI *et al.*, 2021b). They trained multiple Markov Chains to understand the structure of Mega Man's levels and compared to (SNODGRASS; ONTAñóN, 2017)'s approach, which uses Markov chains with a series of restrictions to create levels for the Super Mario Bros. game. In (SNODGRASS; ONTAñóN, 2017), they represent levels as a collection of layers: a structural layer capturing object placement, a player path layer capturing the path of an agent through the level, and a height layer capturing sections of different heights across the level. Their representation even captured some nuances on level generation, such as springboard placement.

(LI *et al.*, 2021b) enhanced the previous model by considering the vertical and horizontal rooms from Mega Man (as Super Mario Bros. has only horizontal levels) and added a direction node to signal the next room's direction. Their approach could only guarantee some levels were feasible, and most were unplayable, but could provide content similar to the original game and generate complex levels.

Considering these successful uses of Markov Chains in PCG, especially in adding complexity and variety to the structures, we chose to use a stochastic grammar with the probability guided by a Markov Chain. Stochastic grammars with probabilities estimated from hidden Markov Models and Bayesian models are relatively standard in approaches besides PCG, as shown in (LIN *et al.*, 2009). The authors use probabilistic grammar to help recognize and represent objects on images. However, the grammar's probabilities were hand-made, and a Markov random field was used as another part of the process to describe the pictorial spatial relationship between elements.

A similar approach for classifying activities in vision tasks (and other applications) is proposed by (VO; BOBICK, 2014) to parse temporal sequences, defined as a composition of sub-activities or sub-actions. A stochastic context-free grammar represents the temporal structure of the high-level activity. Given the grammar, the method determines a Bayes network where the variable nodes are the start and end times of the actions. However, their grammar rules are given equal probabilities or defined by a training dataset.

These grammars have been explored for music generations in the work of (PERCHY; SARRIA, 2009), in which musical sequences are generated to help musicians compose. The grammar generates harmonic progressions (similarly to (ROIG *et al.*, 2018)) using user-defined probabilities for the different rules. Moreover, they have been used to parse music to find some structures of interest. (GILBERT; CONKLIN, 2007) uses the grammar to find tree structures and evaluate the entropy, estimating information compression figures, while (GIRAUD; STAWORKO, 2015) tries to find high-level structures in scores.

As for their applications in games, we have found only two works that use stochastic grammar. The first is (CHEN; GUY, 2018), where the work introduces a domain-specific language for PCG, the Grammatical Item Generation Language. The language supports a compact representation of PCG with stochastic grammar, where the created objects maintain grammatical structures. Their grammar encodes the relationship between aspects of an object, e.g., considering the possible monsters and weapons they may hold, given by the user when creating the grammar, and selecting one based on user-given probabilities. However, they also allow for rules which can stop the grammar when a certain number of exchanges have taken place. This approach bears some similarities to our stochastic grammar (Section 4.3.6), except that we increase variability by using a Markov Chain to update the probabilities at each exchange, following a decaying function, which reduces the chance of remaining in a given state, without forcing a stop.

The other research is (MARTINOVIC; GOOL, 2013), where two-dimensional attributed stochastic and context-free grammar learned from a set of labeled building facades, using a Bayesian Model Merging technique. The authors use it to create a new building, following the learned style. Their approach could successfully generate novel facades, outperforming other methods which needed the grammar rules to be manually designed.

As we did not find many grammar-based approaches in quest generation, we also searched for usages of grammar in a field close to quest generation: story (or narrative) generation. (Kybartas; Bidarra, 2017) define a narrative as having a story and a discourse. The former being the narrative's main content: what happens in it, its plot, and the space where it occurs. The latter is the particular telling of the story, including the space's style, ordering, and duration of the events. A plot is a set of events with a temporal order and causal relation between the elements; They usually consist of one or more low-level actions related to several entities in the space. The space includes characters, settings, props, and anything present within the narrative. It evolves during the game. In our approach, we use a grammar-based plot generator, where usually the plot grammar contains authored rules that determine which narrative contents to link to each event. In Kybartas' and Bidarra's review of story generation techniques, they present different researches that used such techniques. Still, the one closest to ours is the ReGEN project, which we will explain next.

In ReGEN (Kybartas; Verbrugge, 2014), the authors use a context-aware graph rewriting

framework to create a narrative generation system. A graph represents the game world, and the created narratives reflect the current world state and can modify it. They create an initial graph and rewrite rules similar to grammar rules. They search for a graph state and, if found, change it to another configuration. The authors define what could be elements in the game world that create a potential story, and also rules that search for the conditions. Then, they change them into a narrative. For example, finding NPCs that love or hate each other and creating a narrative that someone was murdered for one of these reasons. They set secondary rules that generate branches, when possible, to make the description more interesting. The authors also propose interesting narrative metrics like the size of paths, the number of branches, and costs as irreversible actions. In our work, the proposed grammar contains a set of possible narratives to happen. However, we use our world state to find likely NPCs and items that can become the target of the narrative. Thus, the method creates on-demand quests, not necessarily altering the game world because of NPC or items.

(ALHUSSAIN; AZMI, 2021) wrote a more recent survey on grammar-based algorithms. Most of them are specialized grammars, limited to the domain that produced them, generating a small restricted set of stories. For a more general grammar to generate stories, the authors refer only to psychological models created in the 80s, but no algorithmic implementations for procedural generation.

In conclusion, our review of the literature on this subject points out that quest generation is an area that needs to be explored. As previously mentioned, our first approach was applying simple grammar for a proof-of-concept, shown in Section 4.3.5. We then present our novel approach using stochastic grammars with Markov Chain in Section 4.3.6, based on satisfactory results in the literature applying both Markov Chains and stochastic grammars in other areas, especially in the PCG domain as reported in this section.

## 3.5 Generation of multiple contents

One of our research's main objectives is to orchestrate multiple game content generated by procedural systems. The orchestration of multiple contents is a relatively recent area of PCG, with one of its significant contributions reported in (Liapis *et al.*, 2019). The authors identified the main creative facets of games, showing how different approaches can facilitate orchestration. They also raise questions and challenges for the field, some of which we try to tackle.

In (Liapis *et al.*, 2019), the content of games, especially when focusing on what to generate procedurally, was divided into six different creative facets:

- **V**isuals - How the game is rendered. The visual representation of the game.
- **A**udio - Background music and sounds. It can set the feel and mood of the game.

- **N**arrative - Not required to be elaborate on all games; they are the main motivation to play and complete the game. It may contain elaborate dialogues, stories, or just events and missions.
- **L**evels - The virtual space where the game takes place. It can range from simple spaces to complex labyrinths, and a game may have numerous short levels or a single massive level.
- **R**ules - Players are bound to rules that determine the transition between game states after a player uses a mechanic. These mechanics allow them to interact with the world.
- **G**ameplay - The experience of a game, the process of an agent interacting with a game. Usually, it is created by simulating a player agent in the field of PCG, mainly if it can mimic different players.

We will use these facets to compare the current literature on the generation of multiple contents and how our research will try to advance the state-of-the-art in Table 2, where each facet is represented by its initial letter. Another critical definition from (Liapis *et al.*, 2019) is the orchestration processes. The authors define the orchestration as "the collaboration of multiple computational designers, each focusing on the creation of content primarily for one facet". This orchestration can be done either by the top-down spectrum, a composition that provides as much detail as possible to individual generators, or the bottom-up one, where each generator would create its content and evaluate it with the output of the other generators. A summary of the three approaches follows:

**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;

**Bottom-up** Each generator creates its content, compares to the others, and uses the feedback to improve towards a common goal;

**In-between** Uses different proportions from both.

We describe here four approaches. The *Creative Maestro* is an **In-between**, where the rules from the maestro can be re-interpreted by the generators if they see room for better content, but must be propagated to all others to maintain the feasibility of the contents. The *Jamming with Fake Sheets* is another **In-between**, the one we chose for our project. It consists of specifying only the essential elements of the game structure and letting each generator build on and expand it, following the structure but having the freedom to choose the specifics of the content. In the *Vertical Slices*, each content is generated and upgraded in new iterations. If a new iteration of a certain content underperforms compared to the previous one, it is discarded, and the previous version is used instead. The *Post-production* evaluates the generated content, and some repairs

can occur, smoothing out errors or dissonance and incoherence among components. This one can be done together with other approaches (Liapis *et al.*, 2019).

In (Liapis *et al.*, 2019), there are case studies of orchestration, that we will briefly present in Figure 13, and then discuss the closest related ones in more depth.

Figure 13 – PCG works which generate multiple creative facets. As in (Liapis *et al.*, 2019).

***Angelina*** visuals and audio were created based on an article (that itself acted, in a sense, as the game's narrative), and an independent level generator created platforming stages (COOK; COLTON; PEASE, 2012).

***Game-O-Matic*** A human-made small-scale narrative informs the world's game objects and rules. The visuals from the objects are taken from online sources (TREANOR *et al.*, 2012).

***A Rogue Dream*** A name input by the user is the seed to select the names and visuals of enemies, goals, and items, as well as the special ability name and mechanic. An independent level generator also creates content (COOK; COLTON, 2014).

***Data Adventures*** Generates a plot by linking together Wikipedia articles and using their titles to search for visuals. The levels are created using the real location of cities around the globe, and the layout of each city is based on OpenStreetMap (GREEN *et al.*, 2018).

***Game Forge*** A narrative is generated by a sequence of hero and NPC actions, and it is used to guide the level generation (Hartsook *et al.*, 2011).

***AudioInSpace*** Game audio is used to control the trajectory, speed, and color of the player's bullets. The player can select their favorite weapons and control an interactive evolution of them. On the other hand, the bullets and firing actions from the player affect the audio. Using this loop, the game can adapt itself to the player to some extent (HOOVER *et al.*, 2015).

***Sonancia*** Levels are created based on the tension progression, which can be authored or generated. The level's progression influences the sounds in each room (LOPES; LIAPIS; YANNAKAKIS, 2016).

***Mechanic Miner*** Game programming is adjusted to allow new gameplay, and levels are evolved to use the new mechanics, which are evaluated by the gameplay of random agents (COOK *et al.*, 2013).

***Ludi*** the rules of different pieces and the board layout are evolved. The quality of the content is evaluated by simulating playthroughs with artificial agents (BROWNE; MAIRE, 2010).

*Angelina* (COOK; COLTON; PEASE, 2012), *Game-O-Matic* (TREANOR *et al.*, 2012) and *Rogue Dream* (COOK; COLTON, 2014) are the ones handling a total of four creative facets. Although interesting works, we raise questions over their real-world applicability for games, especially as they depend on content scraped from the web without a means of validation from algorithms or human designers. They are also partially limited in their flexibility to adapt to the different needs of players and designers. In general, they do not present metrics or objective

functions for evaluation, nor an ML algorithm able to learn how to validate or grade contents.

In Cook et al.'s *Angelina*(COOK; COLTON; PEASE, 2012), the system evaluates online sources' moods to select background images, sounds, and contents, placing them in a *metroidvania* game and evolving its levels. However, neither mood nor player preferences affect the level generator algorithm. Moreover, only the creativity is evaluated (which gave positive results), and not other aspects of gameplay and fun.

*Game-O-Matic*(TREANOR *et al.*, 2012) shares a similar context, creating relationships between entities through text and using its content to search the internet for visuals. Verbs become game mechanics via predetermined rules, and a post-processing step eliminates unfeasible rules. A simple level generation is done using rules as a foundation. As it is a co-creative system, it does not adapt its content to players, and there is no evaluation with users.

*Rogue Dream*(COOK; COLTON, 2014) uses the player's name as a random seed to generate an association between objects and the visuals. This association is done via Google autocompletes, and the results are crossed with a prescripted list of rules for mechanics, enemies, and game rules. There is also a level generator, following a more computer-driven approach. The system also lacks user adaptation and evaluation with players.

*AudioInSpace*(HOOVER *et al.*, 2015) generates audio via a Compositional Pattern Producing Network, and another network defines the position and color of players' bullets. Both use each other's output as feedback. The player selects a favorite weapon after a match, and the method uses the information as input for the networks. Thus, the work generates rules (bullet patterns) and visuals (bullet color) at a high level. The system allows adaptation to different players to some degree, as it is limited to the player's bullets and audio contents.

The last example we discuss from Liapis et al. review is *Game Forge* (HARTSOOK *et al.*, 2011). They orchestrate narrative and levels to reach positive gameplay quality metrics, matching the user's desired settings. However, the evaluation did not include feedback from players. Our system has features inspired by *Game Forge*, as we also use a quest-generating algorithm as the base for the other generators.

The authors in (Prager *et al.*, 2019) present another relevant aspect of game facet combination by creating a simple maze game. The visual and audio styles can change between a dark and tense setting and a soothing and joyful one. They tested how 20 players reacted to compositions of facets considered as usual (both audio and visuals matching for either setting) and unusual (the audio was from one setting and the visuals from another). A machine learning model predicts users' reactions, and the results show that, as expected, users perceived the homogeneous settings as more fun and less difficult than the heterogeneous ones. They also found the homogeneous settings to create more self-reported arousal instances during gameplay, and the instances were more stable. Although considering only two facets and a small sample of users, this research shows how a good orchestration of multiple contents is important to create

funnier games.

The authors in (HEIJNE, 2016) (already cited as an inspiration for data collection in Section 3.1) show a procedural clone from *The Legend of Zelda: A Link to the Past* that can generate levels, puzzles and what enemies populate each room procedurally, although with low variability. The puzzles and enemies can be considered part of the rules facet. The authors also adapt the content for different, manually tuned difficulty settings and collect many data from the players, trying to profile them.

Finally, the work in (KARAVOLOS; LIAPIS; YANNAKAKIS, 2021) presents an important experiment involving the creation of multiple game facets and preliminary steps toward player profiling in an environment with multiple procedural contents. They procedurally can generate both levels and rules, in the form of character class parameters) for a shooter game. Artificial agents play the game, simulate real players, and collect data from the outcomes. The level configuration, the character class parameters, and the outcomes train a Convolutional Neural Network (CNN). The network has to predict the kill ratio between the two players, the time of the match, and two entropy scores calculated from heatmaps: the first for all death locations and the second for the players' positions.

The authors tested different inputs for the CNN and found out that the parameters related to the rule facet were the most important ones for correct predictions. Still, the data from the level facet improved accuracy. This is an intriguing result, as two game facets can be used to predict the outcome of a third one (gameplay). This could help systems that create multiple contents procedurally to ensure feasible and balanced content. Furthermore, they show that data extracted from multiple facets can help create a more accurate model for player profiling, and even machine learning models that can be used to adapt the procedural content for different players.

In our approach, we generate three different facets: levels (the dungeons and their rooms), rules (as the mechanics of enemies), and narrative (both as the quests and the locked-door puzzles). Other works have already created up to four facets. However, we create more than a single content for the same facet using different algorithms. For example, the locked-doors, created with the dungeon using an EA, and the stochastic grammar generating the quests are both parts of the narrative facet, adding some complexity to our system.

For most contents, we use algorithms that adapt to different users' needs, allowing for satisfactory adaptability, and tested against players (shown in Chapter 5 with positive results), especially on how they please players with different profiles. We also include novelty in creating a more robust architecture for orchestrating multiple contents, which allows the addition and exchange of algorithms with relative ease. In none of the works we found such quality of adaptation to users and the system's adaptation to different designers' needs.

## 3.6 System Architecture

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[1], which guides the fields of software design and architecture, refactoring, test-driven development, DevOps and alike (FOWLER, 2018; MARTIN *et al.*, 2018; BECK, 2003; KIM *et al.*, 2021).

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(MIZUTANI; DAROS; KON, 2021), which, from 512 studies, found only 36 representative ones with sufficient architectural description and, between them, most tailored to specific games or genres. The authors report studies presenting that software architectures benefit from reuse, providing better systems, even if employed for a particular game application.

Although Mizutani et al.'s review (MIZUTANI; DAROS; KON, 2021) 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 (KREMINSKI *et al.*, 2020), where they created an architecture for a co-creative storytelling game. Albeit in a container-level depth[2], they present the central communications between entities as well as with other containers.

We have yet to find works approaching implementing satisfactory architectures for PCG systems, but there are interesting works on design patterns for PCG. Design Patterns are a more low-level view of the system than architectures, as they are common usages of the creation of specific classes and making them communicate between themselves in known problems, especially in the object-oriented paradigm. We advise the reader to not mistake software design patterns for game design patterns, as the ones presented in the work of (BEAUPRE *et al.*, 2018). They extracted a "design" design pattern after analyzing the corpus levels from the general video game AI framework and proposed a method for generating levels based on said patterns. But there is no relation to software design patterns in such work.

The work of (TREANOR *et al.*, 2015a) provides design patterns for AI applications in games, also proposing a generative ideation technique combining a design pattern with an AI technique or capacity. These patterns also tend to be game design patterns, like software design

[1] <https://martinfowler.com/bliki/DesignStaminaHypothesis.html>
[2] Using C4's model classification (<https://c4model.com/>)

patterns. However, as they indicate software strategies to be used with each pattern, we consider it relevant for our case. The authors then developed two short games as examples of how these patterns may help game development, as well as how to combine them for more exciting games. Figure 14 defines the patterns they stated.

Figure 14 – AI-Based Game Design Patterns. As in (TREANOR *et al.*, 2015a).

***AI is Visualized*** Provide a visual representation of the AI state and make the gameplay revolve around manipulating this state.

***AI as Role-model*** Provide AI agents for the player to mimic their actions, e.g., mimic the states in a Finite State Machine.

***AI as Trainee*** Make players train an AI to perform tasks (usually training a Machine Learning algorithm).

***AI is Editable*** Allow the player to edit elements of an AI agent, such as parameters or data structures.

***AI is Guided*** Make players aid an AI that does not fare well in a complex environment, changing the environment to be suitable for it.

***AI as Co-creator*** Involve the player in a creative task paired with an AI, e.g., the mixed-initiative procedural generators.

***AI as Adversary*** Player must defeat an AI opponent. This is how AI is used in most games, especially single player ones.

***AI as Villain*** The player must complete a task or defeat the AI; however, the AI is not aiming to defeat the player, but to create an experience or atmosphere (e.g., tension).

***AI as Spectacle*** The player observes or interferes with a complex system created by AI agents. Arguably, simulation games with limited interference apply this pattern.

(COOK *et al.*, 2016) describes PCG-based games, inspired by the work mentioned above, which are games guided by procedural content generators. They propose some design patterns based in reoccurrences in the literature. Said patterns focus on game design, but still provide insights into developing the software. The authors then study two cases as examples of how these patterns may help game development. Figure 15 defines the patterns they stated.

The authors also discuss some essential themes when creating games centered on PCG. For instance, the visualization of the process and the output must be clear to the player, so they know how it will respond to their interactions. Another one is how and when to limit player interaction with the system, aiming not to make them feel powerless. And the last is that these games may be a way for non-programmers to be interested in PCG and participate actively in the

Figure 15 – PCG-Based Game Design Patterns. As in (COOK *et al.*, 2016).

***AI as Creative Proxy*** Player designs or tweaks a generative system and the content is created. The player does not create the content *together* with the generator, they design the generator itself.

***AI as Meta-Environment*** The possibility space of the generator can be traversed and transformed by the player. The player may adjust parameters or input data of the generator to, for example, guide the generator to create a particular type of content.

***AI is Filtered*** Player acts as the fitness function or filter for the content, selecting, ranking or filtering said content. Similar to most mixed-initiative generators, but making the selections is part of a game itself.

***AI is Editable*** Allow the player to edit elements of an AI agent, such as parameters or data structures.

***AI is Interrupted*** Player interjects in the generation by stopping, slowing or restarting the process. The player may want to stop the process to use a content generated before the stop criteria is reached, or to reconfigure the setup if it is going in a way the player does not agree with.

design of these systems without programming knowledge.

As our final finding when searching for works about system design and architecture on PCG, we found the work in (COOK, 2020), where the author explores the impact of software engineering decisions on how an automated game design system understands the codebase, generates new code and evaluates the results. They define Automated Game Design as the umbrella term for the study and engineering of AI systems that actively participate in game design, be it creating, critiquing, and editing their core systems or creating content. They argue that these systems, to date, are unlikely to be adopted by the industry, as professionals need more comfort of use and credibility on the systems. These systems generally use game description languages, custom domain-specific languages that describe games, which are interpreted by a custom engine.

The research presented in (COOK, 2020) discusses how to create an Automated Game Design system working directly with code, and how to design a system to be integrated with a large, existing game codebase, especially what software engineering decisions may impact such a system. Most of their analysis is about accessibility, encapsulation, method type signatures, post conditions, and side effects. Such systems must inspect the game code to work and may cause bugs when altering the code. In our approach, as with many other PCG systems applied in games, these aren't problems, as we develop our systems in the same language and the engine where the game is implemented. The authors in (COOK, 2020) also discuss some software design patterns

common in game development, the Model View Controller and the Entity Component System (used in our first enemy generator approach, Section 4.3.3). In the discussion, the Model View Controller is pointed as a model which demands more complexity from the Automated Game Design system, as the code is more decentralized. They present some points to help system development for commercial games. They mention the need for formal specifications, limiting the space where the system works in, the automated discovery of code specifications, developing new programming language features such as refinement types, and creating games using the Entity Component System.

Thus, we conclude with our review that, as far as we know, there is a need for better designs and architectures related to PCG systems, especially considering their application in the game industry, from small to large teams of developers. However, more work needs to be done to find better solutions. Some papers show that game development teams using design patterns in their projects have better results than those that do not. And some works focusing on game design patterns also propose some interesting patterns for software developers to follow. In this thesis, we give a first step towards documenting an architecture able to create multiple contents for a game developed in Unity, one of the most used game engines in the industry. The proposed architecture worked both to develop a game prototype and to create a preliminary version of a framework to handle different algorithms for the generation of multiple contents.

## 3.7 Summary

Table 2 compares our findings, based on creative facets generated as stated in (Liapis *et al.*, 2019): **V**isuals, **A**udio, **N**arrative, **L**evel, **R**ules and **G**ameplay. We also evaluate if the works approach adaptive contents and any player profiling (especially machine learning algorithms that differentiate players according to gameplay characteristics). The values **Y**es or **N**o in each cell indicate the satisfaction or not of those features. The value **P**artially occurs when the execution is not fully procedural or without a high degree of adaptation. We presented why some facets from the works reviewed in (Liapis *et al.*, 2019) are partially generated in Section 3.5. For (BICHO; MARTINHO, 2018), we consider the rules partially generated because, when the level changes to adapt to the player's preference on what action they prefer, they favor a rule over another dynamically. For the work in (HEIJNE, 2016), we consider the content partially adaptive because it can only change between three predetermined difficulties, and the changes in the content have a low expressive range. In our research, we plan to create each content with an expressive range as wide as possible and able to adapt itself to many difficulties and player types.

We can observe many relevant results for the procedural generation of different contents, applying various algorithms and approaches. However, a close look at those works shows that creating a system of systems handling them all together is still a very recent area of research. It is still challenging to provide a cohesive generator for multiple contents; as far as we know, there

Table 2 – Comparison between our work and the closely related ones in reviewed literature. The creative facets have been abbreviated to their first letters to fit the page, in the following order: **V**isuals, **A**udio, **N**arrative, **L**evel, **R**ules and **G**ameplay.

| Work | V | A | N | L | R | G | Adaptive? | Player Profiling? |
|---|---|---|---|---|---|---|---|---|
| Angelina (COOK; COLTON; PEASE, 2012) | ✓ | ✓ | *P* | ✓ | × | × | × | × |
| Game-O-Matic (TREANOR *et al.*, 2012) | ✓ | × | *P* | *P* | ✓ | × | × | × |
| A Rogue Dream (COOK; COLTON, 2014) | ✓ | × | ✓ | ✓ | *P* | × | × | × |
| Data Adventures (GREEN *et al.*, 2018) | ✓ | × | ✓ | *P* | × | × | × | × |
| Game Forge (Hartsook *et al.*, 2011) | × | × | ✓ | ✓ | × | × | × | × |
| AudioInSpace (HOOVER *et al.*, 2015) | *P* | ✓ | × | × | *P* | × | *P* | × |
| Sonancia: (LOPES; LIAPIS; YANNAKAKIS, 2016) | × | ✓ | *P* | ✓ | × | × | × | × |
| Mechanic Miner: (COOK *et al.*, 2013) | × | × | × | ✓ | ✓ | *P* | × | × |
| Ludi: (BROWNE; MAIRE, 2010) | × | × | × | *P* | ✓ | ✓ | × | × |
| (KARAVOLOS; LIAPIS; YANNAKAKIS, 2021) | × | × | × | ✓ | ✓ | ✓ | × | ✓ |
| (Prager *et al.*, 2019) | ✓ | ✓ | × | × | × | × | × | ✓ |
| (BICHO; MARTINHO, 2018) | × | × | × | ✓ | *P* | × | ✓ | × |
| (HEIJNE, 2016) | × | × | × | ✓ | ✓ | × | *P* | ✓ |
| This and (PEREIRA; VIANA; TOLEDO, 2022) | × | × | ✓ | ✓ | ✓ | ✓ | × | ✓ |

is no system creating as many contents as our proposal. Adapting generated content can engage players more, but how to collect and interpret player data and best adapt the content using these data is still an open question. Some researches had positive results, usually, those that relied on a broad amount of data, while others could have yielded more consistent results.

Therefore, our hypothesis that we can collect data from players by using short questionnaires and in-game data may help solve this riddle if we can successfully adapt our system of PCG algorithms to different players. We also found that the generation of enemies with the characteristics and behaviors proposed by us is not present in the literature. Even commercial games with a closer approach either do not consider input from the player's preferences, or not consider the environment the enemies would be used. They also do not account for the gameplay status of said enemies (e.g., their damage or attack pattern), only the appearance of the creatures. Moreover, they used algorithms designed specifically for their needs.

In conclusion, our review showed that our research has the potential to advance by creating an adaptive system of PCG algorithms that can both adapt to players and create feasible and cohesive content from many PCG approaches.

CHAPTER 4

# METHODOLOGY

We describe our methodology to develop the proposed multiple-content generator system in this chapter. Our system follows the C4 model introduced in Section 2.10, once it helps to understand systems with relatively large complexity[1], as the one we propose.

Thus, Section 4.1 introduces our system context, describing the interaction among its different system components. Next, we delve into each system's diagram and show their containers in Section 4.2, by describing our three main systems: the *Game*, the *Profile Analyst*, and the *Procedural Content Generator*. Finally, we present in more detail each of the systems' containers in sections 4.3 through 4.5, describing their algorithms and relevant implementation details.

## 4.1 System Context

Our goal in this research is the creation of an architecture able to generate different creative facets procedurally based on user data. The data is analyzed and fed back to the *PCG* system. The Unity real-time development platform, presented in Section 2.3, was used to create a game prototype to help us evaluate the algorithms. The game prototype is inspired by the well-known 2D dungeon-based action-adventure games, like *The Legend of Zelda* series and *The Binding of Isaac* (NINTENDO, 2020; MCMILLEN; HIMSL, 2011a), mentioned in Section 2.1. The prototype will work as an interface with the different PCG algorithms, reading their created content and decoding it into gameplay elements.

We structured our architecture, which we named *Overlord*[2], as a collection of three different systems: *Game*; *Profile Analyst*; and *Procedural Content Generation*. An external server supports these systems, where we upload a WebGL game version and conduct our experiments.

[1] <https://c4model.com/>
[2] A little wordplay as it rules over the maestro (master, or lord) and others.

For our first experimental setups, we also added a *Serialized Content* container to save generated data for *offline* loading, characterized as an *offline* PCG scenario (TOGELIUS; SHAKER; NELSON, 2016). Figure 16 shows the architecture's context diagram. The system is publicly available at GitHub[3].

Figure 16 shows the player interacting with the system by playing the *Game* (considered a separate system) and providing data for the Profile Analyst system. The latter is responsible for processing data from the player, be it explicit data, such as questionnaires, or implicit, like data collected during their playthroughs. The resulting data goes to the Procedural Content Generator system, being processed and used as input by the generators. The content created goes to the Serialized Content container to be utilized later.

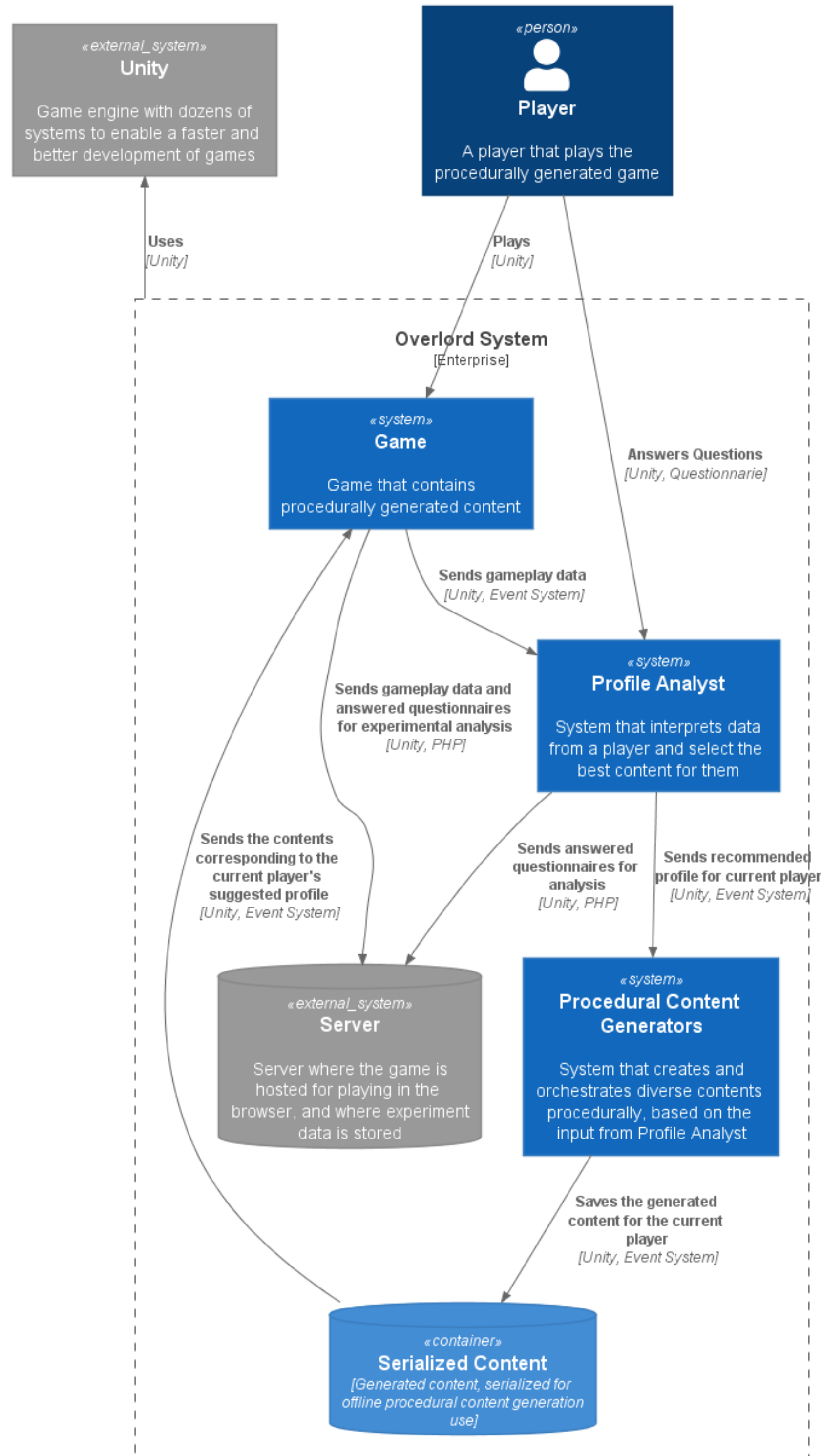
The *PCG* system generates different game facets procedurally and orchestrates their content, guaranteeing feasibility and enhancing their quality to work together. It saves the result in the *Serialized Content* container, allowing it to be used *offline* by the *Game* system. This system, developed using Unity[4], uses the content generated by the *PCG* system and stored in the *Serialized Content* container, providing fun to the player. It also collects gameplay data and sends it to the *Server* and *Profile Analyst*. Both gameplay data from the game, data explicitly given by the player (questionnaires), and essential results provided by the *Profile Analyst* system are stored in a server, allowing them to be analyzed later.

In our architectural 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 (MARTIN; NEWKIRK; KOSS, 2003). Liapis et al. (Liapis *et al.*, 2019) offer 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 *Profile Analyst*'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: storing data for each content in a serialized object. In *Unity*, the *scriptable object*[5] is a helpful data container 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

[3] <https://github.com/LeonardoTPereira/Overlord-Project>
[4] <https://unity.com/>
[5] <https://docs.unity3d.com/Manual/class-ScriptableObject.html>

Figure 16 – System context diagram for the *Overlord* architecture, considering the experiments with offline content generation.

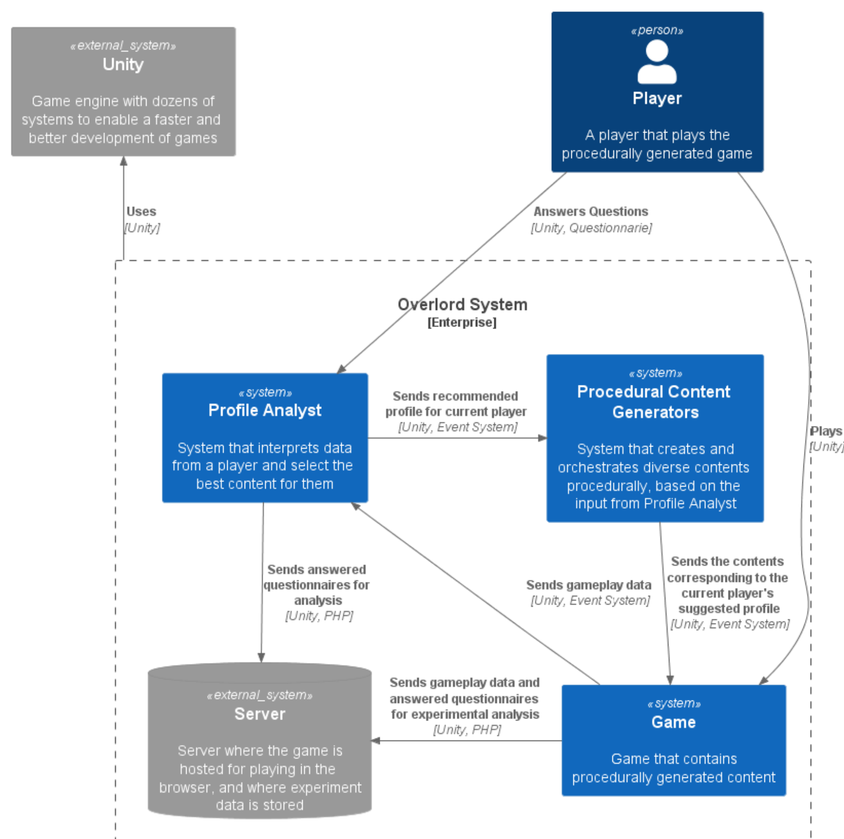



game reads and processes the content into *game objects*, with visuals, sounds, physics, and other game-related aspects ready to be used in-game. By creating this interface, we can use different algorithms to make the same content without changing the *Game* system. Moreover, the *Game* system can change aspects of a *game object* without changes in the generator. E.g., a 2D dungeon can become 3D, but the generator only provides information about its layout and what rooms

have which objects.

After successful tests with our architecture, we adapted it to support online content generation by discarding the *Serialized Content* container. First, we change some parameters in our PCG algorithms to provide a satisfactory result within a limited time. Moreover, after receiving the necessary data for the player's preferences, our system can generate content on demand. Figure 17 shows this final version of our system. The current system's configuration allows for a virtually infinite loop of content generation, as long as input for the new content is fed after each playthrough.

Figure 17 – System context diagram for the *Overlord* architecture, considering the experiments with online content generation.




## 4.2 Systems' Containers

In this section, we present our solution's architecture from a more detailed perspective than the Context one: the equivalent of the Container view from C4 model, showing the high-level technical building blocks. We describe the main containers for each of the three systems: the *Profile Analyst* (Section 4.2.1), the *Procedural Content Generator* (Section 4.2.2), and the *Game* (4.2.3).

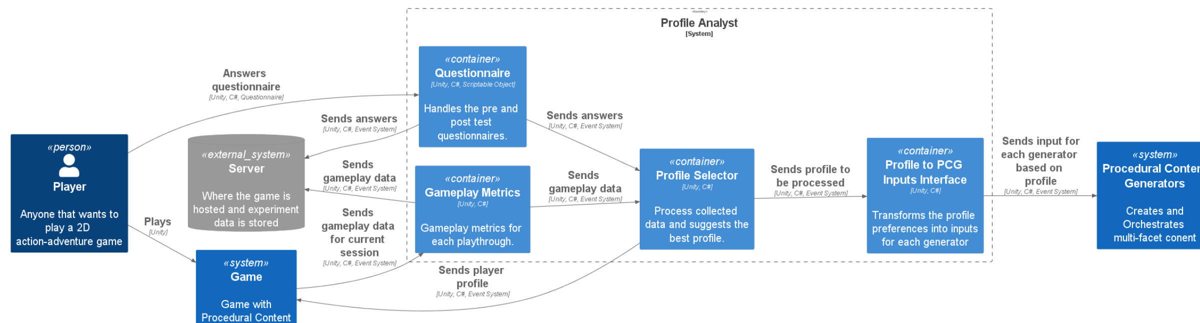
One may argue that our systems may not be complex enough to be categorized as individual systems in the C4 notation, leaning more toward their definition of Container: "something

that needs to be running in order for the overall software system to work."[6]. However, the *Game* application may run without the *Profile Analyst* or the *PCG* contents running, and the same goes for the other ones. Hence, we categorized them, in our abstraction, as different systems.

### 4.2.1 The Profile Analyst System

Fig. 18 shows the *Container Diagram* of the *Profile Analyst* 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 on the Server and send them to the *Profile Analyst*, 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 the experiments in Chapter 5. 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.

Figure 18 – *Profile Analyst* container diagram. The player answers a pre-test questionnaire, the *Profile Selector* analyses the data, suggests a profile and translates into inputs for the *PCG* system via an interface. The *Profile Analyst* also collects metrics from the *Game* system and feed it to the *Profile Selector* container.




### 4.2.2 The 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 four different generators in their containers in Fig. 19: *Quest*, *Dungeon*, *Enemy*, and *Room 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.

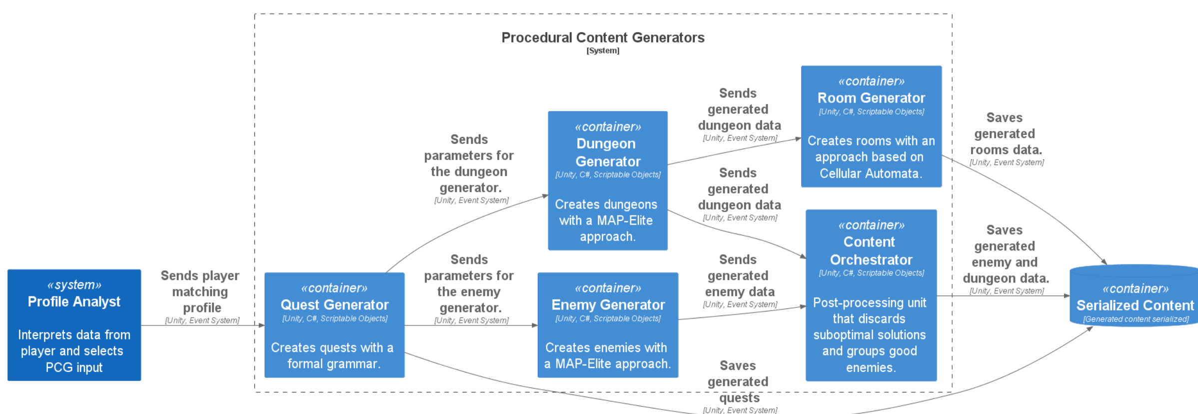
The *Dungeon Generator* container creates dungeons with locked-door puzzles and defines how many enemies occupy each room, providing all items, enemies, and exploration

[6] <https://c4model.com/#Abstractions>

needed from the originating questline. This generation runs parallel to the one in the *Enemy Generator* container in a *Fake-Sheets* step. This one creates enemies with different attributes (e.g., health, damage, speed), weapons, and movement patterns. It guarantees all enemy types needed for the originating questline. Our post-processing 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. The dungeon data is also provided to the *Room Generator*, which creates the layout of the rooms: tile positions, the spawning points of items, NPCs, and enemies. At last, the *Serialized Content* container serializes the resulting dungeon and enemies, and the generation step is over.

Considering Liapis et al.'s definition of content orchestration (detailed in Section 3.5, 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.*, 2019). However, we have some facets with their contents being created in parallel with a *Fake-Sheet*, a frame which generators agree to work into and expand, also from (Liapis *et al.*, 2019). Thus, we can fit the content as requested while slightly diverging. We also apply some post-processing methods in the resulting content to improve its final quality.

Figure 19 – *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 post-processing unit. This unit discards undesired contents and selects each room's best enemy groups, according to the amount needed and available types. A *Room Generator* container creates the rooms as demanded by the dungeon.

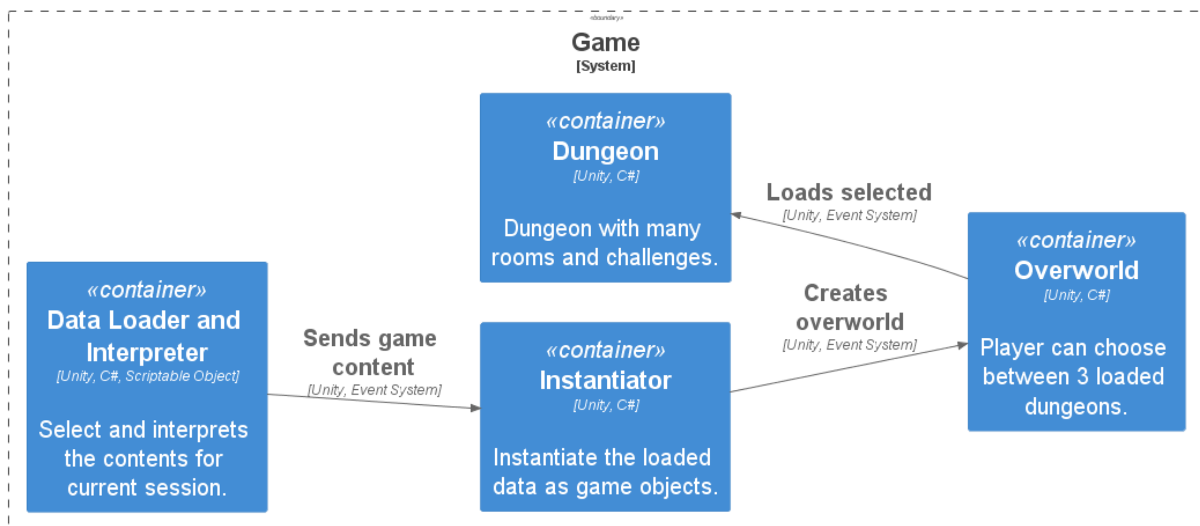



### 4.2.3 The Game System

The *Game* system processes the *Serialized Content* container's data for the current profile, selects what 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. 20 shows the *Game* system container diagram. For the online experimental setup (seen in Section 5.3), we skipped the *Overworld*, and the player starts in the *Dungeon* after the data is loaded.

Figure 20 – *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.




We use a decoupled approach by developing modular components without strong dependencies among them and from the game prototype. PCG algorithms and a *maestro* algorithm control the PCG systems and guarantee the cohesion between contents and their feasibility. The game prototype will receive the generated content and shall be as close as possible to a commercial indie game product to collect more realistic feedback. Bots will initially test the generated content, and other quality-related metrics available in the literature for each type of content (HEIJNE, 2016; Kybartas; Verbrugge, 2014). Questionnaires and implicit data collected from players will be analyzed and used to feed machine learning algorithms to define players' profiles, aiming for personalized content in real-time. Thus, tests with human players will be mandatory during the system development and crucial to validation.

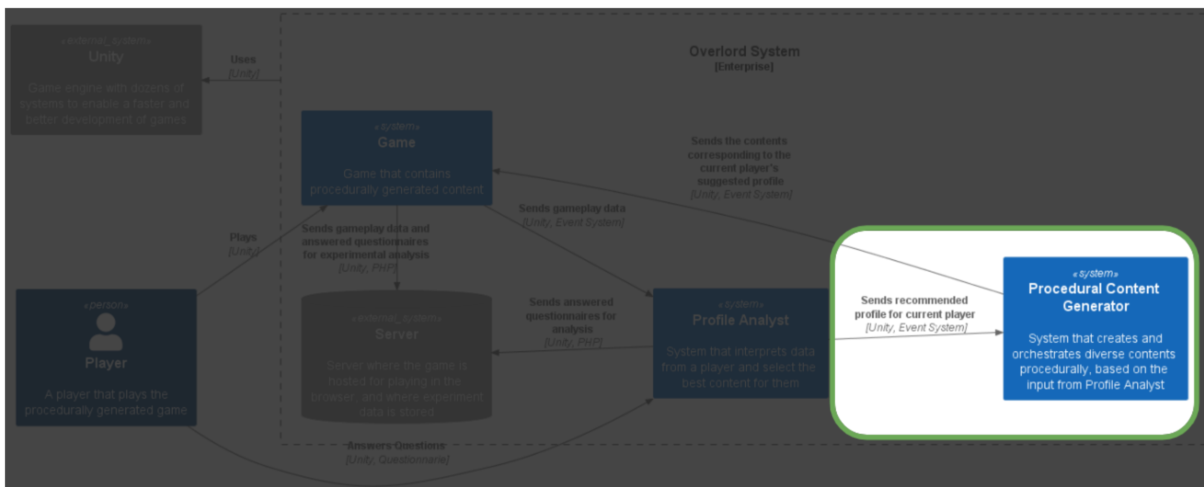
The *Overlord* system works as follows: a new player will answer a pre-test questionnaire so that we can collect their profile. The data will feed the *Profile Analyst* system, which will then decide the best difficulties and configurations for each content based on the player's preferences and provide the *PCG* system. The *Quest Generator* creates the quests and uses the output as input for the other PCG components to create content that matches the player's preferences. Some systems have dependencies, so they must wait for another system to finish their generation

first to receive the necessary input. For example, the *Level Generator* must get the quests from the *Quest Generator* to define how many enemies, items, and NPCs must be in the dungeon, as well as the number of rooms and level linearity. The *Room Generator* must also get information about the number of rooms and their connections. After everything is created and validated by the *Orchestrator*, the data goes to the *Game* system, and the player can play the new content. After playing, the new gameplay data goes to the *Profile Analyst* system, and we may have a new loop.

## 4.3 Procedural Generation

In this section, we present the procedural generation systems and the *orchestration* algorithm, which helps to make the generated contents more cohesive when placed together. They are part of the Procedural Content Generator System in the Overlord Context Diagram (Figure 16), highlighted in Figure 21.

Figure 21 – Highlight of the Procedural Content Generator System inside the Overlord Context Diagram




An EA is proposed to generate dungeons in Section 4.3.1, as well as a MAP-Elites variation of the approach in Section 4.3.2. The Enemy generator will also apply an EA, described in Section 4.3.3, and a MAP-Elites variation, shown in Section 4.3.4. The Quest generator uses formal grammars to create its contents, and it is detailed in Section 4.3.5. An extension to this method, applying Markov Chains besides the grammar, is presented in Section 4.3.6. The rooms inside the dungeons are generated using an algorithm based on Cellular Automata, described in Section 4.3.7. Finally, the *orchestration* algorithm is described in Section 4.3.8, as the one that will tie the content generation together as a cohesive system.

To summarize how the Procedural Content Generation System works, we also present an example in Section 4.3.9.

### 4.3.1 Evolutionary Algorithm for Dungeon Generation

The present section describes how dungeons and their respective locked-door missions are created. We use a tree structure to encode the genotype information about the dungeon layout, missions, and enemy positioning. We then decode the tree to a 2D grid to enable the spatial placement of rooms. We also present the evolutionary algorithm that evolves the structure alongside its operators. The algorithm follows the implementation of Pereira et al.(PEREIRA *et al.*, 2021), developed as the final product of the author's master's degree project(PEREIRA, 2018).

Our dungeons and their missions are modeled after a tree representation as shown in Figure 22a and presented in (PEREIRA *et al.*, 2021). Each node is a type of room. A Normal Room (NR) has no keys nor locks. A Key Room (KR) has a key inside and is represented by a bold node. And a Locked Room (LR), which has a locked path between itself and its parent, has this lock shown as a dashed edge. Each key has a unique ID, shared with the lock that it opens. Thus, we are encoding the mission and the play space at the same time.

Each child node assumes a direction in a grid space related to the parent node (room) as shown in Figure 22a: Right (R), Left (L), and Down (D). Moreover, the parent is always in the North (N) of the room. The root node represents the player's spawn point, and it is shown as the node with a thin dashed line around it in Figure 22a. The node behind the farthest locked door (LR) is considered the goal. The nodes also hold information about the number of enemies in their corresponding room.

The root node will always be created first and placed in a queue of rooms during the population's initialization. A breadth-first algorithm processes the queue until its end and creates children nodes added to the queue. Each room's number of children and their respective position is determined randomly. After the tree generation, we place the enemies by drawing, with repetition, a random node (different from the root and the goal, as, generally, these rooms do not contain enemies in games) and adding a single enemy to it until the placement of all necessary enemies. The number of enemies is given as input and fixed throughout all generations and individuals.

Figure 4 gives a detailed description of the tree-generation algorithm. The breadth-first approach applied to distribute unique rooms will guarantee many constraints' validation by itself.

We maintain the translation from genotype to phenotype from (PEREIRA *et al.*, 2021), processing the tree into a 2D grid and placing each room and corridor according to their position from the parent node. The starting room (root node) is placed at (0,0), and the other nodes have their coordinates calculated according to their positions in the tree as shown in Figures 22 and 23. In Figure 22, the root node is not rotated, and its children can be placed in the grid in the same direction as they are in the tree. However, a rotation is done in Figure 23 to keep the parent node as north for its child. Placing a new child may lead to overlapping with existing rooms.

**Algorithm 4** – Tree Structure-generation algorithm (PEREIRA *et al.*, 2021)

**Ensure:** A new dungeon with several rooms
1: **while** $toVisit.Count > 0$ **do**
2: $actualRoom \leftarrow toVisit.Dequeue(ROOM_TYPE)$
3: // Get the depth of the current room in the tree
4: $actualDepth \leftarrow actualRoom.Depth$
5: **if** $actualDepth > MAX_DEPTH$ **then**
6: // Dungeon reached max depth, clear the list of rooms to be visited
7: $toVisit.Clear()$
8: $break$
9: **else**
10: $prob \leftarrow$ Draw number to check if room will have children
11: $PROB_CHILD \leftarrow PROB_HAS_CHILD * (1 - actualDepth \div 10)$
12: **if** $prob \leq PROB_CHILD$ **then**
13: // Draw number of *actualRoom*'s children
14: $numberOfChildren \leftarrow draw(MAX_CHILD)$
15: // If drawn number is 3, the array will have all possible directions
16: $directionArray \leftarrow$ Draw each child's direction
17: **for** $i = 1 \rightarrow numberOfChildren$ **do**
18: **if** $prob < PROB_KEY$ **then**
19: $ROOM_TYPE \leftarrow KEY$
20: **else if** $prob \geq PROB_NORMAL$ **then**
21: $ROOM_TYPE \leftarrow NORMAL$
22: **else**
23: $ROOM_TYPE \leftarrow LOCK$
24: **end if**
25: $child \leftarrow createRoom(ROOM_TYPE)$
26: //Insert new room as child of *currentRoom*, calculates [X,Y] position
27: $currentRoom.InsertChild(directionArray[i], child)$
28: $child.Direction \leftarrow directionArray[i]$
29: $toVisit.Enqueue(child)$
30: **end for**
31: **end if**
32: **end if**
33: **end while**

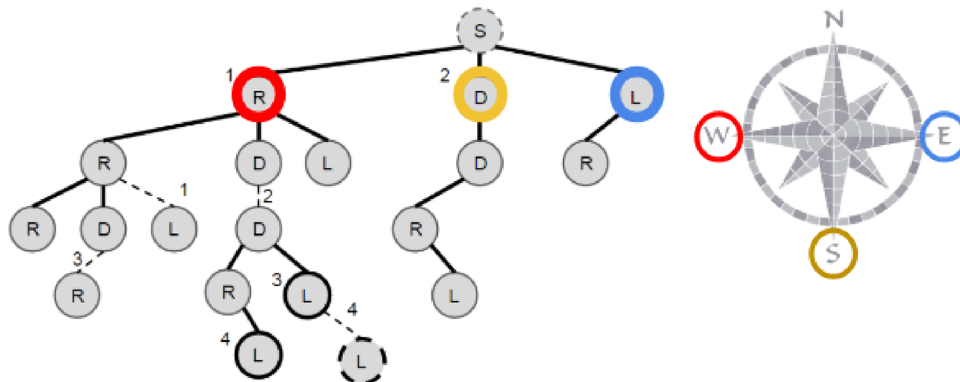
Therefore, they are tested for overlapping when creating the tree and discarded if an overlap happens, guaranteeing the feasibility of every tree.

The overlapping problem and its solution are better illustrated by Figure 24. There is a tree with two overlapping nodes in Figure 24a. When it is translated to the grid, the overlap occurs, and the node that causes the overlap is removed from the grid in Figures 24b through 24e. Thus, the tree structure is fixed and generates a feasible dungeon, as shown by Figure 24f. It is important to note that the enemy distribution is done after we fix any overlaps in the dungeon.

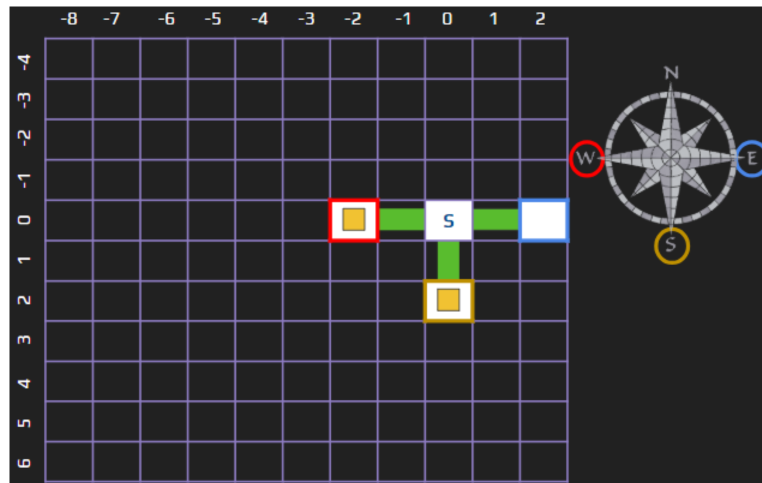
Figure 25 introduces the EA that follows the same basic steps described in (PEREIRA *et al.*, 2021). Crossover and mutation operators create a new population at each generation. The

Figure 22 – Genotype-phenotype translation. Dungeon Parameters: 20 Rooms, 4 Keys, 4 Locks and 1.5 Linear Coefficient

(a) The relative directions of the nodes from the root are taken.




(b) The rotation of their direction is not necessary, as it is the same as the global orientation from the grid. Note that the keys in the rooms are represented by yellow squares.



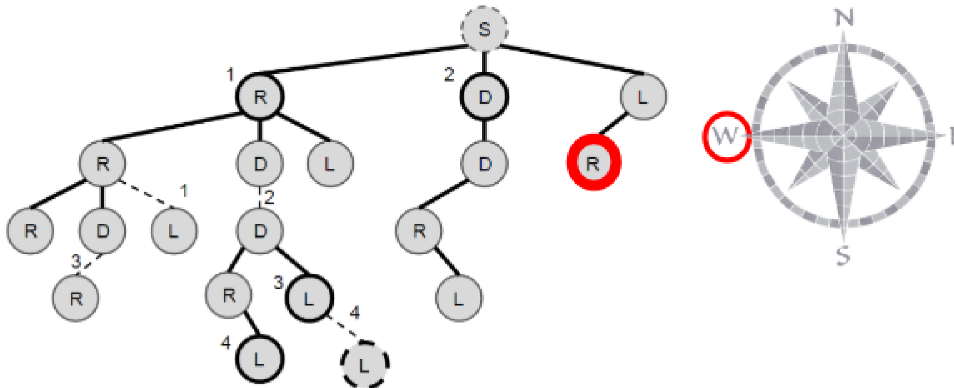

tournament operator with size 2 selects individuals for reproduction, creating two new individuals. The elitism operator will guarantee that the best individual from the previous generation is kept in the next generation.

Figure 26 describes the crossover operator, where the main idea is to better preserve the semantic information about dungeons after the generation of the children's solutions. The crossover algorithm keeps the same number of special rooms of the removed branch when exchanging it between parents. Moreover, it preserves the breadth-first sequence of the special room placement by saving its order before the exchange.

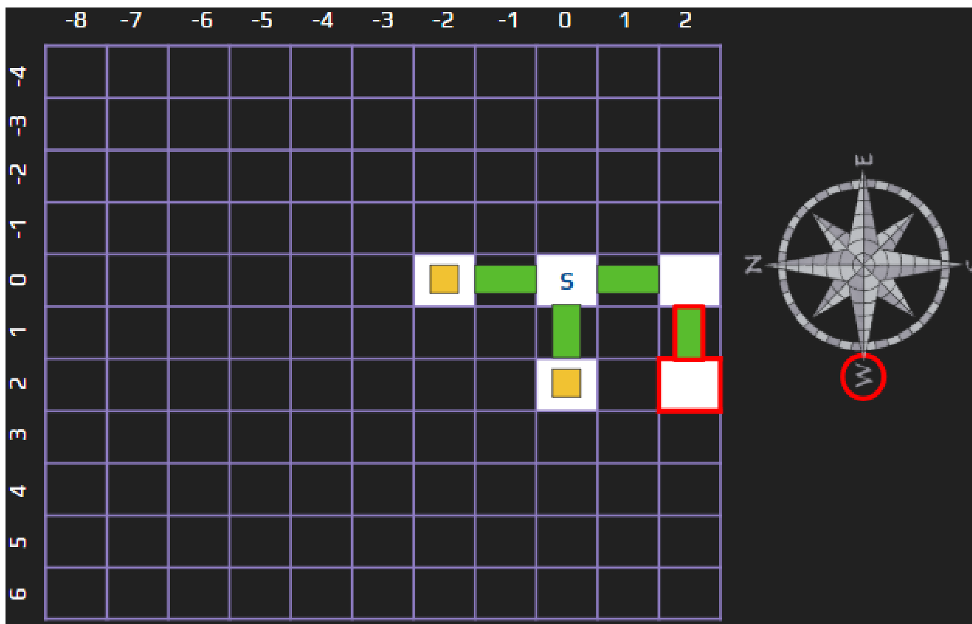
The crossover happens first over copies of the individuals, once overlapping may occur when exchanging branches. After the branch exchanges, we rebuild the grid (Figure 26, lines 12-13) and, in doing so, we may find overlapping nodes as illustrated in Figure 24. Suppose the remaining number of rooms, after removing overlapping nodes, is less than the necessary number of special rooms (Figure 26, lines 14-17). Thus, the method cancels the operation, selecting another pair of cut points without the replacement of said points. These lines also check if there

Figure 23 – Next step of genotype-phenotype translation

(a) The right child of the root node's left child is selected and its direction checked




(b) It needs to be rotated 270° to match the global orientation from the grid, being placed at south from its parent.

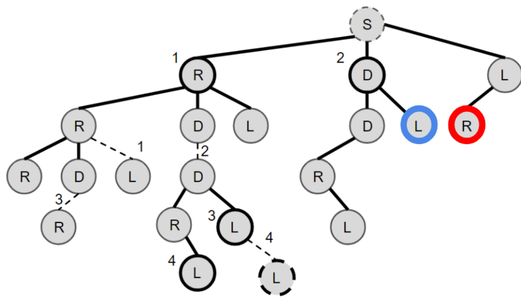



are more than two rooms in each branch to safeguard the crossover of enemies. If the single room to be exchanged is the goal, enemies from the other branch would have nowhere to go, and the child would be infeasible. Otherwise, the crossover is validated, and the copies take the place of the original levels. If there are no feasible cut points, the crossover operation does not happen, as shown in the conditional operation in line 23 in Figure 26.

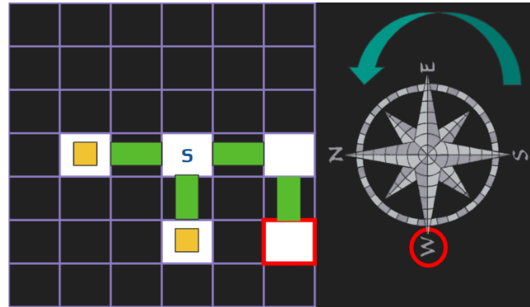
The fix algorithm redistributes the special rooms after the trade (Figure 26, lines 28-29) by, basically, adapting some steps from Figure 4. Instead of generating the whole tree, the fix algorithm is executed only in the individual's newly acquired branch. At the start of the fix procedure, the probabilities of a room being an NR, KR, or LR are the same from the tree-generation algorithm, and the same visit order (breadth-first) is used. If all the special rooms saved from the previous branch have already been placed in the new one, every new room will be an NR. If there are only $n$ rooms not-yet visited by the breadth-first algorithm, during the fix execution, and $n_s$ special rooms left, where $n = n_s$, all the $n$ remaining rooms in the branch

Figure 24 – Example of overlapping nodes and a fixed tree.

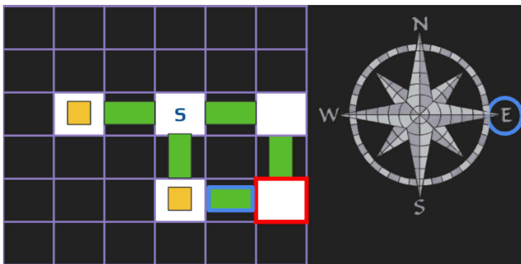
(a) Both blue and red-marked nodes are overlapping.



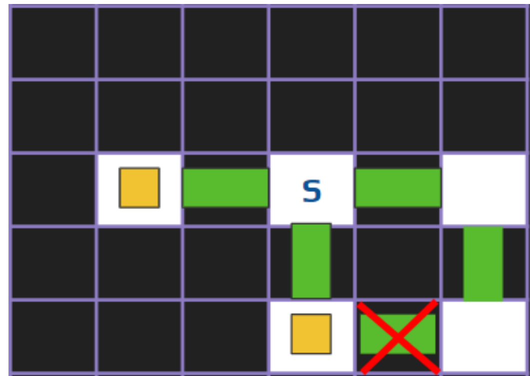
(b) The red node is placed first in the grid.



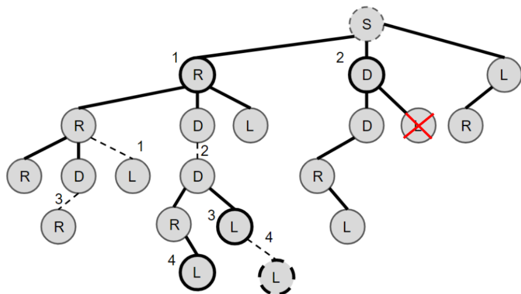

(c) The blue node is tested for placement, but there is already a room in the grid.



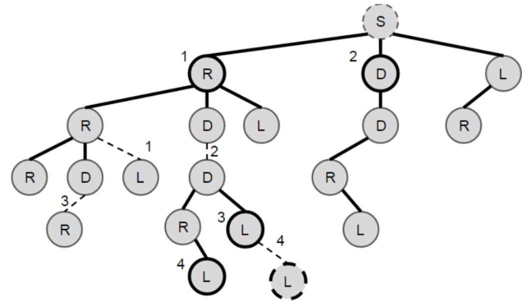

(d) The connection that would be created is discarded.




(e) The overlapping node is also discarded from the tree.



(f) The resulting tree, after fixing the overlap.



are changed into the $n_s$ special rooms, and sorted in the same way they were taken from their original branch.

For the enemies, the crossover's only restriction is not to place enemies in the start or the goal rooms. Therefore, we make a similar approach to the lock and key fix algorithm mentioned above: we save a list of how many enemies are in each room populated by enemies in each branch. If there are more rooms with enemies in the original branch than in the new one, we redistribute them. This redistribution tries to divide the number of enemies without room by the total number of rooms. All rooms receive the extra enemies if the division exceeds zero. If a remainder happens in the division, the first room of the branch receives the extra enemies.

After crossover, the mutation operator is executed if the *mutationRate* is satisfied in line 11 of Figure 25. Next, there is a 50% chance to execute one of the two mutation operators tailor-made for the tree structure: play-space and mission mutations. If the play-space mutation

**Require:** A population *pop*, which is the list of dungeons.
**Ensure:** Final population after $nGenerations$.
1: **for** $i = 0 \rightarrow nGenerations$ **do**
2: $newPop$ // Create a new empty population
3: // Fill the new population with new individuals
4: **for** $j = 0 \rightarrow Length(pop) \div 2$ **do**
5: $\lambda \leftarrow$ draw number within $[0,1]$
6: **if** $\lambda \leq crossoverRate$ **then**
7: $parent1, parent2 \leftarrow Tournament(pop)$
8: $child1, child2 \leftarrow Crossover(parent1, parent2)$
9: **end if**
10: $\lambda \leftarrow$ draw number within $[0,1]$
11: **if** $\lambda \leq mutationRate$ **then**
12: $Mutate(child1, child2)$
13: **end if**
14: $FitnessEvaluation(child1, child2)$
15: $newPop.Add(child1, child2)$
16: **end for**
17: $bestIndividual \leftarrow Elitism(pop)$
18: $newPop.first \leftarrow bestIndividual$
19: $pop \leftarrow newPop$
20: **end for**

Figure 25 – EA – As seen in (PEREIRA *et al.*, 2021)

is selected, there is a 50% chance of executing a leaf node addition or a leaf node exclusion. In the node-addition mutation, an NR node is inserted as a child for the selected leaf node. This change will only happen if the new node does not overlap the existing ones in the grid. In the node-exclusion mutation, we remove only the selected node if it is an NR. Otherwise, the mutation does not occur, keeping the node in the tree.

There are also two possible operations when the mission mutation is selected: addition or change-label of a key-lock pair. The mutation procedure visits the tree in a breadth-first fashion to add a pair, giving each visited NR node a chance to be converted into a KR. This probability is the same as the tree-creation algorithm. After the KR has been created, the next special room is an LR. If the pair is created, the algorithm stops immediately. Otherwise, it proceeds until the tree is fully visited. If no LR is created, then just a KR is added. The change-label procedure randomly selects a KR node to change. KR and LR (if available) from the selected pair are turned into a NR.

After this, still in mutation, we perform an enemy transfer operation. To do so, we select two rooms randomly: a donor and a receiver. If both rooms have no enemies, nothing is done. However, if the receiver has enemies and the donor does not have them, we swap the rooms, i.e., the donor becomes the receiver and vice-versa. Then, we transfer the enemies; to do so, we randomly chose from 1 to the donor's max number of enemies to move them to the receiver room. We repeat this operation without repositioning in the list of rooms from the dungeon until there are no more pairs available, but the operation is not performed in neither start nor goal room.

**Require:** Two individuals $ind1$ and $ind2$ to be crossed over, $cr$, two lists of rooms to save the special rooms that will be exchanged between branches: $spc1$ and $spc2$.
**Ensure:** If crossover occurs, return two new individuals.

1: $cc \leftarrow$ Draw crossover chance
2: **if** $cc \leq cr$ **then**
3: $CanTrade \leftarrow false$ and there are rooms left to draw in both dungeons
4: **while** $CanTrade = false$ **do**
5: $roomCut1 \leftarrow$ Draw random room from dungeon $ind1$ without replacement
6: $roomCut2 \leftarrow$ Draw random room from dungeon $ind2$ without replacement
7: $saveSpecialRooms(outspc1, outspec2)$
8: $clone1 \leftarrow clone(ind1)$
9: $clone2 \leftarrow clone(ind2)$
10: $clone1.Swap(roomCut1, roomCut2)$
11: $clone2.Swap(roomCut2, roomCut1)$
12: $RemakeGrid(clone1)$
13: $RemakeGrid(clone2)$
14: **if** $RoomsInBranch(clone1, roomCut2) < spc1 \parallel RoomsInBranch(clone2, roomCut1) == 1$ **then**
15: $continue$
16: **end if**
17: **if** $RoomsInBranch(clone2, roomCut1) < spc2 \parallel RoomsInBranch(clone2, roomCut1) == 1$ **then**
18: $continue$
19: **end if**
20: $CanTrade \leftarrow true$
21: **end while**
22: //If valid cut point exists, save the copies as new individuals
23: **if** $CanTrade = true$ **then**
24: $clone1.ExchangeEnemies(roomCut1, roomCut2)$
25: $clone2.ExchangeEnemies(roomCut2, roomCut1)$
26: $ind1 = clone1$
27: $ind2 = clone2$
28: $ind1.FixBranch(roomCut2)$
29: $ind2.FixBranch(roomCut1)$
30: **end if**
31: **end if**
32: **return** $ind1, ind2$

Figure 26 – Crossover, as seen in (PEREIRA *et al.*, 2021), but accounting for enemies

The player should explore all rooms and locks. If they do not explore, some paths of the level are not part of the locked door mission, which is generally undesirable in game design. Moreover, we implemented a version of the A* and of the DFS algorithms to simulate the behavior of a player traversing the dungeon created by the EA. We choose A* as it is one of the most computationally efficient path-finding algorithms widely used in the game industry. Its behavior simulates a player trying to reach the end of a dungeon as fast as possible, knowing only where the endpoint is. It is similar to *The Legend of Zelda* games, where the user has an item called the *Compass*, which shows only the location of the dungeon's boss. The number of locks opened by the A* is similar to a skilled player who knows this information. Meanwhile, the

DFS algorithm explores a path until it reaches a dead-end. This behavior is very similar to most players who have no previous information about the dungeon and are trying to map it. Therefore, the number of rooms visited by this algorithm would be close to the one regular players would visit.

Both algorithms must consider the rooms behind locked doors as unreachable until the KR containing its key is visited. Then, when a KR is visited by one of the path-finding approaches, the corresponding LR is immediately altered as “open” for the algorithm to visit. Its parent room, if already changed to “closed” (that is, already visited and its neighborhood explored), has its status changed to “open”, allowing the algorithm to search its neighborhood again and visit the once-locked room. As the DFS algorithm does not have a fixed order to select which child to visit, we select a random child who was not already visited when a node has more than one. This procedure adds to the behavior of different players, as each one may select a different route when playing. However, we must account for this randomness in fitness. So, we ran the DFS three times, each picking other random children to visit. A* algorithm returns how many locks it opened while searching for the final room, and the DFS returns how many rooms were visited during the search.

There are three terms in the fitness function. The first evaluates if the level layout follows the desired inputs provided by the user (game designer), how many of the locks the player must open to reach the end of the level, and how many rooms they must visit. Then, a second term is an extension of the enemy sparsity equation introduced by (SUMMERVILLE *et al.*, 2017) to evaluate the distribution of enemies in the 2D maps. The main objective of the EA is to create levels as close as users desire. Therefore, a double weight is given to user-related parameters. In the third term, we calculate the standard deviation of enemies in the rooms, inspired by the metrics in (SUMMERVILLE *et al.*, 2017), now the standard deviation for enemy tiles. Equation 4.1 has the final fitness expression.

$$L_{fitness} = f_{goal} - f_{es} + f_{std} \tag{4.1}$$

Our experiment to evaluate the first version of the enemy generator, described in Section 5.1, had only the the first factor (f_*goal*), as the dungeon did not evolve the enemy placement at the time. The method followed the same setting in Pereira et al.’s work (PEREIRA *et al.*, 2021) as shown in Equation 4.2.

$$\begin{aligned} f_{goal} = 2 * (|u_{\mathrm{r}} - d_{\mathrm{r}}| + |u_{\mathrm{k}} - d_{\mathrm{k}}| + |u_{\mathrm{l}} - d_{\mathrm{l}}| + |u_{\mathrm{lin}} - d_{\mathrm{lin}}|) \\ + \Delta_{\mathrm{l}} + \Delta_{\mathrm{r}} \end{aligned} \tag{4.2}$$

The user provides as input for EA the desired number of rooms ($u_r$), keys ($u_k$), locked doors ($u_l$), and the level linearity ($u_{lin}$). Our EA will search for the best level whose number of

rooms ($d_r$), keys ($d_k$), locked doors ($d_l$), and linearity ($d_{lin}$) are close enough to the user's input. Therefore, the first four terms in Equation 4.2 will measure how far is the generated level from the user's specification. The last two terms evaluate the exploration of the level based on outputs returned by path-finding algorithms. Equation 4.3 calculates how many rooms were explored by the DFS algorithm.

$$\Delta_\mathrm{r} = d_r - d_r^{DFS} \tag{4.3}$$

Equation 4.4 calculates how many locks were opened by the A* algorithm.

$$\Delta_\mathrm{l} = d_l - d_l^{A*} \tag{4.4}$$

As the DFS selects random children to visit when the parent has more than one, we execute it three times for each dungeon, aiming to reduce the impact of the randomness. Thus, $d_r^{DFS}$ returns the average of the number of visited rooms by DFS, and $\Delta_\mathrm{r}$ gives us the number of non-visited rooms. In Equation 4.4, we have the difference between the total number of locks ($d_1$) in the level and the number of locks opened by the A* ($d_l^{A*}$). The linearity feature is based on the average degree of nodes in a graph, as given by Equation 4.5. The parameter $nodes^{Inner}$ is the tree's number of inner nodes. A $d_{lin}$ closer to 1 leads to a linear dungeon, while large $d_{lin}$ values provide a maze-like level with more paths through multiple doors.

$$d_{lin} = (d_r - 1)/nodes^{Inner} \tag{4.5}$$

The second factor measures the distribution of enemies in the 2D map, aiming the dispersion of enemies in the levels' rooms. Thus, larger values mean more distribution. It is presented in Equation 4.6 where $e_x$ and $e_y$ are the x-position and y-position of an enemy $e$, $\mu_x$ and $\mu_y$ are the average x-position and y-position of all enemies, and $E$ is the set of enemies.

$$f_\mathrm{es} = \frac{\sum_{e \in E}(e_x - \mu_x)^2 + (e_y - \mu_y)^2}{Number\ of\ Enemies} \tag{4.6}$$

In the third term, we calculate the standard deviation of enemies in the rooms, inspired by the metrics in (SUMMERVILLE *et al.*, 2017), now the standard deviation for enemy tiles. This function encourages the rooms to have a balanced number of enemies (lower values imply enemies are evenly distributed) and is shown in Equation 4.7. $r$ is a room in the set of rooms $R$, $r_{enemies}$ is the number of enemies in a room, $\mu_{enemies}$ is the average number of enemies in the rooms, and $|R|$ is the number of rooms, discarding the starting and goal rooms since they cannot have enemies.

$$f_{\text{std}} = \sqrt{\frac{1}{|R|}\sum_{r\in R}(r_{enemies} - \mu_{enemies})^2} \tag{4.7}$$

We subtract this enemy sparsity $f_{es}$ component is our fitness 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. We took $f_{goal}$ from (PEREIRA *et al.*, 2021)'s work and gave equal weights for the other two once preliminary tests showed bias towards the placement of enemies or the dungeon layout. We wanted them both to be close to the desired input.

After the experiments with the offline setup and their respective results, described in Section 5.2, we would need a fitness function more stable for the online setup. Furthermore, the method must converge earlier if it finds good enough results. In an online setup, the generated content must always generate exciting solutions for the player and converge as early as possible, aiming not let players get bored in a loading screen.

To do so, we changed the stop criteria of our algorithm to consider early convergence besides the maximum number of generations. The first criterion for premature convergence is if an individual has a fitness value smaller than a given value (we used 0.1 in the experiments in Section 5.3), which would be a remarkable convergence considering our preliminary computational experiments. The second criterion is if there is no decrease in the best fitness value for a given number of generations (30, in the experiments in Section 5.3). Either way, the generation stops after a fixed number of generations, 200 in the experiment, as mentioned earlier.

We also updated our fitness equation, as it needed to be more precise on the value, as the minimum value was a stop criterion, and also to guarantee a good convergence for different input ranges. Therefore, we applied standardization techniques. First, we fixed 4.6 from (VIANA, 2022), so it correctly represents the sparsity as proposed in (SUMMERVILLE *et al.*, 2017), which is the standard deviation of the distance between enemies [7]. Then, we divided the value by the average, standardizing the standard deviation as the coefficient of variation, better suited to compare data with different means, as in our case.

For better understanding, we first give the equation of the enemy distance in Equation 4.8, where $e_x$ is the enemy's x position, $e_y$ is the enemy's y position, and $\mu_x$ and $\mu_y$ are the average x and y position, respectively, for all enemies in the dungeon. Then, we calculate the mean enemy Euclidean distance in Equation 4.9, where E is the set of enemies in the dungeon (and $|E|$ the number of enemies in the dungeon) and $d_e$ the distance from the previous equation. Then, Equation 4.10 shows the new equation taking place in the third term of our fitness function as the enemy's distance coefficient of variation. $d_e$ and $d_{mean}$ are the ones from the previous equations.

[7] In their publication, they incorrectly use the variance instead of the standard deviation, so we use the square root of their equation, as the correct standard deviation

$$d_e = \sqrt{(e_x - \mu_x)^2 + (e_y - \mu_y)^2} \tag{4.8}$$

$$d_{mean} = \frac{\sqrt{\sum_{e \in E} d_e}}{|E|} \tag{4.9}$$

$$f_{edcv} = \sqrt{\frac{\sum_{e \in E}(d_e(d_e - d_{mean})^2)}{|E|}} * \frac{1}{d_{mean}} \tag{4.10}$$

We apply the coefficient of variation to the standard deviation of the number of enemies in each room from Equation 4.7, resulting in Equation 4.11. Suppose, by any chance, that all enemies are in the same room, which is not intended and will cause a division by zero on the mean value. In that case, a penalty of 100 is given to this part of the fitness, as well as the previous part, from the coefficient of variation of the distance.

$$f_{ecv} = \sqrt{\frac{1}{|R|} \sum_{r \in R} (r_{enemies} - \mu_{enemies})^2} * \frac{1}{\mu_{enemies}} \tag{4.11}$$

We also have separated the fitness goal from Equation 4.2 in their counterparts of the distance from the input, in Equation 4.12 and the usage of rooms and locks in Equation 4.13

$$f_{input} = |u_{\mathrm{r}} - d_{\mathrm{r}}| + |u_{\mathrm{k}} - d_{\mathrm{k}}| + |u_{\mathrm{l}} - d_{\mathrm{l}}| + |u_{\mathrm{lin}} - d_{\mathrm{lin}}| \tag{4.12}$$

$$f_{usage} = \Delta_{\mathrm{l}} + \Delta_{\mathrm{r}} \tag{4.13}$$

Thus, we have our updated fitness function in Equation 4.14. Note that instead of subtracting the value of the enemy's distance, we inverted it, as we want to minimize our fitness at zero, preventing negative numbers. We also gave a weight of 3 to the input part of the fitness, as it is the most important part (matching the user's desire), and a weight of 2 to the usage part, as it is also critical that the dungeon configuration is well-used. We allow some inconsistency in the enemies' distribution throughout the rooms and the number of them in each room.

$$L_{fitnessonline} = 3 * f_{input} + 2 * f_{usage} + 1/f_{edcv} + f_{ecv} \tag{4.14}$$

Moreover, to further normalize the fitness and aid the early convergence to happen when good levels are selected, independent of the dungeons parameters, we applied a min-max normalization to our population's fitness in each generation. After every fitness from the current generation is calculated, each part of Equation 4.14 is normalized from 0 (the minimum possible value) to the maximum value found during the evolution process.

### 4.3.2 MAP-Elites for Dungeon Generation

To allow more diversity in the generated levels, we also used a MAP-Elites algorithm to generate our levels. This MAP-Elites approach was used in the offline experimental setup (Section 5.2), with positive results. But we preferred the EA approach described previously for the online experimental setup, as it could converge faster and because we wanted the players to play the best level possible at every playthrough.

The MAP-Elites-based dungeon generator uses the same genotype and phenotype for the dungeon, the same operators for crossover and mutation, and the same fitness function. The significant difference is the core loop of the algorithm and the mapping of feature descriptors (or dimensions). We decided to map the leniency of enemies against the exploration coefficient of the dungeon. The leniency, adapted from Smith et al.'s work, is the number of safe rooms, i.e., without enemies, divided by the total number of rooms (SMITH; PADGET; VIDLER, 2018). While the exploration coefficient is inspired by the exploration measure introduced by Liapis et al.: we run a flood fill algorithm between rooms to simulate the map coverage, where the reached rooms represent the critical exploration from each starting room and its corresponding goal room (LIAPIS; YANNAKAKIS; TOGELIUS, 2013).

Leniency was chosen as it could bring variety in how many rooms were occupied with enemies, forcing them to be more spread with lesser leniency, and more concentrated otherwise. The exploration coefficient brought variety in the difficulty of the locked-doors puzzle: the larger the exploration coefficient, the farther apart was a key from its door. Both metrics can have different values throughout the evolution, and may appeal differently to distinct player profiles.

The leniency metric is shown in Equation 4.15, adapted from Smith et al.'s work (SMITH; PADGET; VIDLER, 2018). The metric is calculated by the number of safe rooms, i.e., without enemies, divided by the total number of rooms.

$$D_{\text{leniency}} = \frac{\mathit{Number\ of\ Safe\ Rooms}}{\mathit{Total\ Rooms}} \tag{4.15}$$

Equation 4.16 measures our exploration coefficient, inspired by the exploration measure introduced by (LIAPIS; YANNAKAKIS; TOGELIUS, 2013). We run a flood fill algorithm between rooms to simulate the map coverage, where the reached rooms represent the critical exploration from each starting room and its corresponding goal room. $RR$ 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.

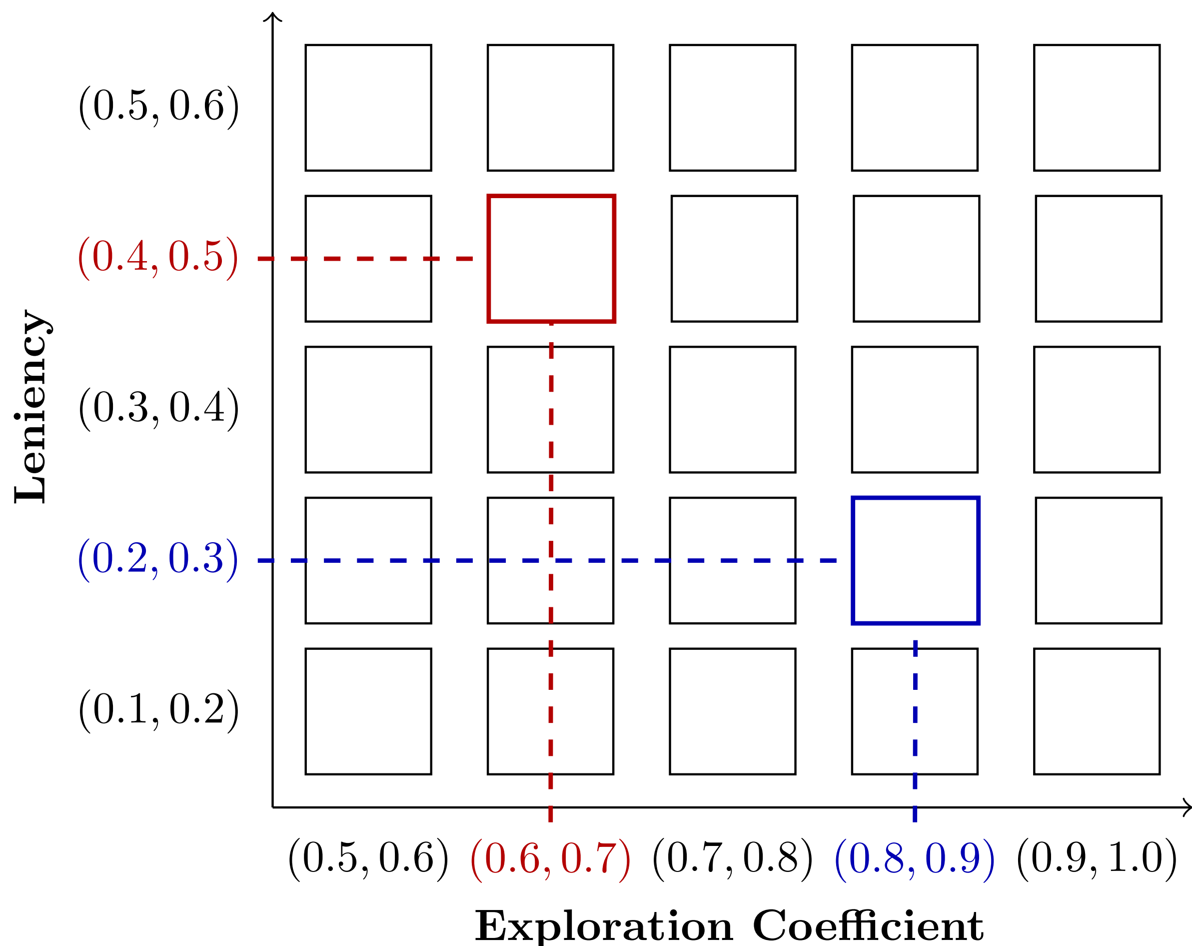
$$D_{\text{exploration}} = \frac{1}{\#RR} \sum_{(r_s, r_g) \in RR} \frac{\mathit{Coverage}(r_s, r_g)}{\mathit{Total\ Rooms}} \tag{4.16}$$

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 the 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 fewer exploration coefficient values lead to rooms much closer to each other. Figure 27 presents our approach's map.

Figure 27 – 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 besides the red one, and it also has more rooms with enemies.

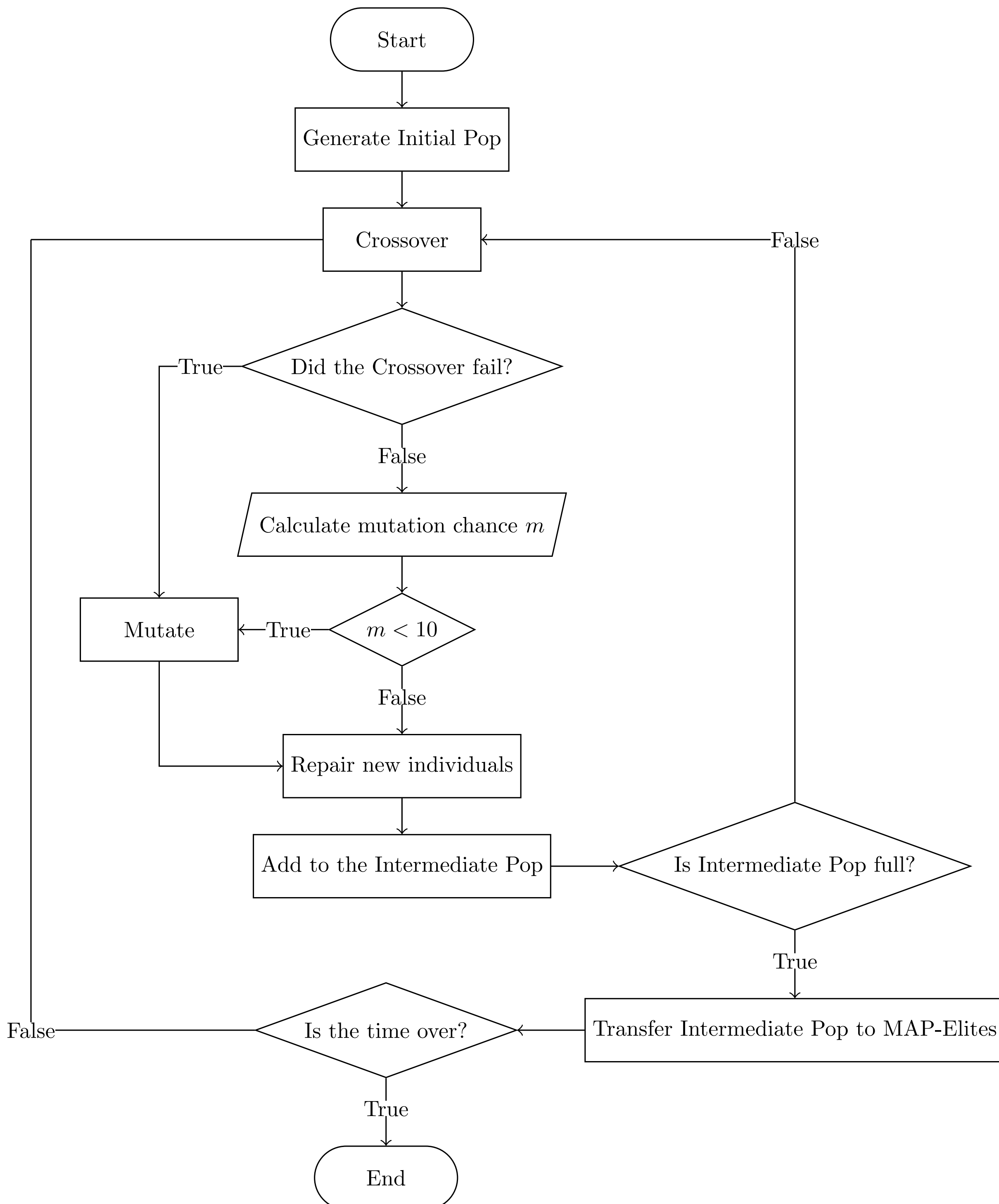



The proposed MAP-Elites application will map 25 individuals based on the defined intervals. When the map receives a new individual, we must calculate the feature descriptors to place it in the correct entry of the MAP-Elites table. If an individual fills a map cell and a new one hits the same cell, the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

Figure 28 presents the flowchart as an overview of our approach. The evolutionary process starts generating individuals for the initial population, i.e., levels with rooms, keys,

and locks, by following the initialization algorithm described in Section 4.3.1, that was present in Pereira et al.'s previous works on dungeon generation (PEREIRA *et al.*, 2021; PEREIRA; PRADO; TOLEDO, 2018). We add individuals to the initial population until it reaches $n$. 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, since the best individual is always kept for that entry.

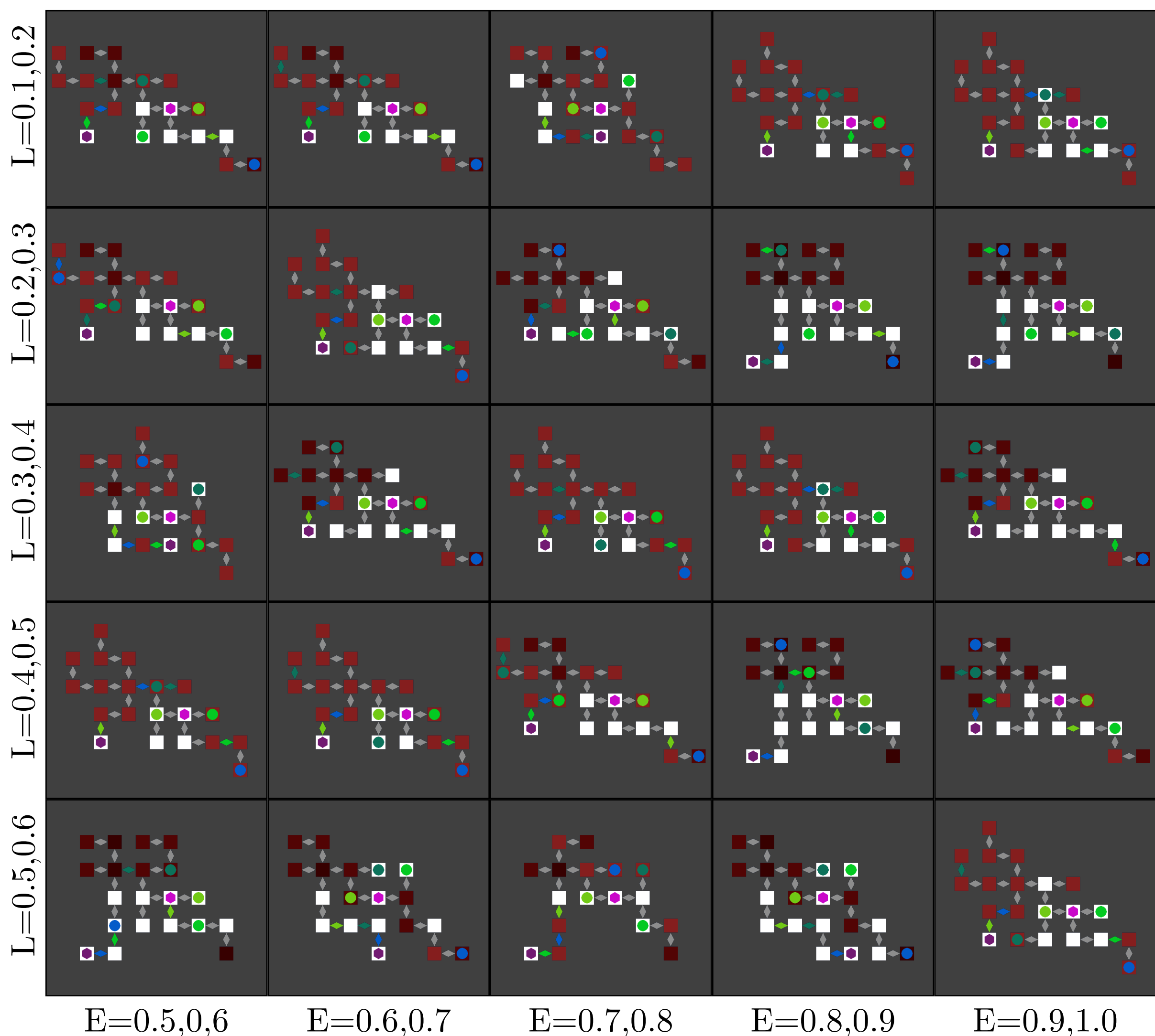
Figure 28 – Flowchart with an overview of our approach. Pop abbreviates population.



Next, we evolve the population using the time-limit stop criterion. In the EA approach described before and Pereira et al.'s approach, we create an intermediate population that always replaces the current one, except by the best individual found so far (PEREIRA *et al.*, 2021). 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 from two parents, chosen by tournament selection with two competitors.

This MAP-Elites approach allows us to create a grid of dungeons, as presented in Figure 29. It will be used in our offline experimental setting, documented in Section 5.2.

Figure 29 – Example of a MAP-Elites population evolved for levels with 20 rooms, 4 keys, 4 locks, 30 enemies, and linear coefficient equal to 2. 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.



### 4.3.3 Evolutionary Algorithm for Enemy Generation

Another important content for a game is its enemies. Most PCG approaches to enemy generation only handle the placement of human-made enemies in a level, according to its structure or the player's performance. We create new enemies with different behavior, statuses, and patterns for each play-through using an Evolutionary Algorithm, presented in this section, and a MAP-Elites approach, presented next, in Section 4.3.4.

For the EA approach, we used Unity's DOTS architecture, ideal for evolving a large population of individuals in a reduced time frame[8], which will be useful to generate different behaviors.

To define our representation, we extracted the most common variables from enemies in different games, focusing on the action-adventure genre. After careful consideration, we came up with the variables in Table 3 that contains the variable type, its possible range, and a brief description of their impact on gameplay. After testing our prototype with different players, we empirically set the range for each variable. The selected intervals provided enemies from easy (but not boring) to hard (but not humanly impossible) configurations. The nominal variables in Table 3 are described in detail by Tables 4 and 5. These variables can be extended as much as the designer likes by creating new behaviors that fit their game.

The use of software architecture (like delegates and Unity's Scriptable Objects) makes it relatively easy to add a plethora of new behaviors, weapons, and parameters, as well as change their appearance for each possible combination of them. We can do all the mentioned addition without altering the EA itself, which will still evolve feasible solutions that converge to the designer's desired fitness.

In the initial population, these parameters are selected randomly, considering the allowed range of integers and float values and the size of the set of movements and weapons for the nominal values. The population evolves using the operators described in Section *Operators*, which happens until a certain number of generations. When the algorithm stops, we save the data of the $n$ best individuals (defined by the designer). Therefore, we use the ability of the EA to evolve whole populations of solutions to have a collection of suited individuals in a single execution. We discuss in Section *Game Prototype* how we handle the data from the resulting enemies in our game prototype.

The fitness function sums all parameters, weighting some according to designer-input values. Equation 4.17 shows the function. $hp$ is the health of the enemy. $dmg$ is the damage it does to the player, which is multiplied by the designer-input weight of the weapon it currently holds ($weapon$). $spd$ is the movement speed that multiplies the designer-input weight of the enemy's movement behavior ($movement$). The time the enemy rests between cycles of active movement ($rest$) has its value inversely proportional to the fitness, as the lower it is, the more

[8] <https://unity.com/dots>

Table 3 – Parameters, variable type, range, and their descriptions for the enemy's genotype in the EA.

| Parameter | Type | Range | Details |
|---|---|---|---|
| Health | integer | 1-5 | Total health. |
| Damage | integer | 1-4 | Damage done. |
| Attack Speed | float | 0.75-4.0 | Projectile's shot frequency (1/Attack Speed). |
| Movement Speed | float | 0.8-3.2 | Multiplies movement direction's vector. |
| Active Time | float | 1.5-10 | Time (in sec.) that the enemy moves before resting. |
| Rest Time | float | 0.3-1.5 | Time (in sec.) the enemy rests before moving. |
| Projectile Speed | float | 1-4 | Multiplied by the projectile's trajectory vector. |
| Movement Type | nominal | x | Calculates direction vector of movement at each frame. |
| Weapon Type | nominal | x | Enemy's weapon. Each may have different properties. |

difficult the enemy tends to be. *active* is the time the enemy moves without stopping before resting. $proj$ is a variable that receives the value one if the enemy can throw any projectile and 0 otherwise. This value is multiplied by the sum of the projectile's speed ($projsped$) and the rate at which the enemy throws said the projectile, denoted as the attack speed ($atkspd$). The fitness value is directly proportional to the enemy's expected difficulty, and the algorithm tries to match this value to the one input by the designer before the evolution began.

$$\begin{aligned} f = hp + dmg * weapon + spd * movement + \\ 1/rest + active + proj * (atkspd + projspd) \end{aligned} \tag{4.17}$$

This enemy generator EA can generate the enemies shown in Figure 30, except the healer, and was tested with players as presented in Section 5.1. As the enemies were well received, we enhanced the algorithm with a MAP-Elites approach, as will be presented in the next Section (4.3.4), and added the healer enemy.

Table 4 – Each movement type that the enemy can have, its weight in the fitness function and the working details.

| Movement Type | Weight | Details |
|---|---|---|
| None | 0 | Stay still |
| Random | 1.04 | Selects a random direction vector (x, y) to move towards in the actual active cycle. |
| Random1D | 1 | Selects x or y-axis from a random direction vector to move towards in current cycle. |
| Flee | 1.1 | Direction vector (x, y) opposes the player's direction. |
| Flee1D | 1.08 | Selects x or y-axis from the player's location opposing vector in the current cycle. |
| Follow | 1.15 | Direction vector (x, y) points towards the player's direction. |
| Follow1D | 1.12 | Selects the x or y-axis of the vector towards the player's location in the current cycle. |

Table 5 – Each weapon type available, its weight in the fitness function and the details. The projectiles' weights(*) are lower as they are multiplied by the projectile speed and attack speed.

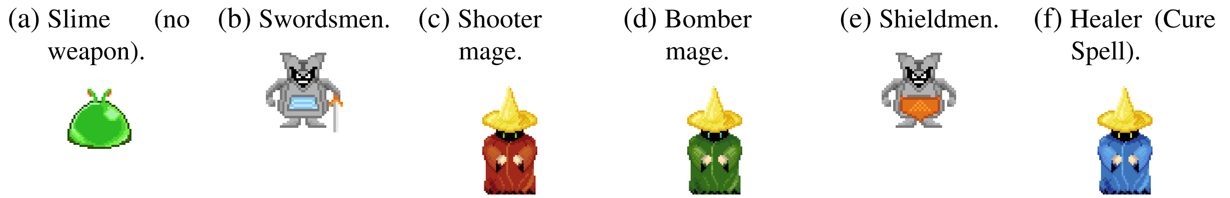
| Weapon | Weight | Details |
|---|---|---|
| None | 1 | Damage on contact. |
| Sword | 1.5 | Holds a sword in front of itself. Increases reach. |
| Shield | 1.6 | Protects the enemy from frontal attacks. |
| Bullet | 0.3* | Shoots a bullet towards the player. Damages on contact. |
| Bomb | 0.3* | Shoots a bomb towards the player. Explodes in 2 seconds. |

### 4.3.4 MAP-Elites for Enemy Generation

This section describes our enemy representation and how we evolve them through our MAP-Elites approach. We use the same representation, crossover, and mutation operators as in the EA version. Our only change in the representation is the numerical values of the max movement speed; we decreased it slightly (from 3.2 to 2.8) because the max value was too fast.

We add the cure spell to generate healer enemies, and our melee enemies use the following

Figure 30 – List of enemies of our game prototype. Slimes have no weapon. Swordsmen use swords. Bower mages shoot arrows. Bomber mages throw bombs. Shieldmen hold shields. Healers use cure spell to heal other enemies.



(a) Slime (no weapon). (b) Swordsmen. (c) Shooter mage. (d) Bomber mage. (e) Shieldmen. (f) Healer (Cure Spell).

weapons: barehanded, 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.

The input for the generation, as in the EA version, is the desired 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 and presented in equations 4.18 and 4.19, where $e$ is the enemy.

$$D_{\text{movement}} = e_{movement_type} \tag{4.18}$$

$$D_{weapon} = e_{weapon_type} \tag{4.19}$$

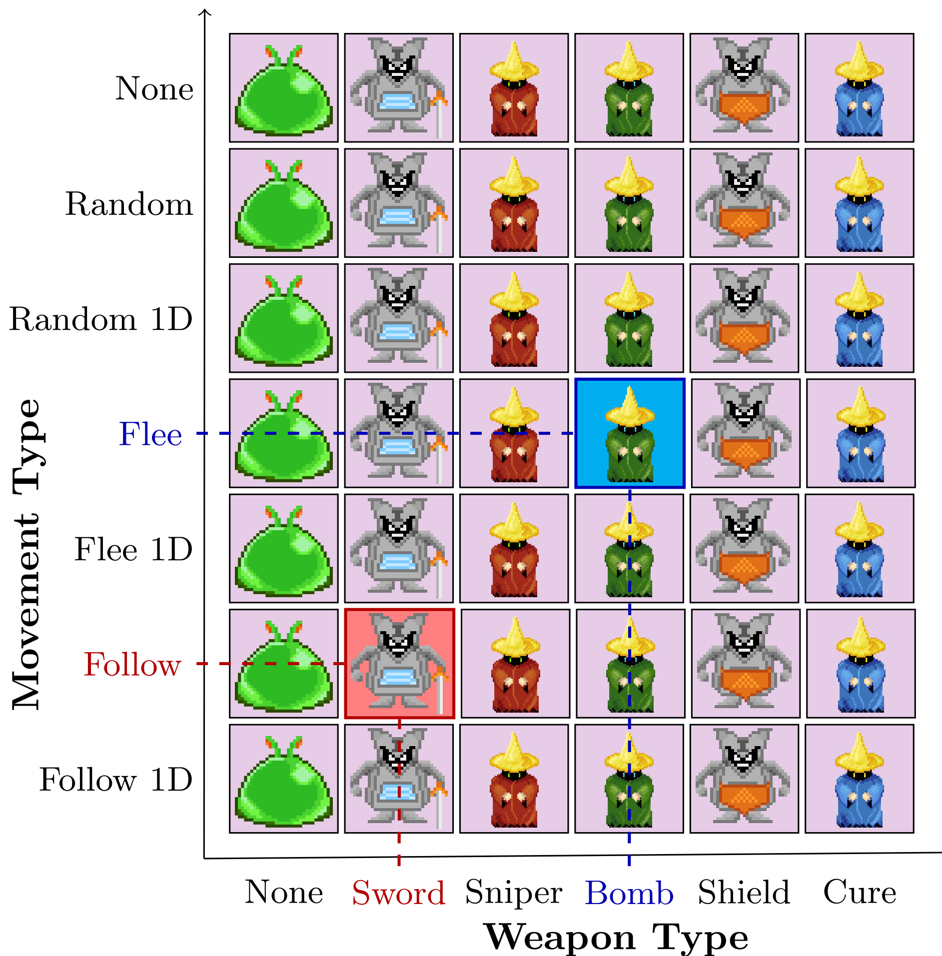
Figure 31 presents our map approach. The cell highlighted in red represents an enemy that follows the player to hit with a sword, while blue represents an enemy that flees from the player while throwing bombs toward them.

The proposed MAP-Elite maps 42 enemies, and 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 randomly generating the initial population filling the attributes from $n$ enemies. 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 as a stopping criterion. The authors in (PEREIRA *et al.*, 2021) replace all the population in each generation with the intermediate population generated. We also have an intermediate population; however, we try

Figure 31 – 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.




adding its individuals to the MAP-Elites population. The reproduction operators create new individuals from two parents, chosen through tournament selection with two competitors.

We first perform a crossover with a 100% rate to generate two new individuals when reproducing enemies. Our crossover is a combination of a fixed-single-point crossover, and a BLX-$\alpha$ crossover (ESHELMAN; SCHAFFER, 1993). We first cross the parents in the fixed point as shown in Table 3. 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-$\alpha$ 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 multigene 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, shown in Equation 4.20. Regarding the movement factor, we consider three attributes of our individuals: movement speed, active time, and rest time, shown in Equation 4.21. There, *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 is actively 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 easier it will be to defeat; thus, we calculate its inverse.

$$d_{health} = 2 \times e_{health} \tag{4.20}$$

$$d_{movement} = e_{mv_spd} + e_{act_tm}/3 + 1/e_{rst_tm} \tag{4.21}$$

The strength is more complex than the previous difficulty factors; it depends on the types of enemies. Therefore, we multiply three different equations, shown in Equation 4.22. For melee enemies, we multiply damage by movement speed, as shown in Equation 4.23, where *dmg* is the damage. We multiply attack speed by projectile speed for ranged enemies and weigh the result by three, as in Equation 4.24, 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 healing area range, as in Equation 4.25.

$$d_{strength} = d_{s_1} \times d_{s_2} \times d_{s_3} \tag{4.22}$$

$$d_{s_1} = \begin{cases} e_{dmg} \times e_{mv_spd}, & \text{isMelee}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.23}$$

$$d_{s_2} = \begin{cases} 3 \times (e_{atk_spd} \times e_{prjct_spd}), & \text{isRanged}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.24}$$

$$d_{s_3} = \begin{cases} 2 \times e_{atk_spd}, & \text{isHealer}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.25}$$

We also calculated the gameplay 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, as shown in Equation 4.26

$$d_{gameplay} = d_{g_1} \times d_{g_2} \times d_{g_3} \times d_{g_4} \times d_{g_5} \tag{4.26}$$

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 in Equation 4.27.

$$d_{g_1} = \begin{cases} 1.25, & \textsc{isMelee}(e) \text{ and } \textsc{isFollow}(e) \\ 0.5, & \textsc{isMelee}(e) \text{ and } \\ & (\textsc{isAnyFlee}(e) \text{ or } \textsc{hasNoMove}(e)) \\ 1, & \text{otherwise} \end{cases} \tag{4.27}$$

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 a projectile speed faster than their movement speed, or they will not behave as rangers since their projectiles will be slower than they own. Thus, equation 4.28 represents these weights. This behavior did not occur in enemies with the movement Follow1D. Therefore, we weight the difficulty as in Equation 4.29

$$d_{g_2} = \begin{cases} 1.25, & \textsc{isRanged}(e) \text{ and } \textsc{isFlee}(e) \\ 1.15, & \textsc{isRanged}(e) \text{ and } \textsc{isFlee1D}(e) \\ 0.5, & \textsc{isRanged}(e) \text{ and } \textsc{hasNoMove}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.28}$$

$$d_{g_3} = \begin{cases} 0.5/(2 \times e_{mv_spd}), & \textsc{isRanged}(e) \text{ and } \\ & \textsc{isFollow}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.29}$$

Healers must protect themselves while keeping healing other enemies. Thus, they should not follow players but avoid them, as shown in Equation 4.30. Besides, faster healers are more difficult to defeat; thus, we also weighed this factor by their movement speed. Therefore, we weigh the enemy's difficulty as in Equation 4.31.

$$d_{g_4} = \begin{cases} 1, & \textsc{isHealer}(e) \text{ and } \\ & (\textsc{isAnyRandom}(e) \text{ or } \\ & \textsc{isAnyFlee}(e)) \\ 0.5, & \text{otherwise} \end{cases} \tag{4.30}$$

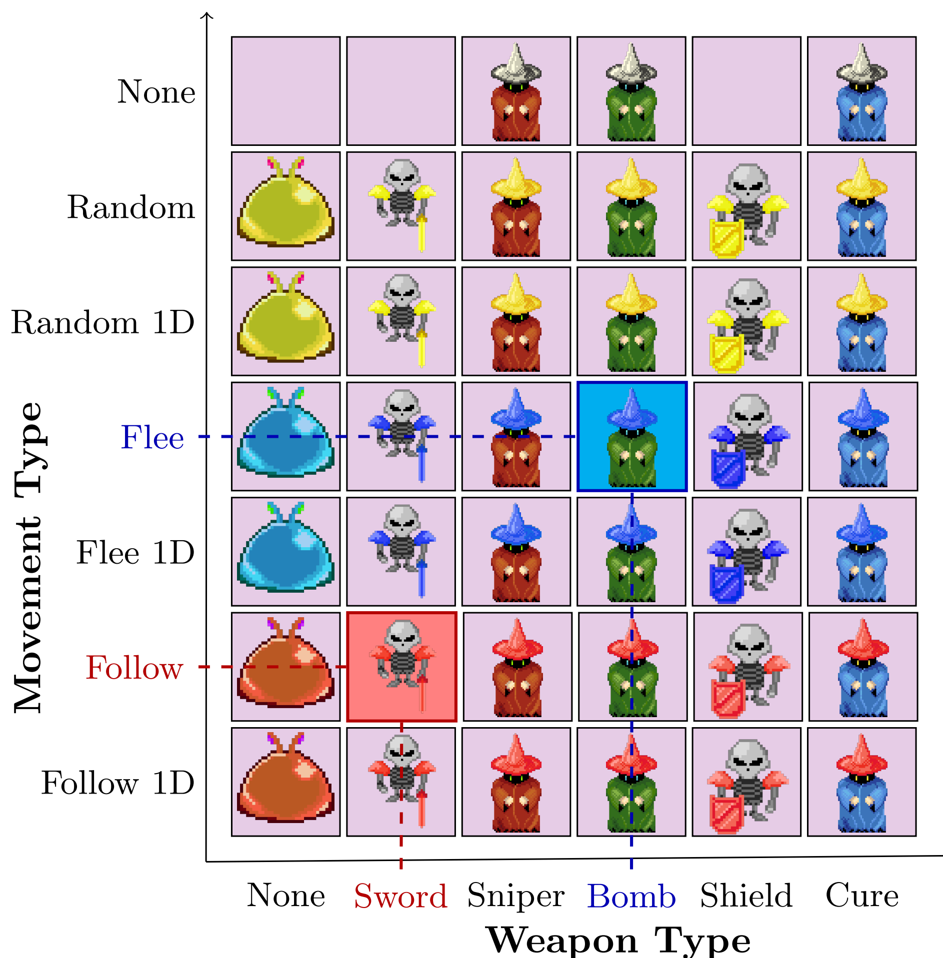
$$d_{g_5} = \begin{cases} 1.15 \times e_{mv_spd}, & \textsc{isHealer}(e) \\ 1, & \text{otherwise} \end{cases} \tag{4.31}$$

Through gameplay experiments, we have empirically chosen all the numeric weights in the equations. Finally, we defined the final difficulty in Equation 4.32.

$$d = d_{gameplay} \times (d_{health} + d_{movement} + d_{strength}) \tag{4.32}$$

This MAP-Elites algorithm to create enemies is used both in the offline and online experimental setups (respectively, Sections 5.2 and 5.3). However, we added new visuals for the swordsman and shieldman for the online setup. Moreover, every enemy changed their color patterns according to their movement: random movements are represented by yellow enemies, flee patterns are blue, follows are red, and no movement is gray. Figure 32 shows these color patterns and their respective movement types.

Figure 32 – The map of MAP-Elites population for the online experimental setup. 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.




We also note that this approach has been published in (VIANA; PEREIRA; TOLEDO, 2022), and the reader may refer to this paper for more details on its implementation and discussions about the quality of the generated enemies.

### 4.3.5 Formal Grammar Quest Generator

We generate quests a using simple formal grammar as proof of concept. The idea was to evaluate the system's ability to create contents for quests as a starting frame. These quests are based on the works of Hartsook et al. (HARTSOOK *et al.*, 2011) and Doran and Parberry (DORAN; PARBERRY, 2011), selecting quests that our current game could handle best while trying to make it representative for each profile.

The grammar's 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). These quests are created through grammar $Gr$ where $N = \{S, W, G, K, E\}$, $\Sigma = \{wander, get, kill, explore\}$. Fig. 33 presents our production rules, with $S$ being the start symbol.

Figure 33 – Quest generating grammar $Gr$.

1. $S \rightarrow W|G|K|E$
2. $W \rightarrow wanderW|wander$
3. $G \rightarrow getG|get$
4. $K \rightarrow killK|kill$
5. $E \rightarrow exploreE|explore$

To start the grammar, we use the weights the Profile Analyst system provides, as described in Section 4.2.1. A loop begins for each profile weight and creates at least one quest of each type, generating the non-terminal for that quest. Each non-terminal has 50% chance of selecting its two rules; therefore, we can create a chain of quests at each iteration. Algorithm 5 contains the steps that process the grammar and generate quests given the profiles' weights. The system allows a maximum of four iterations of that loop per profile weight.

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

```
1: newQuestChance, chainCost ← 0
2: repeat
3:     for all profileWeights > newQuestChance do
4:         process non-terminal of corresponding quest
5:     end for
6:     chainCost ← chainCost + 2
7:     newQuestChance ← Random(7) + chainCost
8: until chainCost < 7
```

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 4.2.2) and uses 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 4.3.3), 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:

- Number of rooms: 16, 20, 24, 28, 32
- Number of keys and locks: 3, 4, 5, 6, 7
- 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.

### 4.3.6 Markov Chain Stochastic Grammar Quest Generator

To further improve the quest content, we use Stochastic Context-Free Grammar in the new quest generator. The main goal is to guarantee quest diversity while attending to the player's preferences. The terminal symbols are quests, once again based on the work of Doran and Parberry (DORAN; PARBERRY, 2011). The non-terminal symbols are used to separate quests related to different motivations (or player profiles). This way, we define our grammar $Gr_2$, where N = { S, C, A, M, I}, Σ = { $\varepsilon$, exchange, explore, gather, give, goto, kill, listen, read, report}. Fig. 34 presents the new production rules, with S being the start symbol.

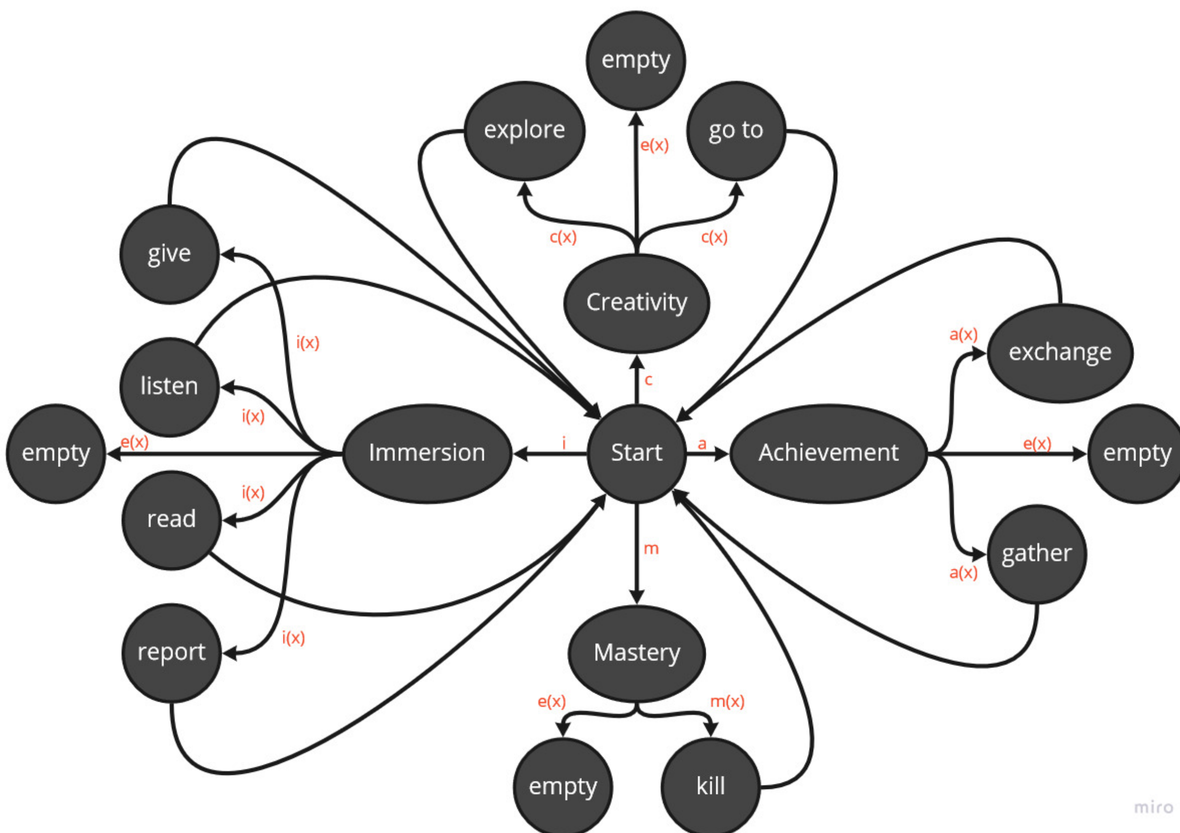
If the non-terminal symbol *C* is applied, the obtained quest line will be related to the creative player profile. This rule's terminal symbols are *explore* and *goto*. These quests involve immersing the player in the game, encouraging them to explore locations and go to different rooms in the dungeon. The non-terminal symbol *A* refers to quests of the achievement player profile. The possible terminal symbols this achieves are *gather* and *exchange*. These represent quests that revolve around obtaining items through different methods. The player might be encouraged to get items for themselves or exchange them with game agents for different items.

Figure 34 – Quest generating grammar $Gr_2$.

1. $S \rightarrow$ C | A | M | I
2. $C \rightarrow$ explore C |goto C |$\varepsilon$
3. $A \rightarrow$ gather A |exchange A |$\varepsilon$
4. $M \rightarrow$ kill M |$\varepsilon$
5. $I \rightarrow$ listen I | read I | give I | report I |$\varepsilon$

The mastery player profile is correlated to the *M* non-terminal symbol. The only terminal symbol obtained by *M* is a *kill*. This quest aims to defeat different enemies. Finally, the last non-terminal symbol, *I*, relates to the immersion player profile. The terminal symbols that can be obtained by *I* are *listen*, *read*, *give*, and *report*. The quests in this line will mostly be related to interacting with game agents. We observe these relations in Figure 35. There's a tenth terminal symbol we haven't mentioned, which is the *empty* one. This symbol does not belong to any particular profile; instead, it indicates when the derivation is over. Algorithm 6 contains the algorithm for $Gr_2$'s derivation.

Figure 35 – All achievable derivations.




When selecting the next symbol to derive, a random number between 0 and 100 is generated. Every derivable symbol's probability is added to a sum of all the possible symbol's

**Algorithm 6** – Derivation algorithm.

$derivation \leftarrow newDerivation()$
**while** $derivation.GetLastSymbol().CanDrawNext$ **do**
    $lastSelectedQuest \leftarrow derivation.GetLastSymbol()$
    $lastSelectedQuest.NextSymbolChances \leftarrow ProfileCalculator.StartSymbolWeights$
    $lastSelectedQuest.SetNextSymbol(derivation)$
    $nonTerminalSymbol = derivation.GetLastSymbol()$
    $nonTerminalSymbol.SetNextSymbol(derivation)$
    $derivation.GetLastSymbol().DefineQuestSo(Quests, generatorSettings)$
**end while**

probabilities. The last summed symbol will be derived if the current sum is greater or equals the generated number. Otherwise, the algorithm will continue the process until a symbol is derived. We show this in Algorithm 7.

**Algorithm 7** – Algorithm that selects the next derived symbol. This function is invoked as *SetNextSymbol()* in Algorithm 6.

$newQuestChance \leftarrow Random(100)$
$cumulativeProbability \leftarrow 0$
**for all** nextQuest in nextQuests **do**
    $cumulativeProbability \leftarrow cumulativeProbability + nextQuest.chance(symbolNumber)$
    **if** $cumulativeProbability >= newQuestChance$ **then**
        $derivation.SetSymbol(nextQuest.symbol)$
        **break**
    **end if**
**end for**

The player's quest preferences provide the probability of $Gr_2$ deriving from the starting symbol to a non-terminal symbol (probabilities $c$, $a$, $m$ and $i$, shown in Figure 35). These probabilities are calculated based on the score given to each player profile during the questionnaire before the generation process. In this questionnaire, the players give a weight from 1 to 5 for each preference, with 5 normalized to 1.0 and 0 normalized to 0.0 (therefore, the minimum weight a player will give to a quest is 0.2). Then, we normalize the values of each preference dividing by the sum of preferences, transforming the sum into 1. The final result is a set of probabilities for each player profile used to derive $Gr_2$ from the start symbol to a non-terminal symbol. We summarize this process in Equation 4.33, where $ps_i$ is the player's score on each profile.

$$pref_{\mathrm{i}} = \frac{ps_i}{\sum_{j=1}^{4} ps_j} \tag{4.33}$$

The probability of a non-terminal symbol deriving to a terminal symbol (probabilities *a(x)*, *c(x)*, *e(x)*, *i(x)* and *m(x)* in Figure 35) is determined by a function. These functions are based on the total number of possible quests for each non-terminal symbol and should be determined based on the condition stated by Equation 4.34. There, $f(x)$ is the probability function that will

be determined, $e(x)$ is the probability function of the grammar deriving to the *empty* symbol, and $n$ is the total number of all possible terminal symbols from a profile, except the *empty* symbol.

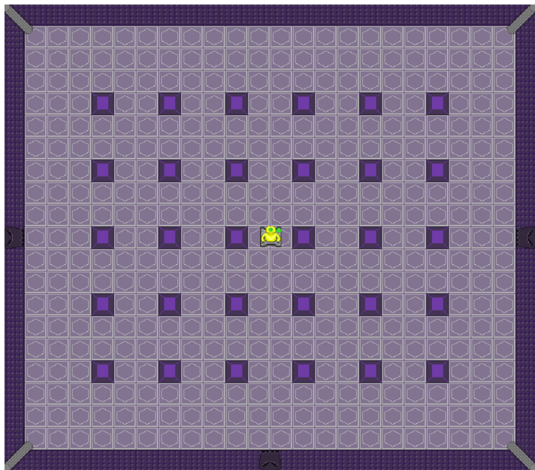
$$f(x) = \frac{100 - e(x)}{n} \tag{4.34}$$

A quest line with a single quest, or even no quest at all, is less interesting for the player than a quest line with multiple quests. Similarly, a quest line with too many quests might be exhausting for the player. This way, we determined the probabilities by functions to vary the quest amount according to the current number of quests.

### 4.3.7 Cellular Automata Room Generator

For both the enemy generator experiment and the offline setup (presented, respectively, in Sections 5.1 and 5.2), we select the rooms for the dungeon by randomly drawing, with repositioning, from a set of previously generated rooms. Figure 36a shows an example of a room created with the checkerboard pattern, and Figure 36b a room with the vertical lines pattern.

(a) Room created with a checkerboard template.



(b) Room created with a vertical lines template.

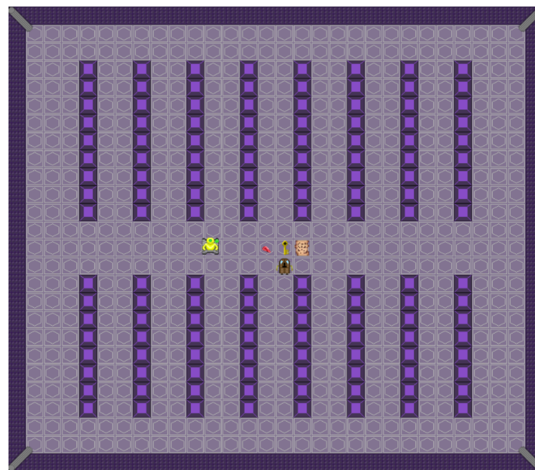


Figure 36 – Example of rooms generated with fixed patterns.

Simple rules generate these patterns so that we can support any room size. The rooms have at most four doors, one in each cardinal direction (North, South, East, West), as described in both versions of the dungeon generator algorithm, in Sections 4.3.1 and 4.3.2. We also fix the doors' positions in the room: they must always be in the center of their respective walls.

However, for the online setup (Section 5.3), we added a generator based on a Cellular Automata approach, increasing variety. Algorithm 8 presents this approach, where $chanceToCreateBlockTile$ is 20%, $generations$ is 5, and the Cellular Automata follows the rule B345/S2345 in string notation. This rule means that a block tile is created if it has 3 to 5 blocks as neighbors, and it stays alive only if it has 2 to 5 blocks as neighbors, considering the cell's 8-neighborhood.

**Algorithm 8** – Room-generating algorithm.

```
Place the rooms' doors in the center of theirs corresponding walls
for all tile in tilesInRoom do
    if chanceToCreateBlockTile < randomPercentage() then
        tile ← blockTile
    else
        tile ← floorTile
    end if
end for
for all tile in tilesInRoom do
    Erode the room with the cross-shaped structuring element Matrix from Figure 37
end for
for i ← 0, generations do
    for all tile in tilesInRoom do
        ApplyCellularAutomataRule()
    end for
end for
for all door in doors do
    for all otherDoor in doors do
        path ← FloodFillTraversal(door, otherDoor)
        if ( then!CanReachDestination(path))
            tileInPath ← path.getRandomTile()
            RemoveBlocksFromShortestPath(tileInPath, otherDoor)
        end if
    end for
end for
for all tile in tilesInRoom do
    Erode the room with the cross-shaped structuring element Matrix from Figure 37
end for
```

We use the cross-shaped structuring element shown in Figure 37 for erosion. That is, only blocks containing a complete 4-neighborhood are maintained. The erosion helps delete single blocks that may feel visual. Figure 38 shows some rooms created with the Cellular Automata-based algorithm.

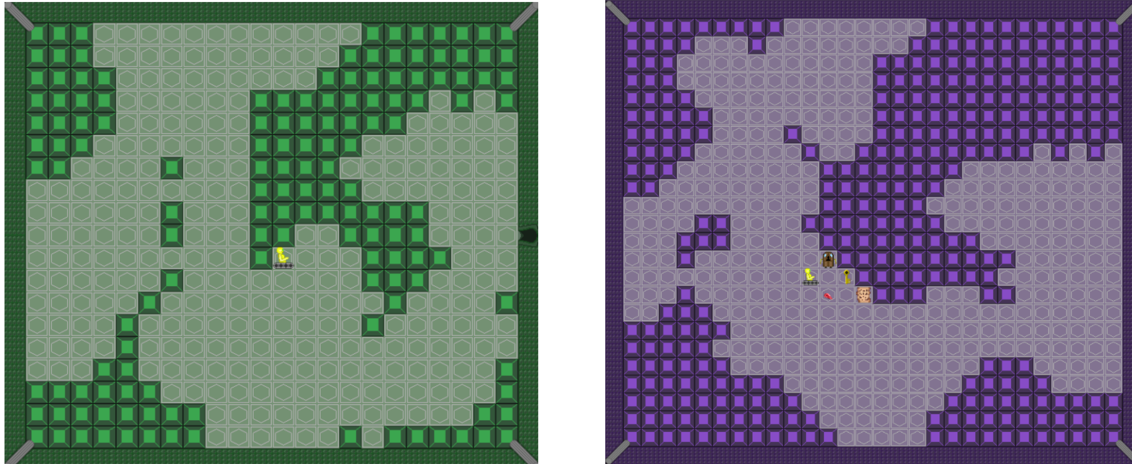
Figure 37 – Cross-shaped structuring element matrix applied for the erosion.

$$\begin{bmatrix} & 1 & \\ 1 & 1 & 1 \\ & 1 & \end{bmatrix}$$

### 4.3.8 Content Orchestrator

The Content Orchestrator component is mainly a post-processing unit of the generated contents, which follows the steps in Algorithm 9. The proposed orchestrator alters contents

Figure 38 – Example of rooms generated with the Cellular Automata-based algorithm.



following a predetermined set of rules created after design decisions and preliminary tests conducted with a smaller sample of players. For the levels' contents, the Content Orchestrator discards those with fitness values too distant from the desired one based on the levels returned by the MAP-Elites. Content Orchestrator first removes some undesirable combinations of enemies, as the MAP-Elites algorithm does not have a fix function to remove them, 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, the 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 places all needed enemies of that type.

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. Algorithm 9 presents pseudocode summarizing the orchestrator's operations.

### 4.3.9 PCG Workflow Example

To summarize and enhance the understanding of the workflow of the PCG module of the Overlord's system, we present an example of how a series of weights extracted from a player's profile will be processed and passed along each algorithm, and what content will be generated. Figure 39 shows a part of the diagram shown in Figure 59 focusing on the PCG part of the system. The generation starts by receiving the weights of the player's preferences on each profile, through the orange *portal*, and create the quests. Then, the quests are used to generate levels and enemies. Both are used to feed the orchestrator, which arranges enemies inside the dungeon. The dungeons are also fed to the room generator system, which places the tiles in each room. The resulting content is sent to the Game System (the red *portal*).

Let's exemplify the generation with a player who has an *achievement* weight of 0.8,

**Algorithm 9** – Content Orchestration algorithm.

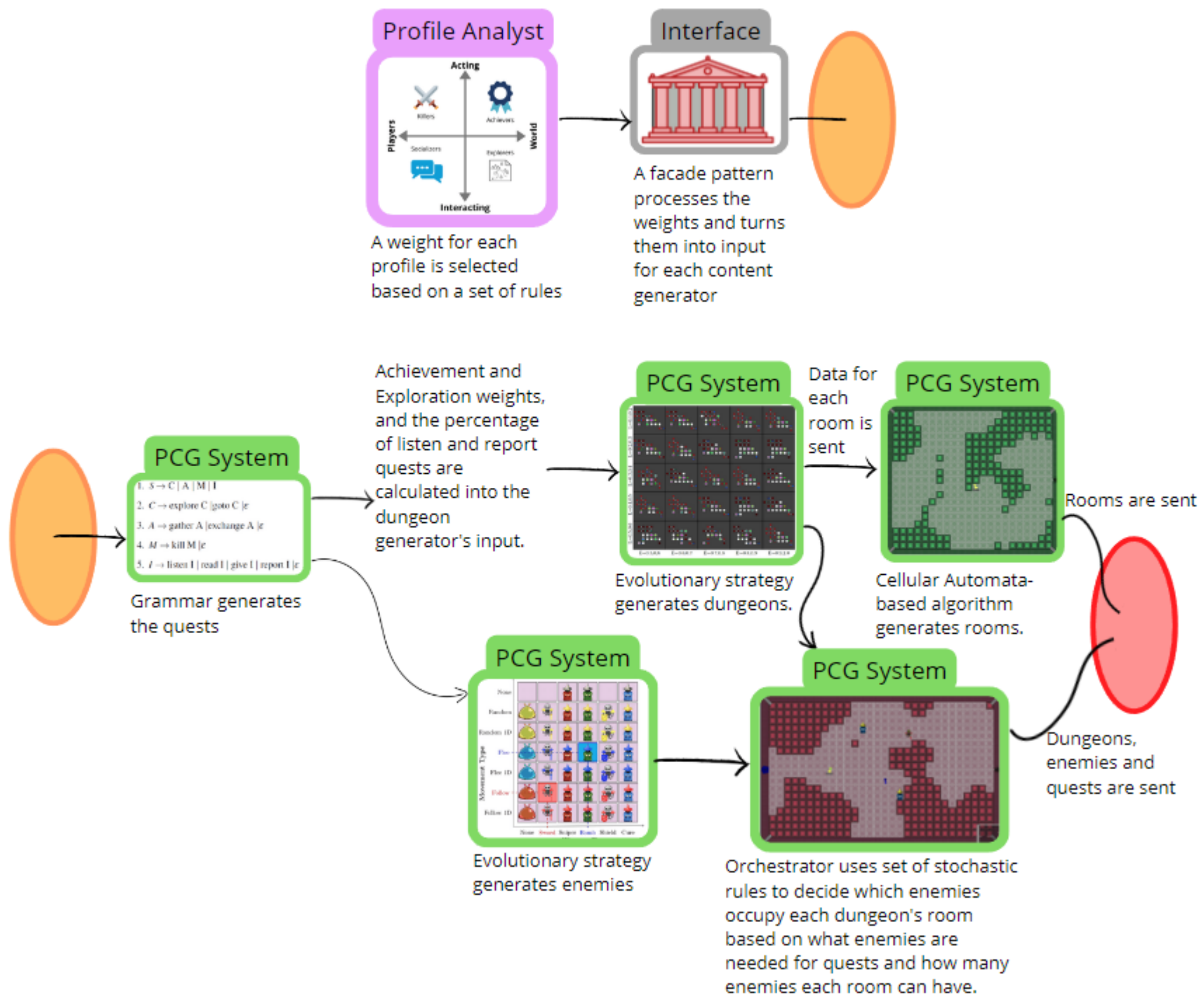
```
for all enemy in generatedEnemies do
    if movement = none AND (weapon = none OR weapon = sword OR weapon = shield) then
        generatedEnemies.Remove(enemy)
    end if
end for
for all dungeon in genereatedDungeons do
    if dungeon.Fitness > threshold then
        genereatedDungeons.Remove(dungeon)
    end if
end for
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())
            end if
        else
            if Union(melees, ranged).Count() > 0 then
                room.Enemies.Add(Union(melees, ranged).Pop())
            else
                room.Enemies.Add(healer.Pop())
            end if
        end ifroom.NeededEnemies − −
    end while
end for
```

*creativity* and *immersion* weights of 0.6, and a *mastery* weight of 0.4. The input given to the PCG System by the Profile Analyst System would be an array with these 4 weights: [0.8, 0.6, 0.6, 0.4], following the alphabetic order of profiles. We call this array the $playerProfile$. The quest generator would then create quests using the algorithm provided to it. In our case, it could be both the formal grammar or the stochastic grammar.

To make the example easier, our hypothetical quest generator will generate only two questlines for this input, and add them to a list of questlines, which we will call $questLines$. Each questline has an array of quests, which consist of the quest type (the terminal of the grammar) and additional information about it. For example: kill quests need the name of the enemy to kill and how many; exploration quests the percentage of the dungeon to explore; etc. The quests below are for the first questline in $questLines$, $questLines[0]$:

- (type:) Kill Quest, (enemy to kill:) Slimes, (amount:) 5;
- (type:) Listen Quest, (NPC to listen to:) Alasdoor;
- (type:) Gather Quest, (item to gather:) Diamond, (amount:) 4;

Figure 39 – Workflow of the PCG system, a part of the diagram shown in Figure 59. After receiving the weight of the player's preference for each profile, the generation starts and proceeds to generate all contents, send to the Game System at the end.




Therefore, we must kill 5 slimes. Then, we must listen to what the NPC Alasdoor has to say, and, to finish the questline, we must collect 4 diamonds. The quests below are for the second questline in $questLines$, $questLines[1]$:

- (type:) Kill Quest, (enemy to kill:) Bomber Mage, (amount:) 4;
- (type:) Give Quest, (NPC to give:) Alasdoor; (item to give:) Sword;
- (type:) Explore Quest, (percentage to explore:) 50%;

Thus, we must kill 4 bomber mages. Then, we must give a sword to the NPC Alasdoor, and, to finish the questline, we must explore at least 50% of the dungeon. The $questLines$ and the $playerProfile$ are used as input for the Dungeon Generator algorithm, according to Equations 4.35 to 4.39. $|Quests|$ is the total number of quests, $|ListenQuests|$ and $|ReportQuests|$ are the total number of Listen and Report type quests, respectively. Both quests are subtracted to the

number of quests which demand space as they only need the NPC to interact with to exist, and said NPC will occupy a single room independent of how many quests need it.

$$QuestsDemandingSpace = (|Quests| - |ListenQuests| - |ReportQuests|) \tag{4.35}$$

Equation 4.36 calculates the dungeon size (*DungeonSize*) by multiplying the number of quests which demand space and multiplying by 1 added by the player's creativity weight and adding to the minimum dungeon size (12, in our experiments). With this, the minimum number of dungeons is guaranteed, extra rooms are added to guarantee the dungeon can hold all items, NPCs and enemies, and the creativity weight may further expand dungeons to have more space for the player to travel.

$$DungeonSize = QuestsDemandingSpace * (1 + playerProfile.Creativity) + MinDungeonSize \tag{4.36}$$

The linearity is calculated by Equation 4.37, which uses twice the creativity weight and the achievement to reach a normalized value between 0 and 1, where 0 means a fully linear dungeon, and 1 a dungeon where every room is branched, as long as the rooms do not overlap.

$$Linearity = (2 * playerProfile.Creativity + playerProfile.Achivement)/3 \tag{4.37}$$

The number of keys (and locks, which are the same for our experiments) is defined in Equation 4.38. A minimum of 3 keys is guaranteed, and up to 5 more are added, if the player has a creativity weight of 1. Thus, the keys range from 3 to 8. However, Equation 4.39 adds a conditional clause that may reduce the number of keys for medium-sized dungeons or smaller: if there are less than 20 rooms, the number of keys is capped in 5. This is necessary because trying to add too many keys in smaller dungeons both result in bad game design and difficulty to converge the EA. For example, if we had 8 keys, we would also need 8 locked rooms, for a total of 16 special rooms in the dungeon. If this dungeon had less than 17 rooms, that would be impossible.

$$Keys = (playerProfile.Creativity/0.2) + 3 \tag{4.38}$$

$$DungeonSize/20 = 0 \rightarrow Keys = Min(Keys, 5) \tag{4.39}$$

The generated dungeons are passed as input for the room generator. This algorithm does not need any input besides the number of rooms it must create, and the location of the doors for

each room. Therefore, no processing is needed. The output of the room generator is the dungeon, with all its rooms tiled. The dungeon is also fed to the orchestrator, together with the enemies.

Said enemies are generated using only the $playerProfile$ as input. It uses the mastery weight to define the difficulty of the enemies, which is the input of the enemy generator algorithms. These weights are calculated according to Algorithm 10.

**Algorithm 10** – Conditions to transform the player's mastery weight to a numeric difficulty value for the enemy generator algorithm.

**if** $playerProfile.Mastery < 0.15$ **then**
  $Difficulty \leftarrow 11$
**else if** $playerProfile.Mastery < 0.35$ **then**
  $Difficulty \leftarrow 13$
**else if** $playerProfile.Mastery < 0.65$ **then**
  $Difficulty \leftarrow 15$
**else if** $playerProfile.Mastery < 0.85$ **then**
  $Difficulty \leftarrow 17$
**else**
  $Difficulty \leftarrow 19$
**end if**

Finally, the orchestrator algorithm will receive the generated enemies and dungeon, and the $questLines$. First, it removes enemies with incompatible combinations, like melee enemies (slimes) that do not move. Then, it will select random enemies from the generated ones which has the same type as the required by the quests. That is, it will randomly select between those with the desired weapon type, until the sum of all enemies to kill of said type is fulfilled. For brevity, we suppose a dungeon with only 3 rooms was generated by the dungeon generator, presented below with their x and y coordinates and total enemies inside:

- Room[0]: x=0, y=0, enemies=3
- Room[1]: x=1, y=0, enemies=2
- Room[1]: x=0, y=1, enemies=4

Figure 40 shows what the dungeon would look like after the distribution of enemies. The NPC and items are distributed randomly throughout the dungeon in a last post-processing step. This complete dungeon is sent to the Game System to be played.

## 4.4 Game Prototype

In this section, we present in detail the Game System from the Overlord Context Diagram (Figure 16), highlighted in Figure 41. The game prototype's role is to test the generated content and collect user data, feeding the Player Profile model.

Figure 40 – A possible solution of the orchestrator on how to divide the enemies, items and NPCs through the dungeon's rooms

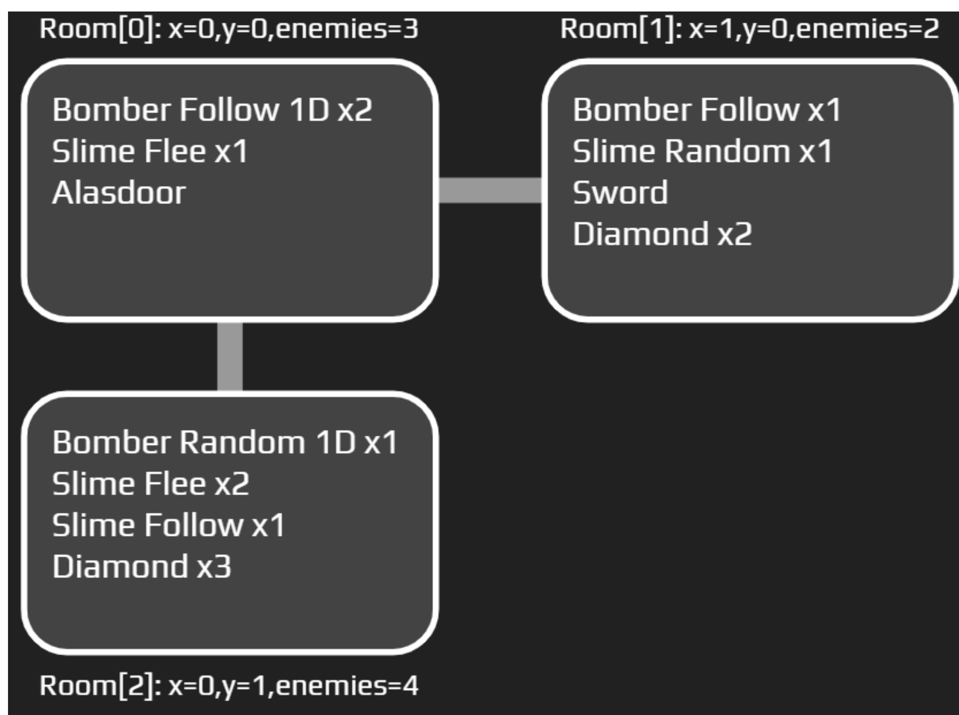



Figure 41 – Highlight of the Game System inside the Overlord Context Diagram

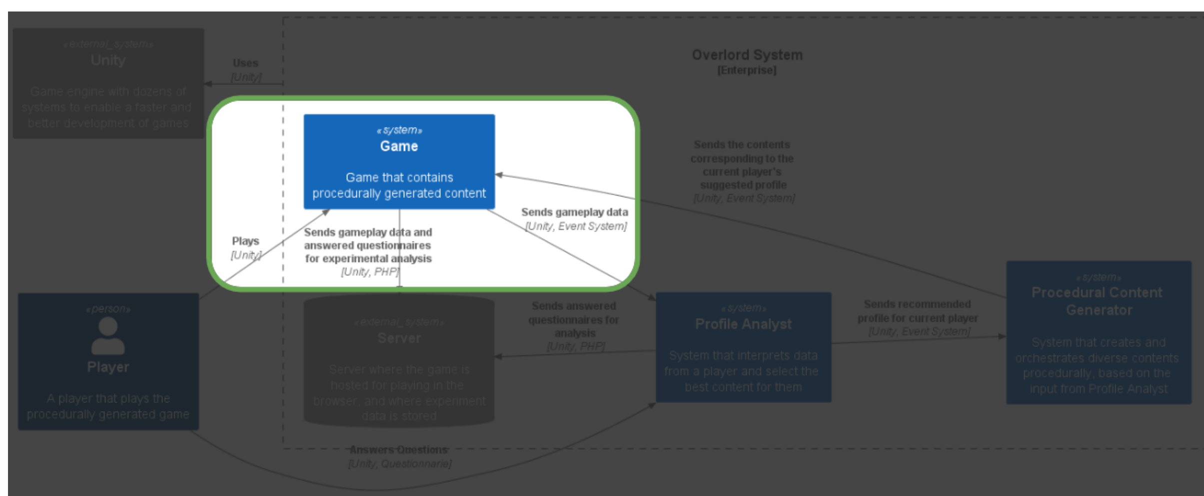


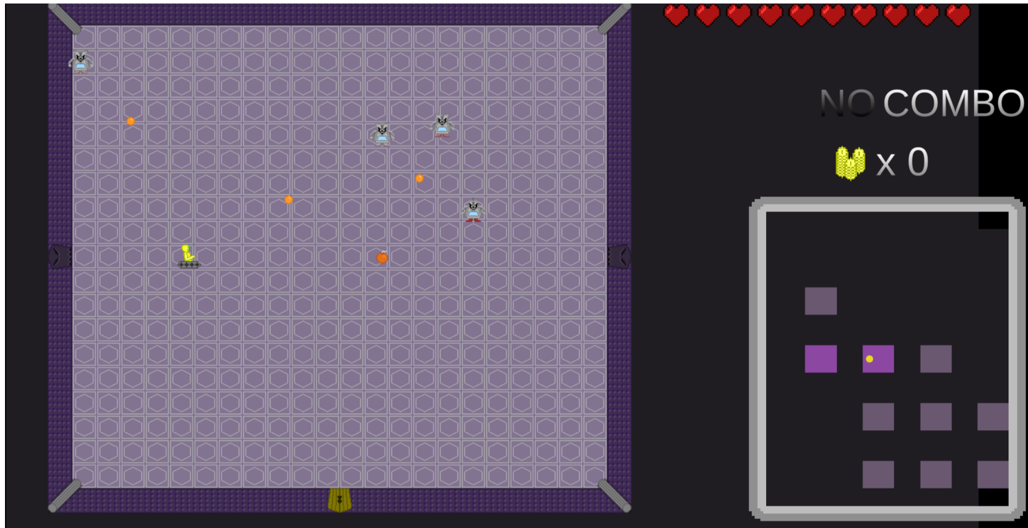

Figure 16 shows the game prototype module of the Overlord system. Such a game is an action-adventure that mimics the main mechanics from *The Binding of Isaac* game's combat and the dungeon exploration of *The Legend of Zelda* game. The player controls a yellow robot that must move around the many rooms of a dungeon to collect keys, open locked doors and find the special item (a green Sierpinski triangle) at the end of the dungeon. The player may collect items to trade with NPCs, add to their inventory, and fight enemies. The NPCs will give the players quests to complete according to what the Quest Generator component demands.

Each room, except the starting room and the one with the special item, may be filled with enemies. Figure 42a has a screenshot of the game in the earliest version, used for the Enemy

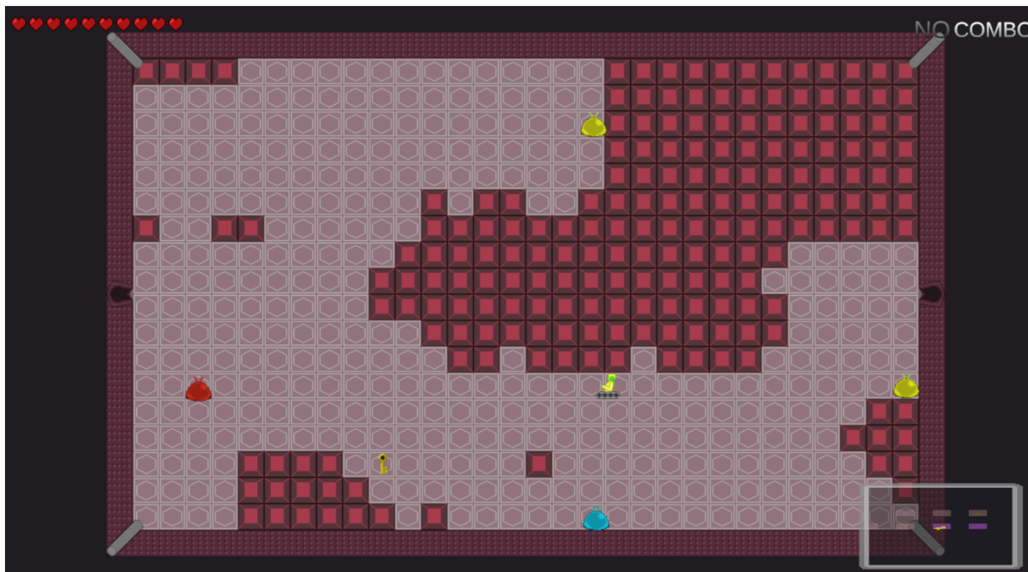
Generator Experiment (described in Section 5.1) with the player avatar (yellow), the enemies (gray) and the enemies' projectiles (orange). All doors in the room are locked (as we did in the enemy generator and offline setups). At the bottom right, there is a mini-map showing the current and neighboring rooms. At the top-right is the player's health, always starting at ten health points. Each enemy's damage results in losing a certain number of hearts (or health points). Figure 42b has a screenshot of the most recent version of the game, with the slime enemies, a yellow key, the Cellular Automata-generated room, and open doors, as in the online experimental setup (Section 5.3) we did not close the doors, allowing for players to play the game as they preferred.

Figure 42 – Screenshots showing differences from the early version of the game from the most current version.

(a) A screenshot from the game in its early version, used in the experiments in Section 5.1.




(b) A screenshot from the game in its most recent version, used in the experiments in Section 5.3.

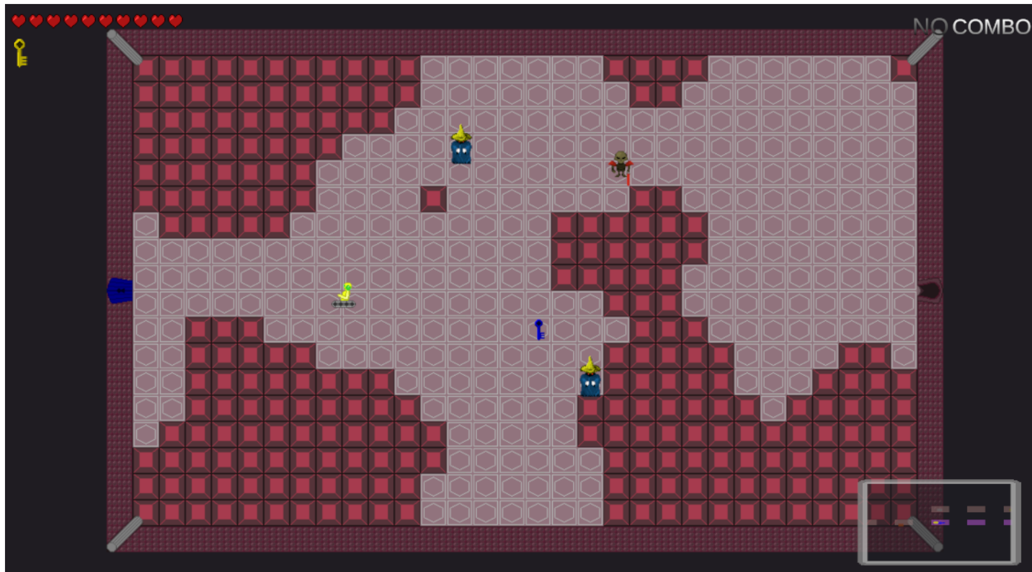



Some doors are locked and need a key of the same color to open them. The player must search for and collect the keys, as seen in Figure 43a, showing a blue key and a blue locked door opened by it. Figure 43b shows some items on the floor (a garnet and an emerald), which the

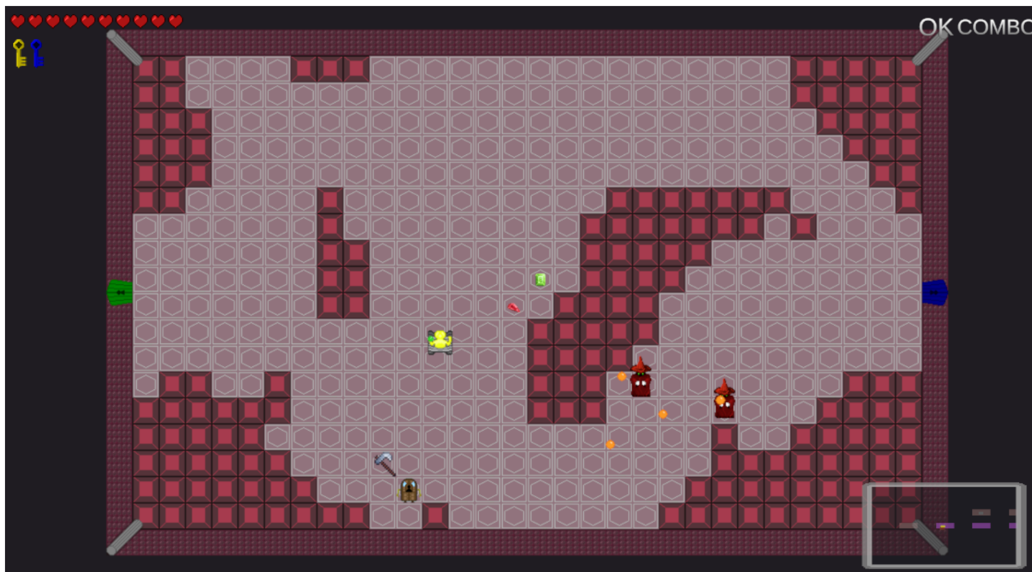
player collects by colliding with them. It also indicates Alasdoor, our door-like NPC, who the player may interact with by pressing the *X* button when in contact with it. And some red mage enemies of the following movement type variant, as their red hats indicate.

Figure 43 – Screenshots showing some major gameplay elements.

(a) A blue key that must be collected to open a blue lock, both in the same room.




(b) Items scattered in the floor (garnet and emerald), as well as Alasdoor, our door-like NPC, and some enemies.




The following subsections will describe the core gameplay and the main mechanics of the game: the enemies, the quest system, the dialogue system, the content placement system, the items and inventory, and the user interface elements. We also show how we collect gameplay data to provide to the Profile Analyst component, described in Section 4.2.1.

### 4.4.1 Gameplay

The game's core mechanics consist of moving around the dungeon, shooting enemies, collecting items, solving the locked-door puzzles, and talking to NPCs. The player can do these actions by using the keys mapped in Figure 44.

Figure 44 – Mapping of each game's action to their corresponding key in the keyboard.

The player can move around in the dungeon in 8 directions, as the diagonal movement occurs when they simultaneously press two adjacent movement keys (the *WASD* ones). The player will collide against blocks, walls, closed doors, NPCs, and enemies. The arrows allow the player to shoot bullets in the directions the arrow points. No diagonal shots are permitted, but they use the player's movement momentum, being possible trick shots in this way.

The items are collected automatically by moving inside their colliders. When the player collides with doors, they automatically open as long as the player has the corresponding key. When touching an unlocked door, the player goes to the adjacent room, with a minimap guiding the player through the dungeon. This minimap becomes full screen by pressing the Tab key.

Figure 45 shows the full scream map. The not explored rooms appear in green rectangles, and the rooms visited appear in pink. The minimap does not show corridors, as the game was designed for players to memorize the path as part of the challenge for the locked-doors puzzle.

When pressing Q on the keyboard, the player opens the quest menu, a straightforward UI showing both completed and ongoing quests. Figure 46 has the menu. These quests open when the player talks to an NPC by pressing the keyboard's X key. Figure 47 shows an NPC introducing a quest to the player. This quest is a quest to kill enemies.

The enemies in the game are simple agents controlled by AI. They have difficulty ratings to increase or decrease their stats, defined by the Enemy Generator component, which also defines which movement pattern and the weapon they will use. At this point, we did an implementation similar to the Strategy design pattern. The game reads the serialized data of the enemy and

Figure 45 – Map of a dungeon as shown in the game's UI when in the full-screen map mode.

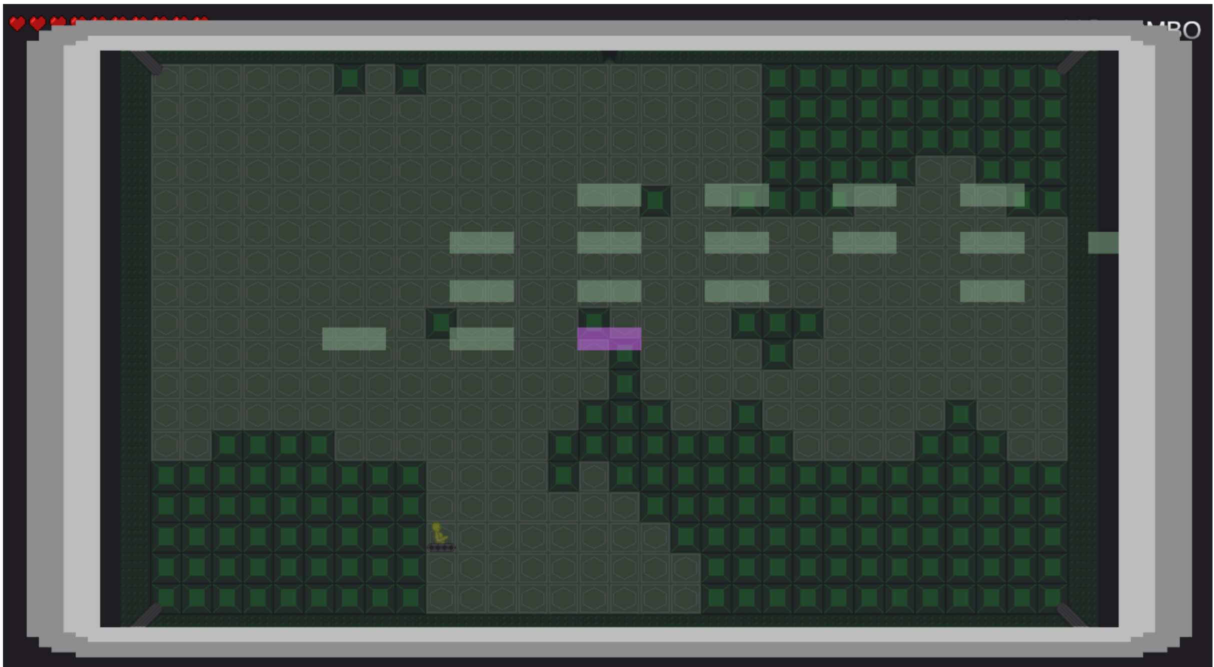


Figure 46 – Menu that shows the completed and currently open quests for the player.

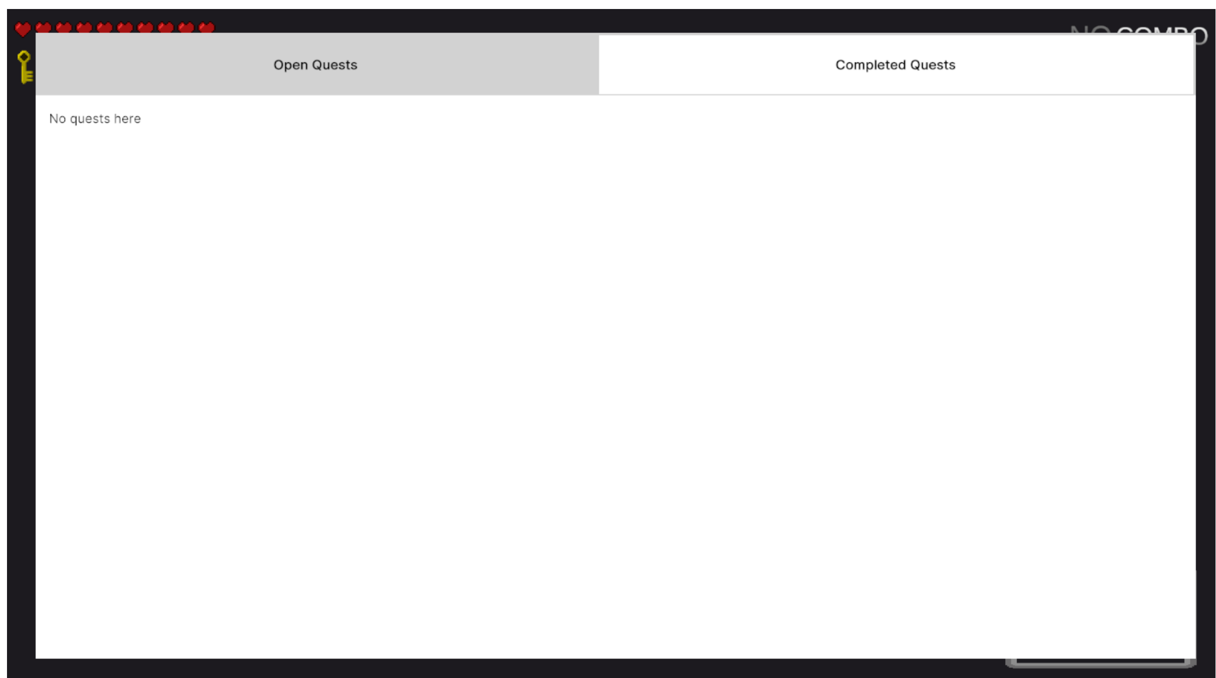
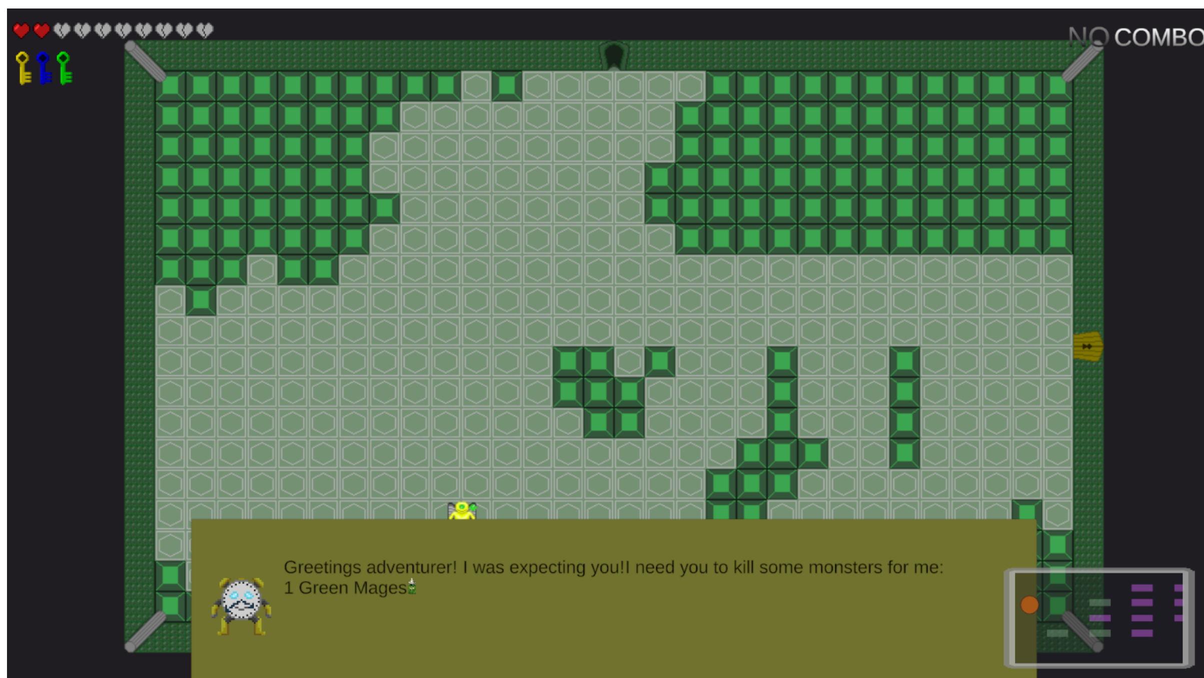


Figure 47 – Dialogue UI, where an NPC is requesting the player to kill a certain number of enemies from a specific type.




selects which *prefab* will use to instantiate the enemy (containing their sprite, animators, scripts, etc.). Next, it delegates the corresponding movement method and weapon attack script to the enemies while instantiating them.

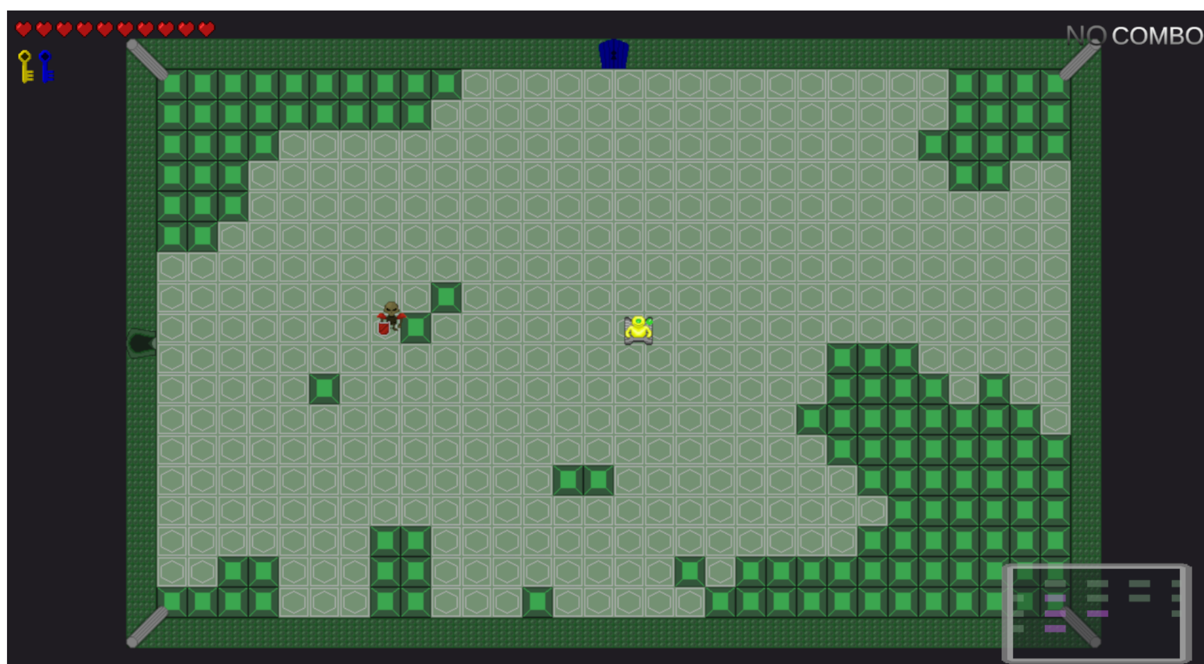
Enemies can damage the player character through contact or using projectiles, and the damage depends on the enemies' attack points. When an enemy dies, it will not respawn. However, when the player leaves the room, the game respawns them when the player comes back to the room. Figure 48 shows a red skeleton (sword-type enemy with a movement pattern to follow the player) in the same room as the player.

The player character has ten life points, which are only refilled at the start of each level, and will end the game if all of them are lost. They appear in the upper-left corner in Figure 48. Enemies also have lives just like the player character and will be removed from the game if all of them are lost.

The player may select one of four weapons before starting a level. One has increased damage but reduced fire rate and speed, and the other a medium damage, speed, and fire rate. The third one has lower damage, medium speed, and increased fire rate, while the last one fires three medium-damaging bullets in an arc but with a reduced fire rate and medium speed.

Each shot hitting an enemy increases a Combo counter, and it is reset to zero whenever the player character loses one of their health points. This successful shot adds a reward for players who like combat, capable of defeating multiple enemies without being hit. Figure 48 shows the combo counter with no hits: "*No Combo*". The words are updated with better combos when their thresholds are reached.

Figure 48 – A red skeleton, an enemy, in a room with the player.




### 4.4.2 Enemies

The enemies have the properties provided by the generator algorithm (health points, damage, movement speed, active time, rest time, weapon type, and movement type), as well as the scripts that handle their movement, attack, and overall behavior patterns. They also have their sprites and animations, their sound effects for when they are hit and killed, and a collider.

The health points handle their health: when it reaches 0, they are deleted after playing their death animation and a death sound effect. The damage is the number of health points subtracted from the player's health when colliding with the enemy or colliding with the enemy's projectiles. Movement speed is the amount of space in-game units move per frame. The active time is when they keep moving before the next pause, which lasts for the number of seconds stipulated by rest time.

The weapon type decides which specific instance of the enemy controller's inherited classes the enemy will be created with, as well as their *prefab* within Unity, which will, in turn, define the enemy's visuals. The possible weapons and their corresponding enemy representation are presented next:

***No weapons*** A slime. Just moves around and gives damage on contact.

***Sword*** A robot with a sword in the offline setup and a skeleton with a sword in the online one. Gives damage on contact, with a slightly extended reach thanks to the sword.

***Shield*** A robot with a shield in the offline setup and a skeleton with a shield in the online one. Gives damage on contact. The shield has a collider that blocks the player's bullets,

destroying them on contact.

***Projectile*** A mage in a red robe. Gives damage on contact, but also throws a projectile aimed at the player's current position. The projectile's speed is given by the enemy generator's resulting values, as well as the fire rate.

***Bomb*** A mage in a green robe. Gives damage on contact, but also throws a bomb at the player's current position. The bomb's speed is given by the enemy generator's resulting values and the fire rate. The bomb waits a fixed number of seconds before exploding, damaging the player if they are within the explosion radius (a circle collider spawning from the bomb's center, with its corresponding sprite indicating said area).

***Heal*** A mage in a blue robe. Gives damage on contact. The cure magic fire rate is determined by the enemy generator's resulting value, and the healing amount is the enemy's damage. The healing occurs in a wide circle around the caster, which is a sprite signaling the collider is where all enemies residing within (except the caster) are healed for the given amount.

For the movement, the game reads the desired movement patterns and adds the corresponding delegate (function pointer) to the movement function inside the enemy's movement script. In the online experimental setup, we painted some props in each enemy's sprites to be colored according to the movement pattern. This helps to give the player feedback on what pattern each enemy has. For the mages, we paint their hats. For the skeletons, we paint their spaulders and swords or shields. For the slimes, it is their whole color. Figure 32 showed their colors according to the movement type. We describe below the possible movement types:

***None*** The enemy does not move. For the online experimental setup, paint the enemy's props in gray.

***Random*** At the beginning of each movement cycle, select a random 2D vector pointing to the direction that the enemy will move until the end of their active time. For the online experimental setup, they have yellow props.

***Random 1D*** At the beginning of each movement cycle, select a random 1D vector pointing to the direction that the enemy will move until the end of their active time. For the online experimental setup, they have yellow props.

***Flee*** At each active frame, moves to the direction of a 2D vector pointing in the opposite direction of the player. For the online experimental setup, they have blue props.

***Flee 1D*** At each active frame, moves to the direction of a 1D vector pointing in the opposite direction of the player. The axis (x or y) is defined randomly when the enemy is spawned and remains the same until they are destroyed. For the online experimental setup, they have blue props.

***Follow*** At each active frame, move to the direction of a 2D vector pointing to the player. For the online experimental setup, they have red props.

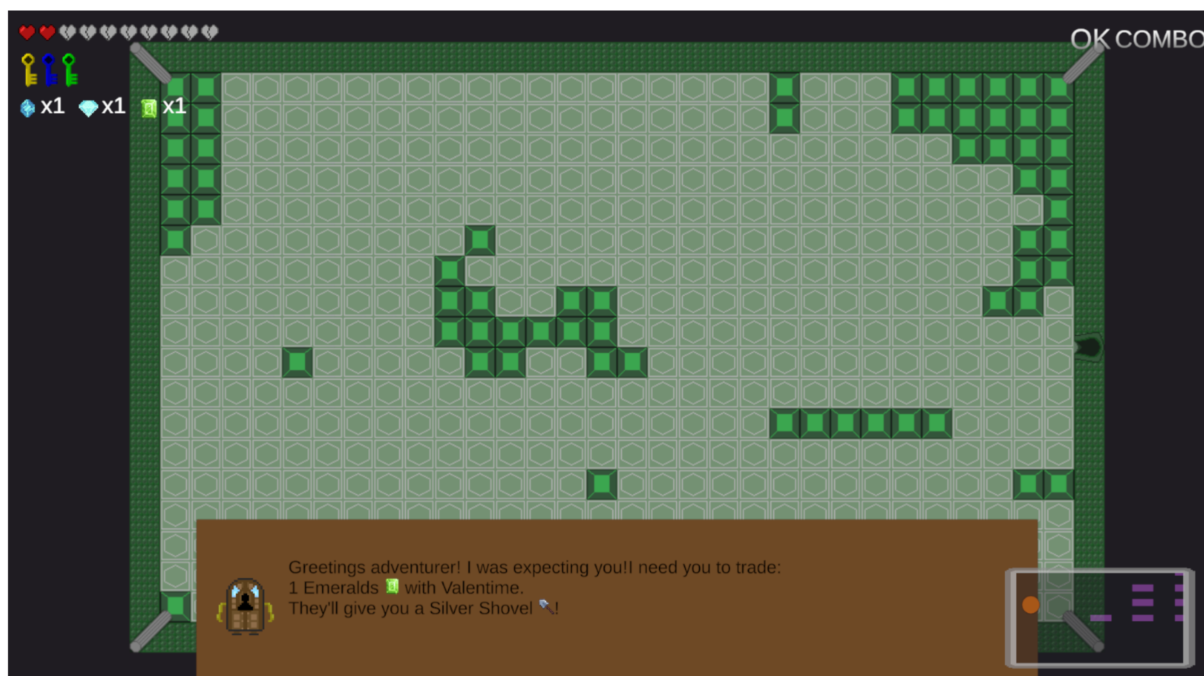
***Follow 1D*** At each active frame, move to the direction of a 1D vector pointing to the player. The axis (x or y) is defined randomly when the enemy is spawned and remains the same until they are destroyed. For the online experimental setup, they have red props.

### 4.4.3 Quest System

Our game has a quest system to handle the quests generated by their respective generator (Sections 4.3.5 and 4.3.6). When starting the game, each questline is processed, and the first quest opens, which means the quest controller broadcasts an event with the quest data and the NPC in charge of that quest. The NPCs listen to the event and, when they are the ones in charge, add a said quest to their quest queue.

When receiving a quest, the NPC creates an opener dialogue, getting the quest's data and transforming it into an adequate string. The NPC adds such a string to its list of dialogue lines to speak when interacting. The Dialogue System handles the interaction of the NPC with a player, as described in Section 4.4.4. The dialogue is shown in Figure 49. This opener dialogue repeats as often as the player talks to the NPC. When the quest opens, the broadcast data is also received by the Quest UI component, adding it to the list of open quests.

Figure 49 – An NPC is requesting the player to collect some items and trade for another, with a fellow NPC.



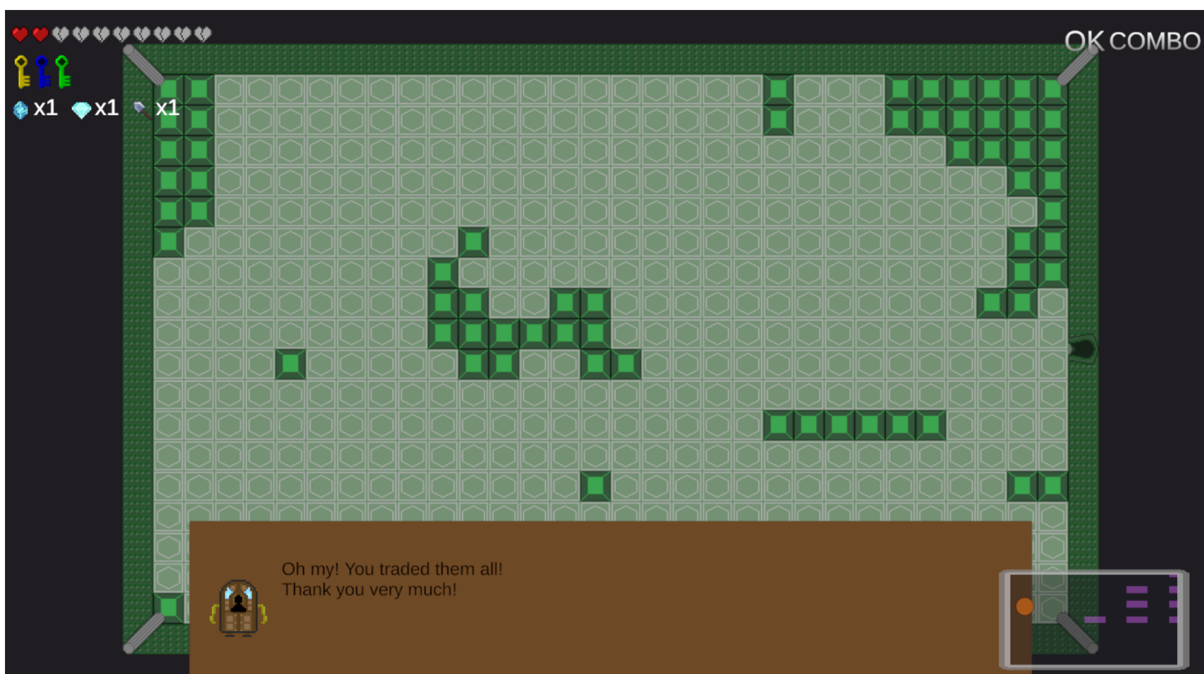

Each game object related to a quest needs to implement the *IQuestElement* interface, which has a method that calls an event when the game object's quest-related task is completed, and need to save their quest's ID, so it can be later identified. This event is broadcast to the quest

system, passing along the quest ID and any relevant data (e.g., the type of enemy killed, the name of NPC spoken with, etc.). The Quest System uses polymorphism to know the type of the quest and uses the ID to find it in the questline that contains it. Then the questline removes the task from the quest, and if it did all tasks, the questline sends another event, now broadcasting that quest is completed.

When a quest is completed, the NPCs listen to the event broadcast by the questline. If it is one of their quests, they remove the opener dialogue from such quest and add a closer dialogue, thanking the player for doing the task, and building the dialogue according to the quest's type.

When the player interacts with the NPC, and they speak the closer line, another event is broadcast from the Dialogue Controller and listened to by the Quest Controller. The controller will find the closed quest in its questlines, and the latter will close the quest and open the next one if it exists. Figure 50 shows the exchange quest closing dialogue.

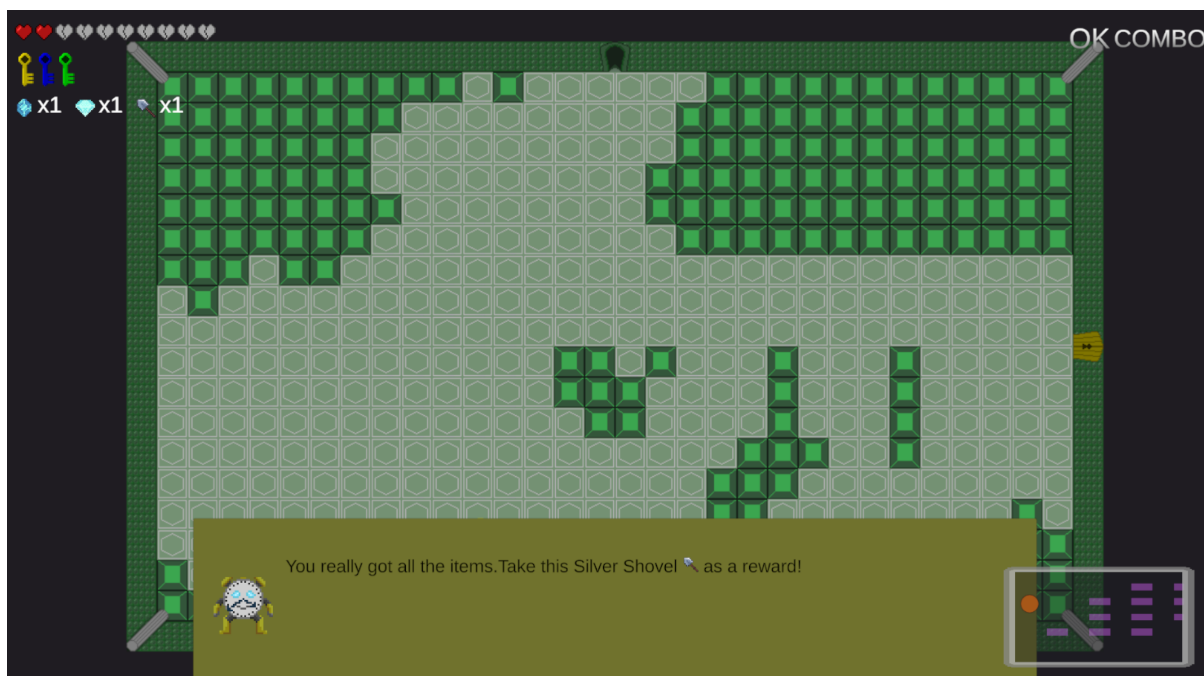
Figure 50 – An NPC is congratulating the player for completing an exchange quest. When the dialogue appears on the screen, an event is broadcast and the quest controller, which is listening, closes the quest.




Some specific quests need an intermediate dialogue before they are complete. That is the case of exchanging items, for example. In this case, when the player completes the intermediate step (in the exchange quest case, gather all items), an event is broadcast from the quest itself once identifying the conclusion of all previous tasks. The target NPC for the quest listens to it, getting the necessary data for the next step. Said NPC will then create an exchange dialogue and, the next time the player interacts with them, the Dialogue System broadcasts the quest closer. The NPC in charge of the quest listens to this event, generating a closer dialogue. Furthermore, in this case, the same event is heard by the inventory controller and the inventory UI. Moreover,

the player's inventory and the corresponding UI are updated. Figure 51 shows the exchange happening.

Figure 51 – An NPC is exchanging items with the player, after all necessary items were collected. When the dialogue appears on-screen, an event is broadcast and the quest controller, which is listening, completes the quest. The inventory controller and inventory UI also listens to the event, updating their data accordingly.




### 4.4.4 Dialogue System

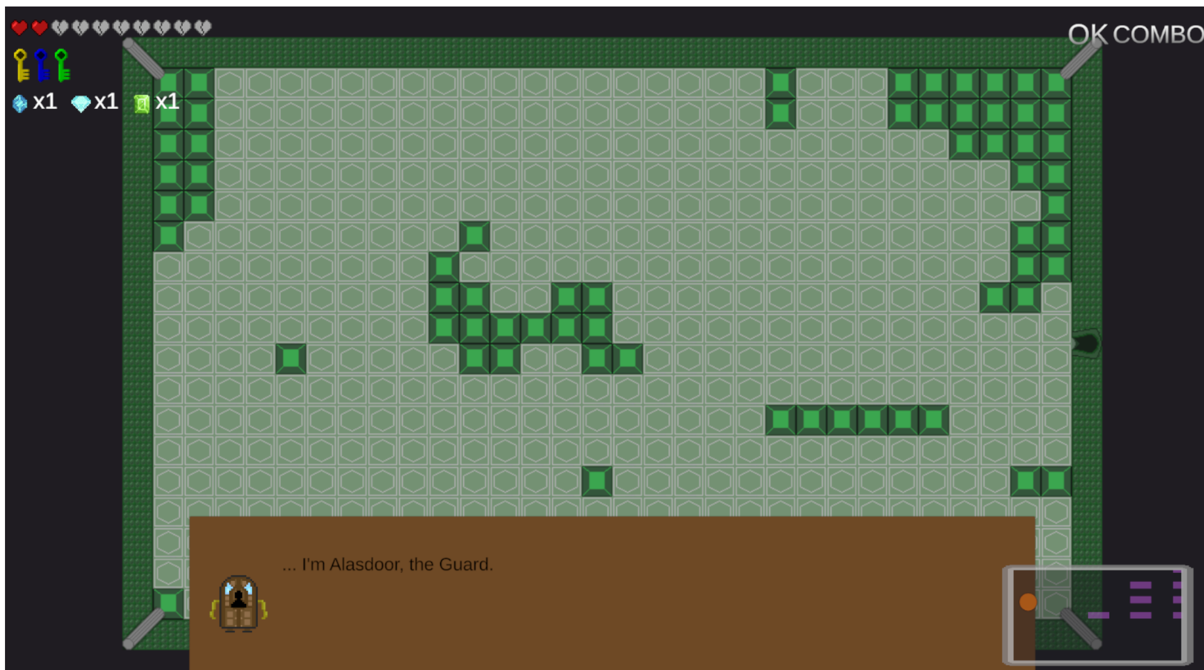
For the dialogue system, we used the Dialogue Module[9] from *FellowshifOfTheGame*'s GitHub repository as a base. It is a dialogue system adapted from two previous projects from the Fellowship of the Game extension group in our university (Universidade de São Paulo - USP), and I am a developer in one of these projects: Final Inferno[10]. Therefore, it was a system I was familiar with, hence the choice to use it.

This system saves dialogues as a list of dialogue lines belonging to a Dialogue Controller Scriptable Object. This controller must be attached to the Game Object that will speak them (in our case, the NPCs and readable items, such as books) and must have the data to show the background color and portrait (sprite) according to the speaker. Each Game Object that wants to show dialogues must have a collider and the *QuestDialogueInteraction* script as components.

Figure 52 shows an introduction dialogue from our NPC called Alasdoor. This dialogue starts when the player is within the NPC's collider range and presses the "talk" button (X key). When the player presses the "confirm" button, the dialogue line is finished, and if there is another, the system shows the next. Otherwise, the dialogue ends, and the UI is closed.

9 <https://github.com/FellowshipOfTheGame/DialogueModule>
10 <https://github.com/FellowshipOfTheGame/FinalInferno>

Figure 52 – An example of a dialogue inside the game. The NPC portrait and background color varies according to the speaker. The dialogue can be divided in different lines, which are displayed one after the other, after the player press the "confirm" button (X key).




In addition, we adapted Mix and Jam's AC-Dialogue repository's code[11] to work according to our existing Dialogue System. Their code can encode events in the dialogue with custom tags. Our system adds these tags while creating the quests' openers, closers, and intermediate (e.g., exchange) dialogues. When the Dialogue System process the dialogue line for the player, while they interact with an NPC, the system removes the tags from the final dialogue and calls the events corresponding to the tags it found.

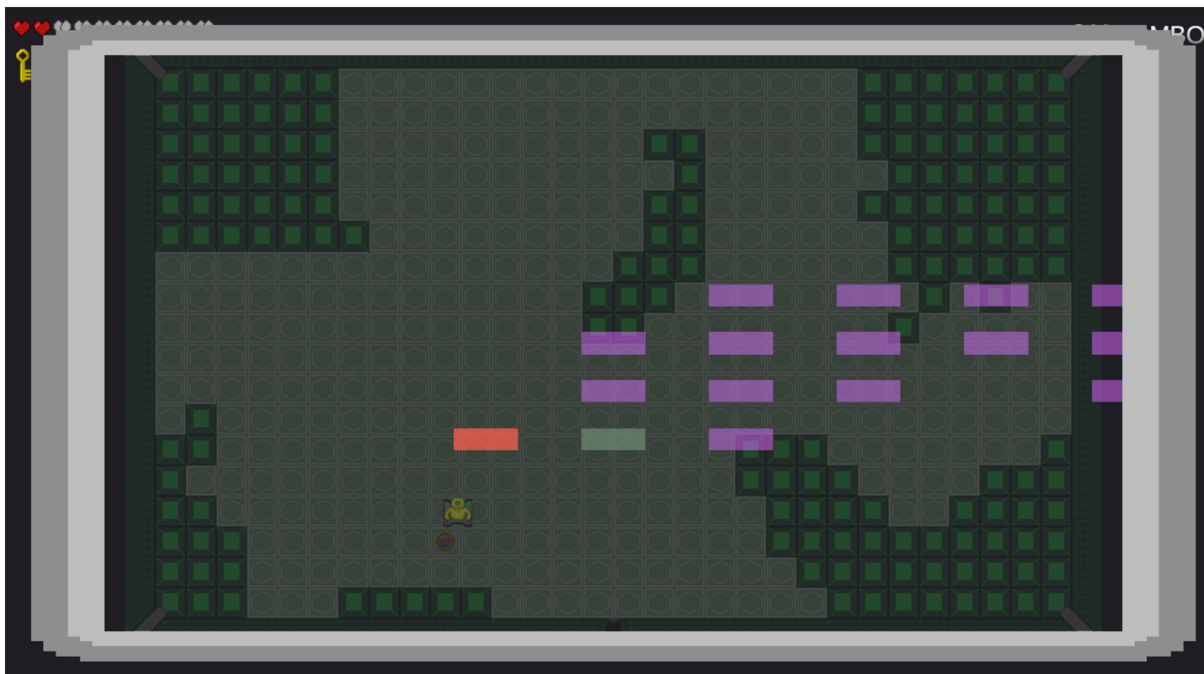
Our system has three custom tags. One to send the event when the dialogue is about the completion of a quest, another to handle the item exchange event (more details on both in Section 4.4.3), and the last one for the opening of a quest to go to a specific room. In the last one, an event is broadcast, and the minimap controller listens to it, opening the minimap in full screen. It becomes the room to be visited by the player in red, slowly moving the minimap camera to the goal room. The idea is to give the player better feedback about where the destination is. Figure 53 shows a room marked by this event.

### 4.4.5 Content Placement

Although our content generator creates a dungeon where the number of enemies in each room is given, there are no rules about where to spawn them exactly in each room, nor which rooms will have items and NPCs. Therefore, our game has a content placement algorithm to decide these.

[11] <https://github.com/mixandjam/AC-Dialogue/blob/master/Assets/TMP_Animated/Runtime/TMP_Animated.cs>

Figure 53 – A room marked in red showing where the player must go. This mark is made after an event is called by the dialogue controller when an opening dialogue for a *go somewhere* quest is spoken.

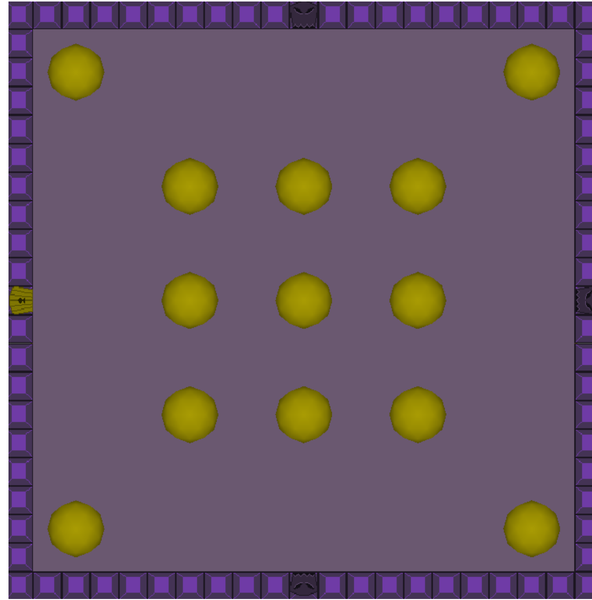


Our algorithm places treasures and NPCs in a determinist approach. For the treasures, it divides the number of treasures by the total rooms that are leaf nodes in the tree representation of the dungeon and rooms. For each room, it adds the resulting items, taking them sequentially from the list of necessary items from the quests generated by the PCG component. If there is a remainder in the division, the exceeding items are placed in the first room of the dungeon, which is not the starting room. We prioritize the leaf rooms and those behind locked doors, as they are usually found after some exploration, making the items feel more like a reward for the player's efforts.

The algorithm prioritizes the leaf nodes and rooms behind locked doors for NPC placement. The algorithm places the enemies sequentially in each of these rooms. If there are not enough rooms for the NPCs, the algorithm places them in any available room, as long as it is not the start room.

Regarding the enemy placement, our algorithm creates spawn points by dividing the room (of any size) into a fixed number of rows and columns (given as a parameter). These spawn points are removed if they are too close to a door (less than a two-tile radius) or if their resulting position collides with a block tile (which is not walkable). We delete those close to doors to avoid enemies spawning close to players, which would be unfair to them. An example of spawn points when passing to the algorithm the limit of 5 rows and columns is presented as the yellow circles in Figure 54.

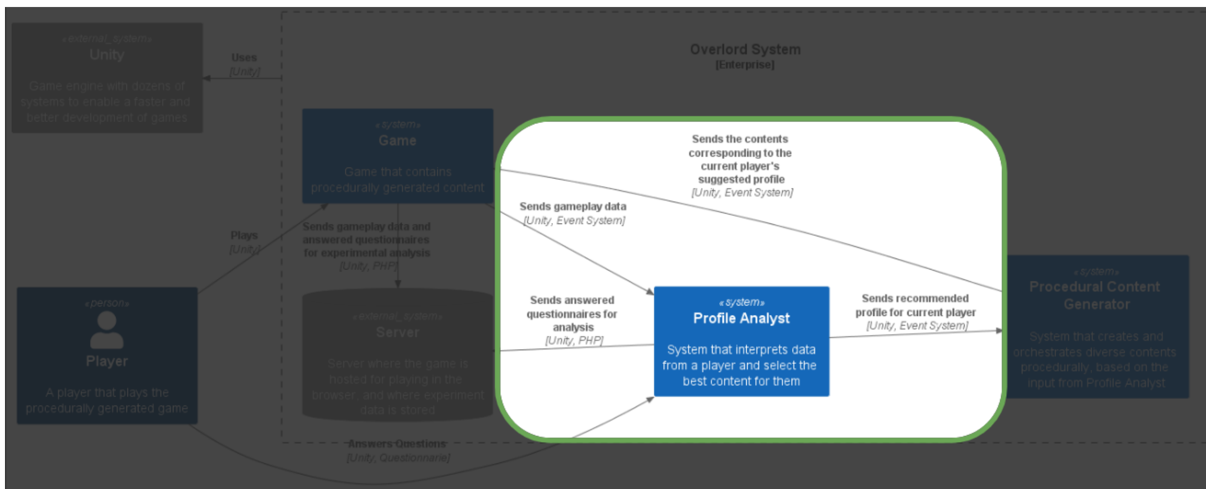
Figure 54 – Enemies' spawn points represented as yellow circles.



## 4.5 Player Profile

Now we focus on the Profile Analyst System, the area inside the Overlord Context Diagram (Figure 16) highlighted in Figure 55. The game prototype allows collecting data from the player, both explicitly and implicitly. We gathered the data implicitly when the player plays the game prototype, while two questionnaires provided explicit feedback. One questionnaire is answered before playing, named "*Pre-test*", belonging to the category of *self-assessments concerning general dimensions* in Jameson's definition. The second is answered after each level, named "*Post-test*", a *self-report on specific evaluations*, also following Jameson's categories (JAMESON, 2002).

Figure 55 – Highlight of the Profile Analyst System inside the Overlord Context Diagram




The *Questionnaire Container* has the data of the questions we asked the player inside the Unity engine and controls the logic to handle the UI and process the players' answers. Figure

56 shows a pre-test questionnaire, while Figure 57 has a post-test questionnaire.

Figure 56 – Example of a pre-test questionnaire inside the Unity engine, as the players in our experiments has seen and responded. They may click in one of the radio buttons, indicating their agreement to the statement from 1 to 5, or ignore, which will save the question as not-answered, or 0.

Before we begin the game, we would like to know a little more about you. Please, try to answer the questions below with the answers that best suit you.

I like to play games where I can collect rare items and hidden treasures. I also like to complete every possible mission, including those that are not necessary to finish the game.
1 = Strongly Disagree <-> 5 = Strongly Agree
1 2 3 4 5

I like playing games where I can explore the world and find secrets and mysteries. I also like exploring places, elements and characters from a virtual world.
1 = Strongly Disagree <-> 5 = Strongly Agree
1 2 3 4 5

I like playing games where I can immerse myself in the role of the character and make significant decisions. I like to create bounds with the game characters and work together towards a common objective.
1 = Strongly Disagree <-> 5 = Strongly Agree
1 2 3 4 5

I like playing games where I can explode, smash, destroy, shoot and kill. I also like to fight with close combat skills and dodge from quick attacks.
1 = Strongly Disagree <-> 5 = Strongly Agree
1 2 3 4 5

Figure 57 – Example of a post-test questionnaire inside the Unity engine, as the players in our experiments has seen and responded. They may click in one of the radio buttons, indicating their agreement to the statement from 1 to 5, or ignore, which will save the question as not-answered, or 0.

Congratulations! You've finished the level!
Now, we'd like your opinion about it. You just need to answer the questions below and click the "Send" button. A new level will be loaded for you to play :)

How much do you agree with the following statement: "The level was fun to play"?
1 = Strongly Disagree <-> 5 = Strongly Agree
1 2 3 4 5

How much do you agree with the following statement: "The level was difficult to complete"?
1 = Strongly Disagree <-> 5 = Strongly Agree
1 2 3 4 5

How much do you agree with the following statement: "The enemies were difficult to defeat"?
1 = Strongly Disagree <-> 5 = Strongly Agree
1 2 3 4 5

How much do you agree with the following statement: "The challenge was just right"?
1 = Strongly Disagree <-> 5 = Strongly Agree
1 2 3 4 5

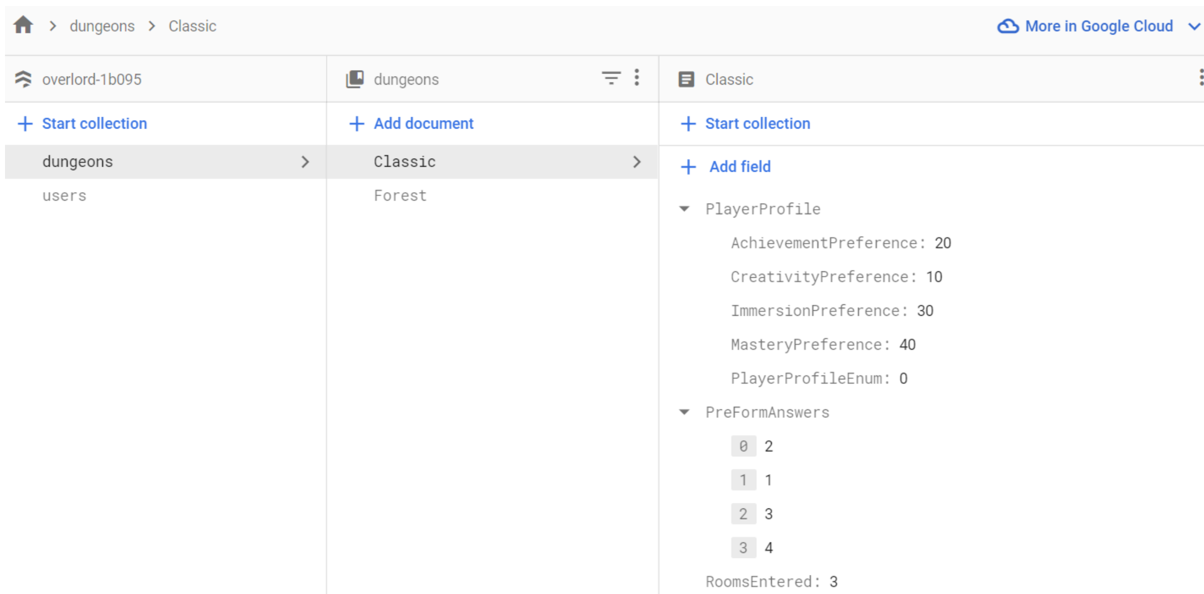
How much do you agree with the following statement: "The rewards were on the proper amount"?
1 = Strongly Disagree <-> 5 = Strongly Agree

The *Gameplay Metrics* container listens to all gameplay-related events that the game system broadcasts. These data vary a little between experiments but contain the player's main activities: how many rooms entered, how many enemies defeated, total damage taken, total items collected, total quests completed, what was the input for the generators of the dungeon, enemies, and quests in each dungeon played, the profile of the player, locks opened, and many more.

Each action of interest in the game (e.g., killing an enemy) broadcasts an event containing relevant data for this container to listen to and save. In the enemy generator and offline

experimental setups (sections 5.1 and 5.2) this data was stored in a proprietary server from our university. However, Google's Firebase[12] was set up with Unity, and the current setting saves data in a Cloud Firestore[13] database, as shown in Figure 58.

Figure 58 – Example of data collected from a player's gameplay in a dungeon named as *Classic* in the Firestore Cloud database. The picture shows the given weight the player scored for each profile, how they answered the pre-test, and the number of rooms they entered in that dungeon.




The *Profile Selector* container processes the inputs from the questionnaires or gameplay data and provides a numeric value for the current player's preferences. The processing of such data and their results vary for each experiment, so we give the implementation details for each experiment described in Chapter 5.

## 4.6 Final Remarks

With the methodology presented in this section, we propose a system that is an alternative to answer our main research question on "How can procedural content generation be effectively applied to simultaneously create multiple and coherent contents in real-time and adapting to different users' needs". This answer can be summarized as to first get user data to use as input for the PCG algorithms, which, in our case, is mostly via explicit data from questionnaires. Then, use PCG algorithms able to map different inputs, representing the player's needs, to interesting and feasible outputs, which enables us to always offer a solution. Next, we need an orchestrator, as well as an architectural pipeline, able to send the output from one generator to another and maintain the solution's feasibility while matching the player's profile. Our approach guarantees these steps by using model-based generator algorithms, where the models are created after

[12] <https://firebase.google.com/>
[13] <https://firebase.google.com/docs/firestore>

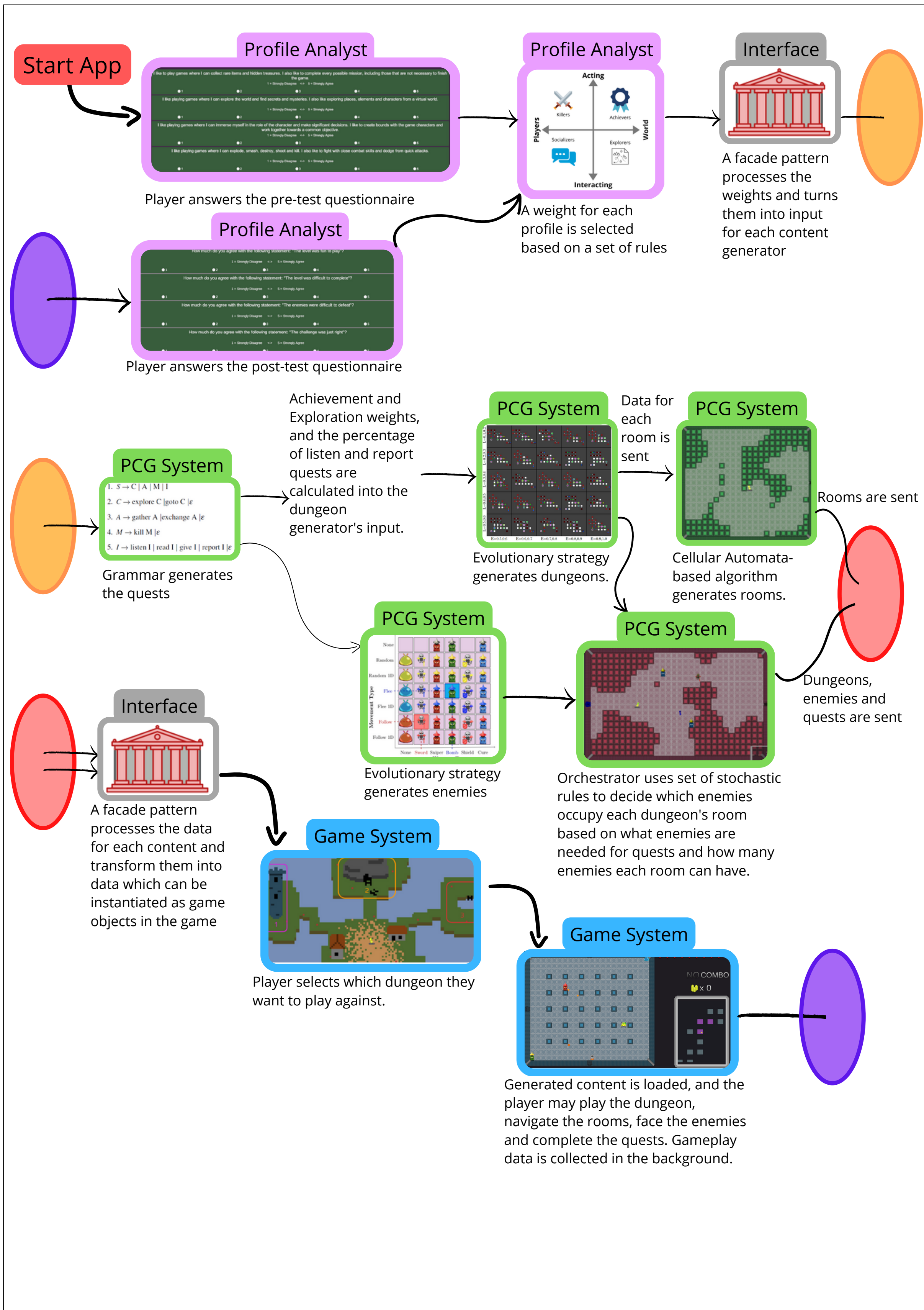
studying the space of the game and the contents, and using known metrics to guide the generation to what the players' desire.

The system's architecture that enables such use of PCG can be summarized in Figure 59, which in itself summarizes the answer to the specific research question: "What type of architecture can effectively coordinate the generation of narrative, levels, and rules facets?". First, the Profile Analyst System enter in action, collecting the player's answers to the pre-test questionnaire, which provides their profile data and is fed to a profile selector, weighting this player against each profile accounted for. Then, a facade interfaces these weights and creates the inputs for the generator algorithms for the PCG System. This system will create quests through one of our grammars, and these quests are fed to the dungeon and enemy generators, which use different evolutionary strategies to generate their contents, depending on the experiment. The generated dungeon is fed to the room generator, and, together with the enemies, also is fed to the orchestrator. The generated room and the orchestrated dungeon with the enemies placed, and the quests are all fed to another facade, which translates content in a way the Unity game engine can understand to place inside the game. Then, the Game System starts, allowing the player to select the level and loading the corresponding content for such level. After the player finishes the level, the content collected during the playthrough may be fed to the Profile Analyst System to update the profile, and the answers of the post-test questionnaire are also collected.

By maintaining the generator algorithms able to evaluate their contents and always provide targeted and feasible contents, we see that the orchestrator algorithm needs only to handle some basic steps of optimizing content placement, as the algorithms and our architecture itself handles the feasibility and enjoyability of the majority of the contents, while targeting them for the player's profile, providing a possible answer for two other specific questions: "How can an orchestrator algorithm process the generated content to make them feasible and more enjoyable when placed together?" and "How can the orchestrator deal with simultaneous content generation, considering or learning from player or game designer profiles?".

As for the last specific research question, "How to adapt the simultaneous content generation within online environments?", the use of model-based algorithms which guarantee the feasibility and that every solution will be at least good enough to solve the required input is enough to solve one of the problems of real-time content generation, that is to deliver the right content without supervision. Another problem for real-time generation is the time needed to generate the solutions. And, as each of our algorithms can generate their contents in less than a second, except for the dungeon generator, which takes a few seconds, this was not a major problem for our system. The major changes were reported at the end of Section 4.3.1, specially regarding the early convergence criteria for the dungeon-generating EA. The next chapter will present the results both corroborating the quality and time needed to build our solutions, and how users evaluated the generated content, specially when comparing content generating for their profiles and those of a different profile.

Figure 59 – An overview of our system, showing the flow of data during its execution and summarizing our methodology. The PCG strategy for each content may vary according to the experiment, as well as when the content is generated (offline vs. online), but the structure is the same for all experiments.



CHAPTER

# 5

# RESULTS

This chapter presents the main research results achieved through experiments aiming to validate our system. However, we first present results for our enemy generator EA described in Section 4.3.3 once it was also a contribution of this thesis. The EA for an enemy generation was evaluated using the early version of our multiple content generator systems, which handled content offline: contents generated before the game had been uploaded and played by the users.

Pereira et al.'s work (PEREIRA; VIANA; TOLEDO, 2021) presents the same results for enemies. It is worth mentioning that (VIANA, 2022) and (VIANA; PEREIRA; TOLEDO, 2022) also report results applying MAP-Elite as an approach for both the dungeon and the enemy generators. Both works had results achieved using our multiple-contents generator systems. The master thesis in (VIANA, 2022) extends the EAs in Pereira et al.'s work in (PEREIRA, 2018) for dungeons and (PEREIRA; VIANA; TOLEDO, 2021) for enemies.

The early version of the multiple-contents generator system needed improvements when running the results in (PEREIRA; VIANA; TOLEDO, 2021), (VIANA, 2022), and (VIANA; PEREIRA; TOLEDO, 2022). This version had the profile analyst collecting gameplay data from players and pre-test and post-test data from questionnaires. However, it still needed a quest system, NPCs, and a dialogue system.

In Section 5.2, we had the results evaluating the offline version of our multiple generator systems as a whole. We added the first version of the quest generator with the grammar method in Section 4.3.5 as a frame to guide the dungeon and enemy generator. We also developed the content orchestrator in Section 4.3.8. In this experiment, we also used the MAP-Elites approach for dungeon and enemy generation from sections 4.3.2 and 4.3.4, respectively. However, this version still needs NPCs, quests, or dialogue systems.

Section 5.3 show our final results, where quests, dialogue systems, and NPCs are present in the gameplay. In this version, we updated the quest generator to use a combined approach with formal grammar and a Markov chain to enhance diversity. Finally, we made the system

work in an online approach: when the player answers the pre-test questionnaire and contents are generated in real-time.

## 5.1 Enemy Generator Results

This section reports the computational results of the enemy generator applying an EA. In this experiment, we use the early version of the system to generate the enemy and dungeons contents procedurally but in an offline fashion. We evaluated the performance and diversity achieved by the EA, the players' feedback from questionnaires, and gameplay data gathered by the system. The results presented here are also in (PEREIRA; VIANA; TOLEDO, 2021).

The summarized workflow is presented in Figure 60 and detailed throughout this section. The player answers a pre-test questionnaire and the profile analyst decides which are the corresponding enemy difficulty and dungeon settings for them. In this case, the dungeons and enemies were generated offline, before the experiment began. Thus, the PCG System was run before the game was provided to the players, and during the experiment, pre-created dungeons and enemies were selected based on the pre-test answers. Then, the data was loaded, and the player could enjoy the content.

### 5.1.1 *Enemy Generator Evolutionary Algorithm's Performance*

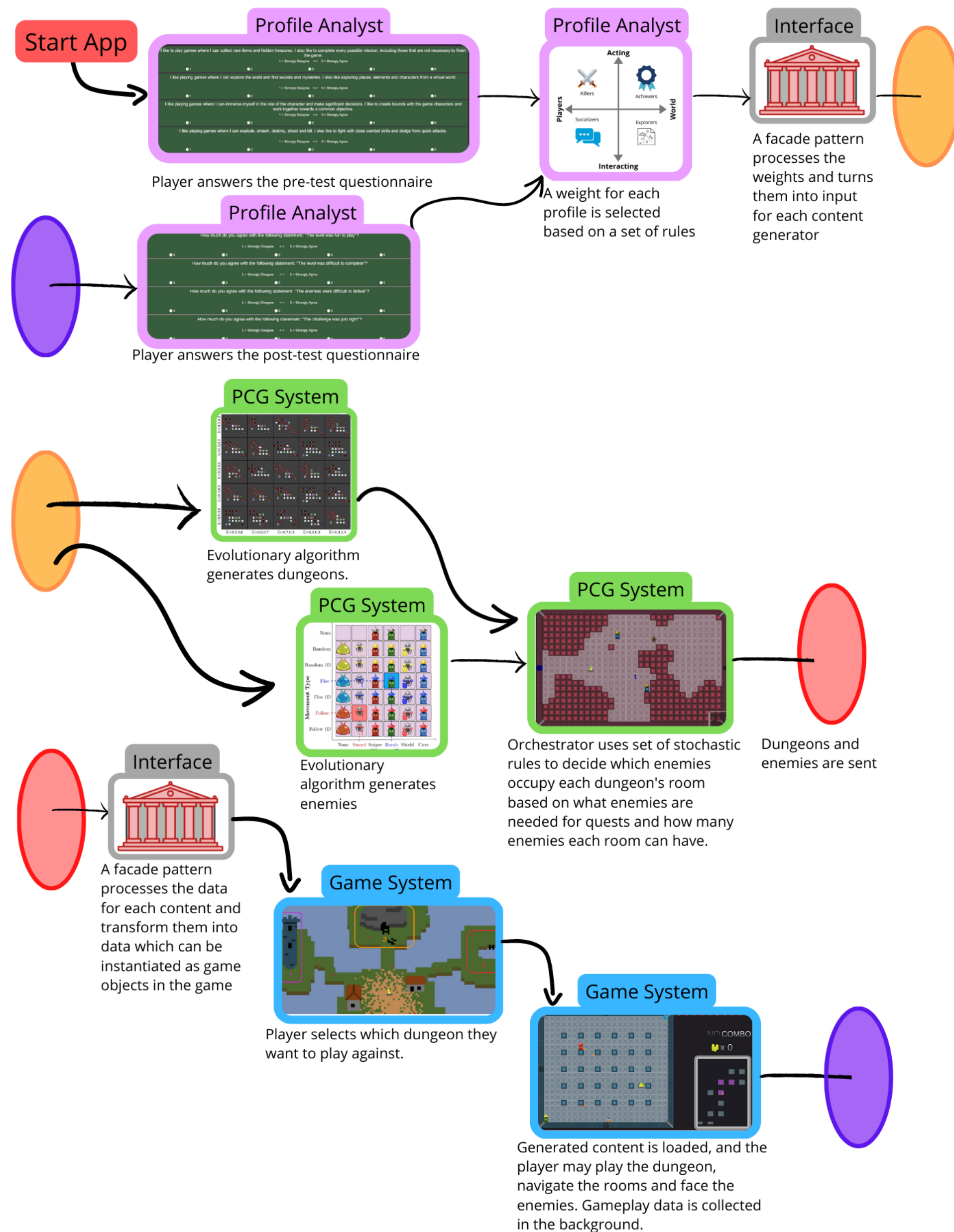
Our first set of experiments tests the convergence time and quality of solutions given by the EA, where we executed the method for nine sets of input parameters, 200 times for each setup, and measured the generation time. The tests run in a relatively standard PC setup: AMD FX-8320 Eight-Core 3.50GHz Processor, 16 GB DDR3 RAM, 223 GB SSD memory, NVIDIA GeForce GTX 1060 3 GB graphics card.

Table 6 shows the results of this experiment, considering three different fitness objectives: Easy, with desired fitness 14; Medium, with fitness 17.5; and Hard, with fitness 22.5. The table contains the average execution time, average, and standard deviation for the whole population's fitness, and the average and standard deviation of the fitness considering only the 20 best individuals across the 200 executions for each input. We can observe that the algorithm is fast, even with large populations, taking less than 0.2s to evolve a population of 10,000 individuals over ten generations and 1.7s for 100 generations. These values had already indicated that we might be able to apply our approach to an online enemy generation.

The algorithm can evolve the 20 best individuals close to the difficulty goal input. The worst-case scenario average is 0.28 units away from the target, with a standard deviation of 0.17. This difference gets a little higher for the whole population: for the hard difficulty, the average exceeded 2.5 units below the desired fitness and had a standard deviation of 2.277.

Therefore, we cannot guarantee that the whole population harbors adequate solutions for

Figure 60 – Data workflow for the experimental setup about the enemy generator EA.




this difficulty. Furthermore, this result is somewhat expected since we evolve 10,000 individuals. However, we can ensure that at least the 20 best individuals, which already guarantees each EA's execution, can conceive a good variety of feasible individuals. This result is very useful to entertain players and increase the uniqueness of each playthrough.

Trying to settle for good results in the lowest time possible, we decided to create our population of enemies using 10,000 individuals and evolve the population for 30 generations for each difficulty setting for the experiments with players. Table 6 highlights in bold such test cases.

Table 6 – Data collected from averaging 200 executions of the EA with different input parameters. The input is abbreviated with D-P-G, D is the difficulty, P is the population size, and G is the number of generations. Regarding difficulty, E, M and H represents Easy = 14, Medium = 17.5, and Hard = 22.5 difficulties, respectively. Finally, *AVG Best* and *STD Best* are collected from the average fitness of the 20 best individuals.

| Input | Time | AVG Best | AVG | STD Best | STD |
|---|---|---|---|---|---|
| M-$10^2$-10 | 0.168s | 17.4 | 16.59 | 0.170 | 1.870 |
| M-$10^3$-10 | 0.174s | 17.60 | 16.57 | 0.017 | 1.872 |
| M-$10^4$-10 | 0.175s | 17.50 | 16.54 | 0.002 | 1.883 |
| M-$10^3$-30 | 0.508s | 17.50 | 16.79 | 0.011 | 1.914 |
| **M-$10^4$-30** | **0.512s** | **17.50** | **16.81** | **0.001** | **1.922** |
| M-$10^5$-30 | 0.660s | 17.50 | 16.81 | 0.000 | 1.922 |
| M-$10^4$-$10^2$ | 1.706s | 17.50 | 16.81 | 0.001 | 1.929 |
| **E-$10^4$-30** | **0.578s** | **14.00** | **14.01** | **0.001** | **1.619** |
| **H-$10^4$-30** | **0.512s** | **22.50** | **19.88** | **0.011** | **2.277** |

### 5.1.2 Enemy Generator Evolutionary Algorithm's Diversity

As we select a group of the best solutions from a single population, concerns about the diversity of said solutions can arise. Therefore, we evaluated the diversity by plotting a parallel coordinates visualization of each numeric parameter from the ten best solutions of a single execution for each difficulty setting. We normalized each value to the interval 0 to 1, using the minimum and maximum values presented for this parameter between the ten evaluated instances. We selected the ten best instead of the 20 best used in the experiments with players to make the visualization easier to read.

The results presented in Figure 61 show that for most parameters, there was a significant diversity, especially when increasing the difficulty. However, the health and damage parameters did not present much variety, especially in the easy setting because they have a more considerable impact on fitness: i.e., when an enemy with two health points is one whole fitness unit (and, therefore, one difficulty unit) above another with one health point; the other parameters are floating-point values and can have more subtle differences.

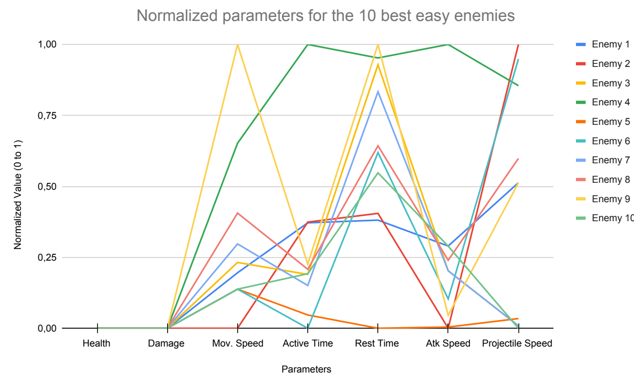
This diversity inside a single-population EA arose because we could use a massive population because of our parallel approach. Therefore, we had many more solutions spread in local optima across the search space, even after several generations.

### 5.1.3 Enemy Generator Evolutionary Algorithm's Player Feedback

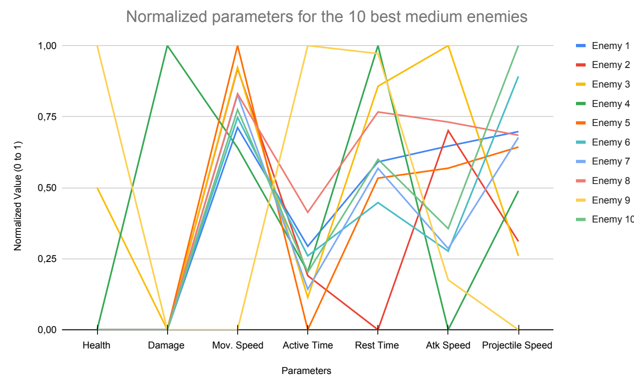
Our experiment with players allowed them to select between three difficulty settings: easy, with a difficulty degree of 14; medium, with a difficulty degree of 17.5; and hard, with a difficulty degree of 22.5. We calculated these values with our difficulty function and determined empirically in previous tests with a smaller number of testers. For each difficulty setting, we executed the EA with the presented parameters. Then, we saved the 20 best non-equal individuals

Figure 61 – Parallel coordinates visualization of the numeric attributes for the top 10 best enemies for each difficulty setting, used in the experiments with players. The charts present normalized parameters.

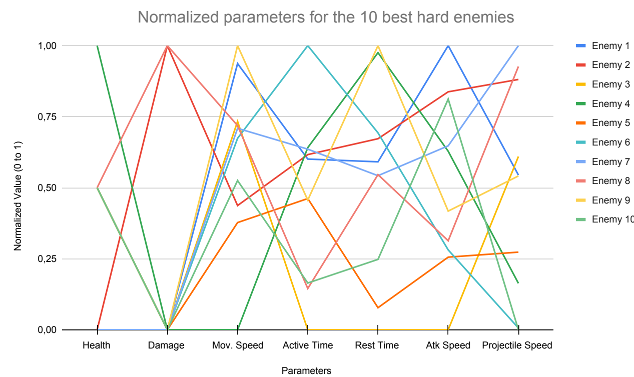
(a) Diversity of the top 10 easy enemies.




(b) Diversity of the top 10 medium enemies.




(c) Diversity of the top 10 hard enemies.




(a total of 60 different enemies) to provide enemy variety for the levels. In these executions, the largest observed difference from the target fitness for any given difficulty was 0.0845.

We uploaded our game prototype to a public server and asked on social networks for anyone interested in playing it. The game presented instructions about the experiment and gameplay. We ensured to follow protocols such as stating that the users were to remain calm, as they were not being evaluated, only the game itself was. We also stated we would be collecting

gameplay data and asking them a questionnaire at the end to answer if they felt like it, but we would not collect any personal data. All our respondents remained anonymous, as it was an opinion survey. Figure 62 presents the questionnaire we applied in our experiment. We also allowed them to quit at any time during the experiment section.

Figure 62 – Experience questionnaire on a 5-point Likert scale. Q1 through Q3 are answered only once per player.

Q1 How would you best describe your overall experience with games (knowledge, playing time, skills, etc.)?

Q2 How would you best describe your experience (knowledge, playtime, skill, etc.) with Action-Adventure games (e.g., The Legend of Zelda, The Binding of Isaac, Darksiders, Uncharted, etc.)?

Q3 What difficulty level do you usually choose to play your games in?

Q4 How much do you agree with the statement: "This level was fun to play"?

Q5 How much do you agree with the statement: "The combat against the enemies was hard"?

Q6 How much do you agree with the statement: "The combat against the enemies' difficulty was adequate to the selected difficulty level"?

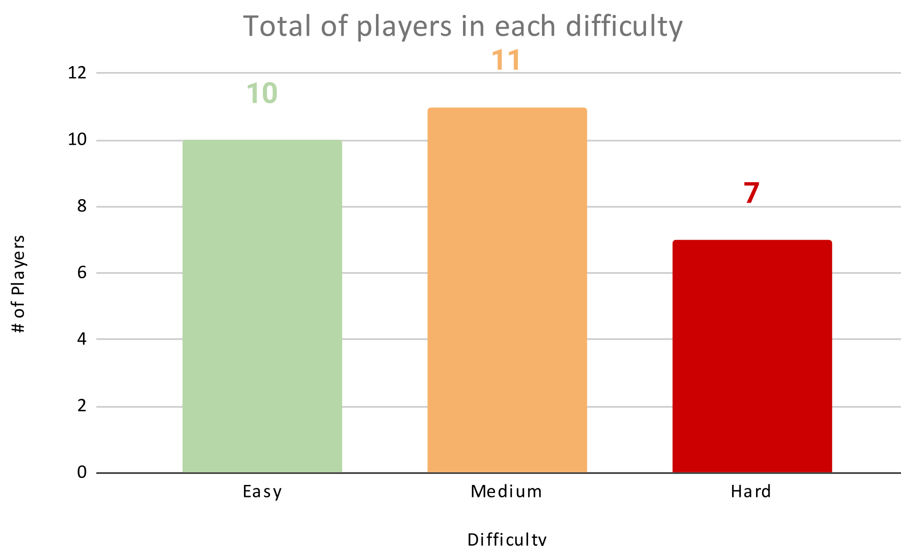
After reading this introduction, the player selects the difficulty level they would like to play (easy, medium, or hard), and the game starts. If they died, they could restart or quit the game, which would take them to our questionnaire screen. They could retry any number of times. After answering our questionnaire, the user could leave the game or go back to the difficulty select screen and start over.

In terms of data collection, we did not collect personal data. We identified the users only by their session starting time and a random ID – it was impossible to check if any user came back in another session. We also collected implicit data, such as if the player succeeded or failed to complete the level.

A total of 16 players answered our questionnaires, but only some answered for all difficulties (Figure 63). As we observe in Figure 62, we divide the questions into two groups. The first is demographic data, answered only once per player, with the first three questions about the players' experience with games and preferred difficulty. The second group is their actual answers about our game prototype and enemies created by the EA.

The first three questions are independent of the chosen difficulty setting, and they show in Figure 64. We observe that most players consider themselves well-experienced with video-games and reasonably experienced with Action-Adventure games. They also prefer the medium difficulty setting, and the hard difficulty comes second. A small sample of players used to play in the easy, very easy, or very hard settings. Therefore, we expected most of them to play in the

Figure 63 – Total number of players by difficulty setting.




medium difficulty, as confirmed in the results of the next section.

In Figure 65's, we observe that players had the most fun in the easy and hard difficulties. We believe this occurred because players who do not enjoy challenging games are happy with the easy level, while those who prefer challenges were entertained with the hard setting. It is important to note that the number of answers for each difficulty is different, as players could play at any difficulty, any number of times. For the medium setting, the answers were mostly neutral, and slightly positive. In this case, we suppose that the players did not evoke a strong feeling of joy nor challenge for most players, or some of the casual players tried these levels and faced more difficulty than expected (as we can confirm with the great loss rate in the next section). So, the overall answer was that the levels were indeed fun to play, disregarding difficulty.

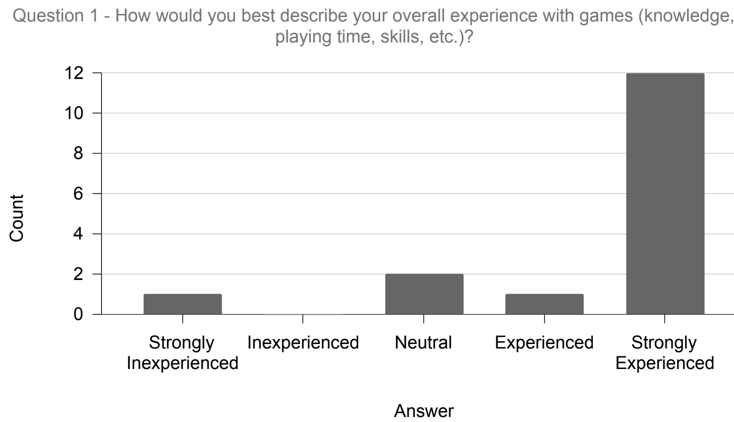
Figure 66 shows the player's perception of difficulty for each setting on the left column (figures 66a, 66c, 66e). The easy difficulty was considered not challenging for most players (as expected), while the medium one had mixed reviews, tipping a little more for the opinion that it was difficult. Moreover, the hard difficulty was unanimously considered difficult. Those answers further confirm the ability of our EA to generate a variety of enemies that also respect the input difficulty setting.

Figure 66 shows that most users agreed the difficulty was adequate to what they expected on the right column (figures 66b, 66d, 66f). The same was true for the remaining difficulties. Again, the medium one was a little more controversial, as it was probably a little harder than expected.

We collected the number of failed and succeeded attempts for each difficulty degree. Figure 67 shows the results regarding the players' performance. We observe that the medium level had the most attempts, reiterating the preference of users to play in such difficulty, as

Figure 64 – Results from the demographic questions.

(a) Answers for Q1.




(b) Answers for Q2.

(c) Answers for Q3.

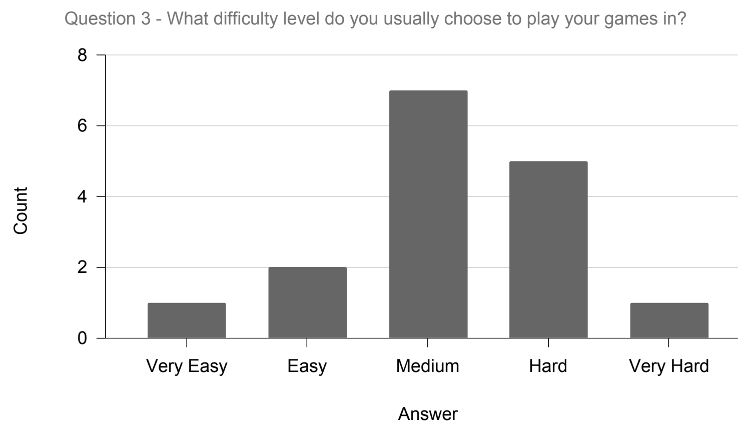
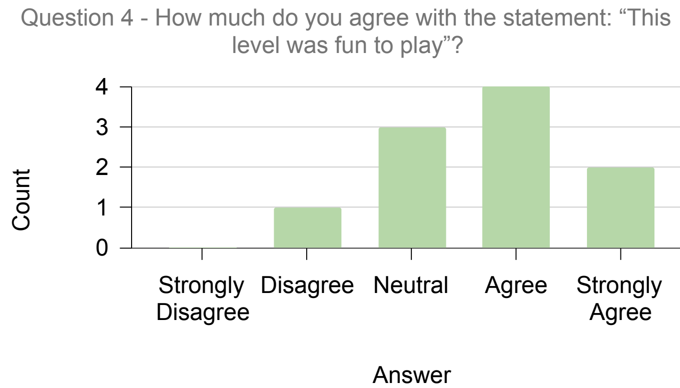
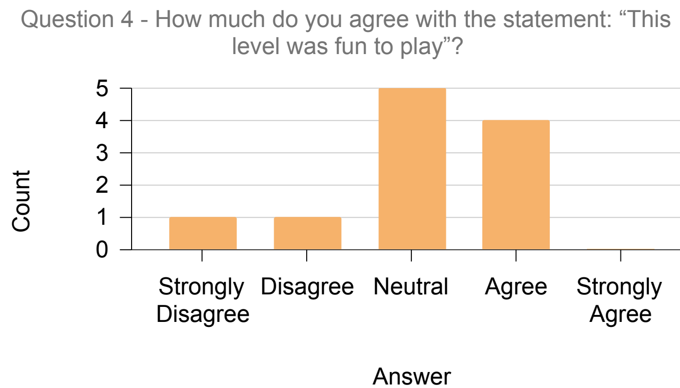


Figure 65 – Results from how fun the game was for each difficulty.

(a) Answers for Q4, for the easy difficulty.




(b) Answers for Q4, for the medium difficulty.




(c) Answers for Q4, for the hard difficulty.

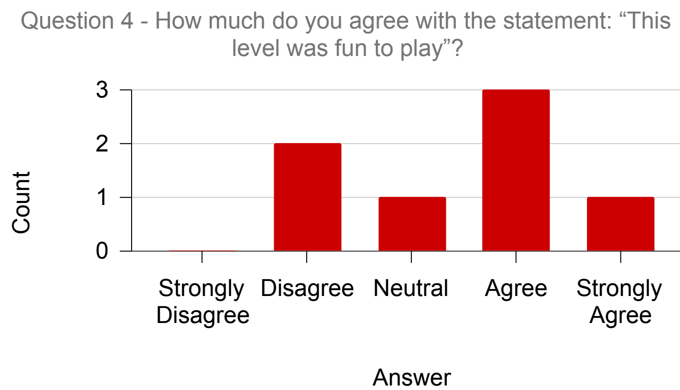
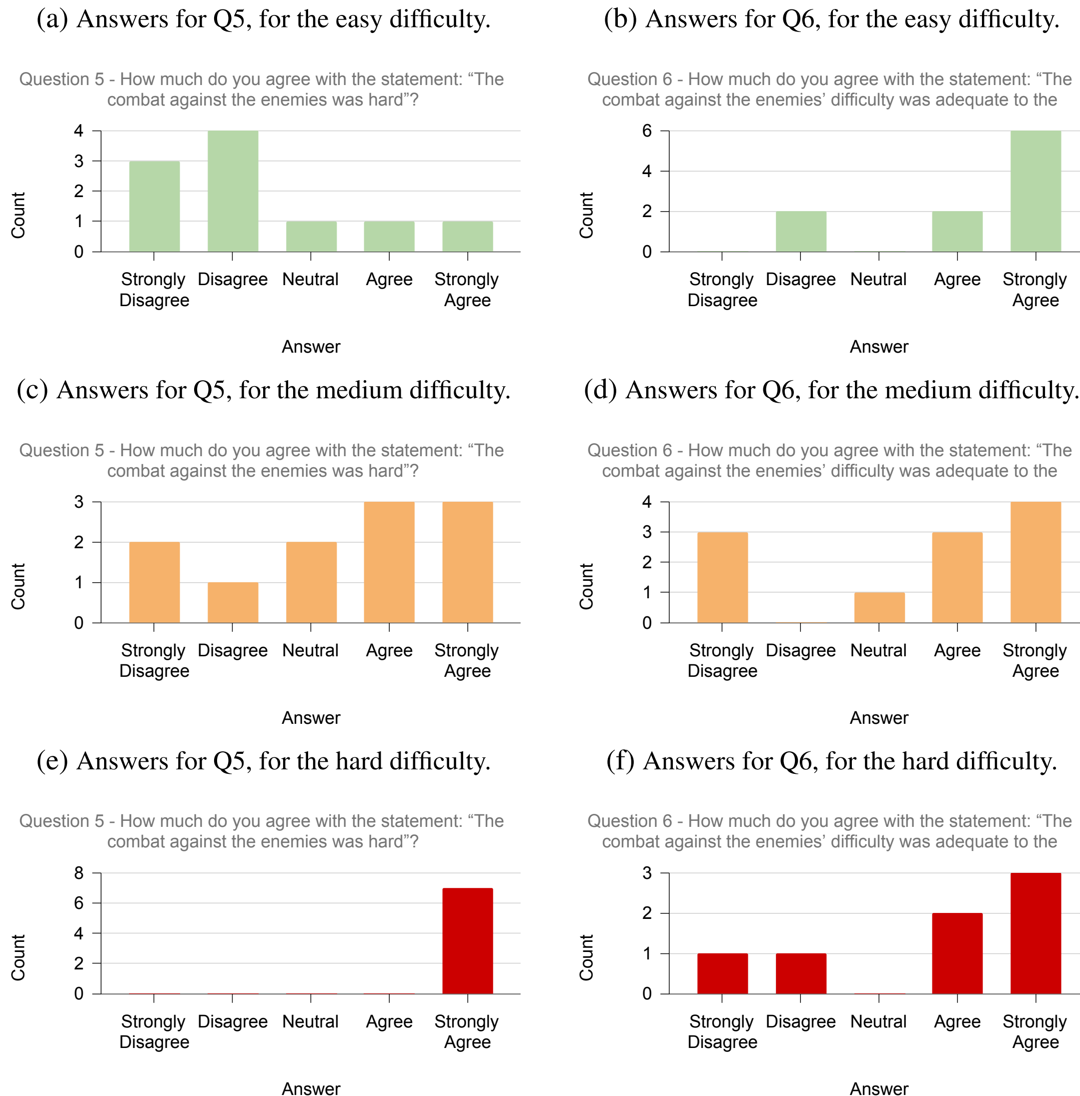


Figure 66 – Results from the questions about the combat difficulty (left column) and difficulty adequacy (right column) for each setting.




(a) Answers for Q5, for the easy difficulty.

(b) Answers for Q6, for the easy difficulty.

(c) Answers for Q5, for the medium difficulty.

(d) Answers for Q6, for the medium difficulty.

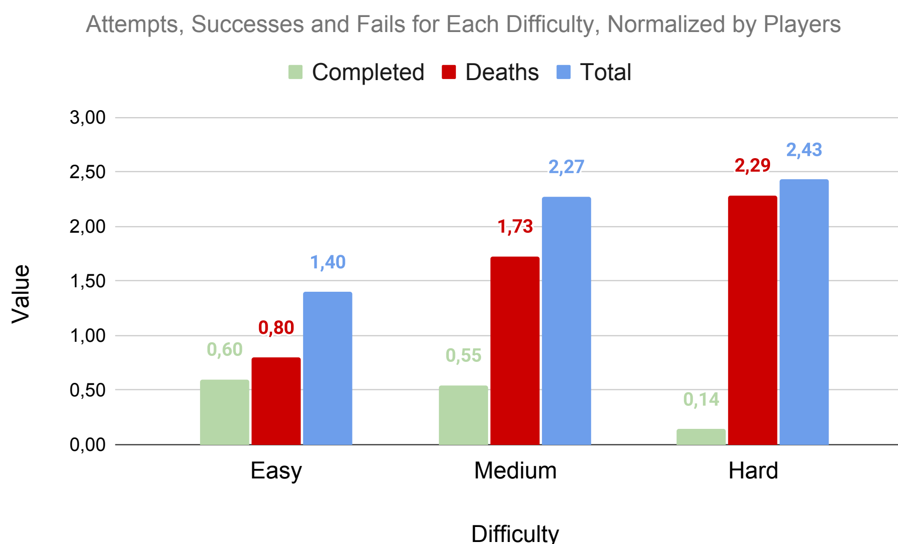
(e) Answers for Q5, for the hard difficulty.

(f) Answers for Q6, for the hard difficulty.

shown in Figure 64. However, normalized by the number of players, we observe that the hard difficulty had more attempts by player, possibly because players who liked the hard setting felt challenged to complete it. When comparing success rates, the easy difficulty was the one with the greatest one, followed by the medium and the hard one. These results highlight that our algorithm can successfully create enemies with different difficulty settings, and these settings were progressively harder, as intended.

These data also show that our target fitness for each difficulty setting could be slightly lower to allow more players to complete the game. It is usually intended for easier difficulties to allow victory in most attempts. While our current medium fitness could be substituted for the easy one, allowing a little above half of the players to win. Then, the current medium difficulty could be used as the hard setting once most players failed, and the current hard difficulty as very hard since only one player won.

Figure 67 – Total attempts players gave to each difficulty, how many times they succeeded and how many times they failed. Each series is normalized by the number of players on said difficulty: 10 for the easy, 11 for the medium and 7 for the hard, as shown in Figure 63.




## 5.2 Offline Generator Results

The release used for the experiments in this section are tagged in the repository as tag v0.2.0[1]. Our respondents can play the game version via a Unity's WebGL in a Firebase server[2]. This setup still generates content in an *offline* fashion, but uses the MAP-Elites version of both the enemy and dungeon generators (described in sections 4.3.4 and 4.3.2, respectively). The results presented here are also in (PEREIRA; VIANA; TOLEDO, 2022)

The summarized workflow is presented in Figure 68 and detailed throughout this section. The player answers a pre-test questionnaire and the profile analyst decides which are the corresponding weights for the quest generator for them. In this case, the quests, dungeons, and enemies were generated offline, before the experiment began. Thus, the PCG System was run before the game was provided to the players, and during the experiment, pre-created quests, dungeons, and enemies were selected based on the pre-test answers. Then, the data was loaded, and the player could enjoy the content.

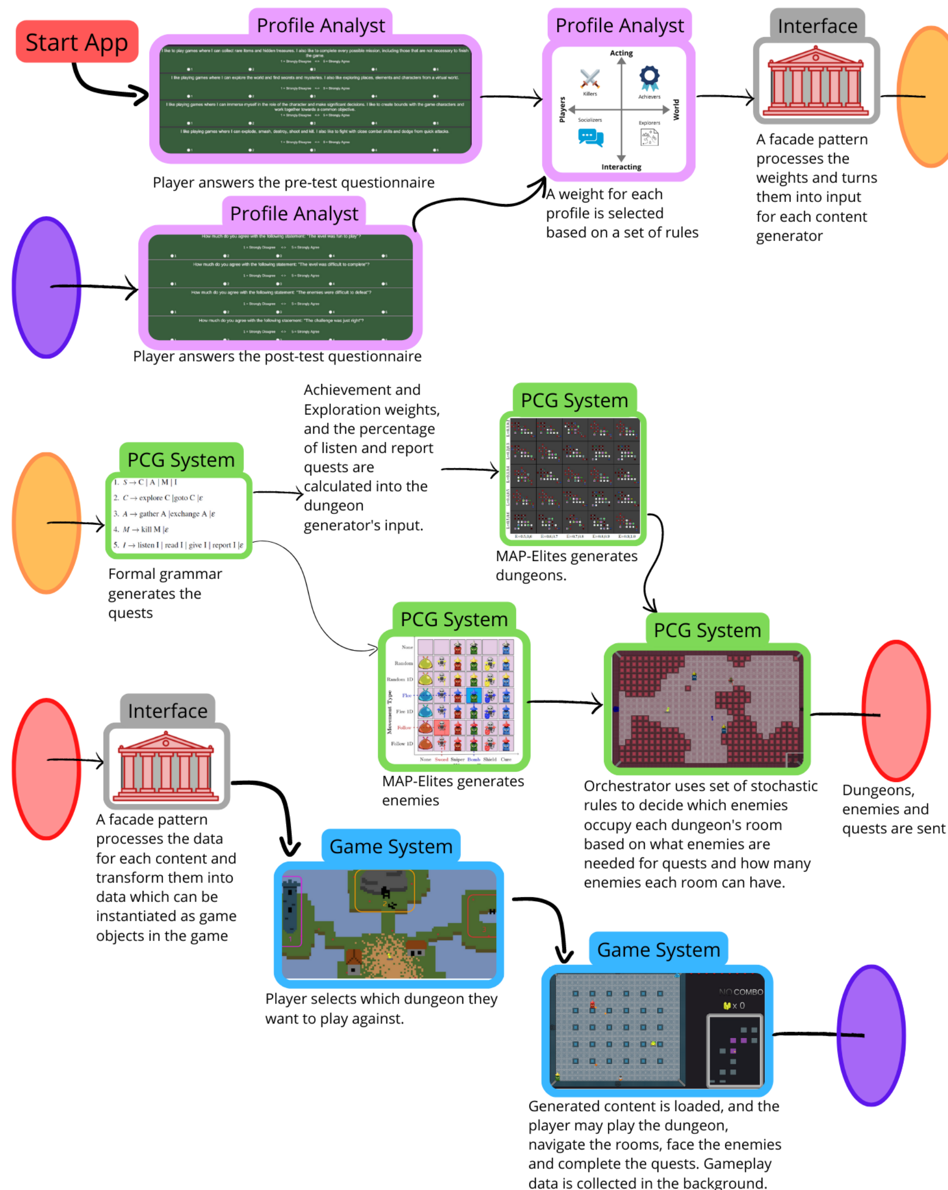
### 5.2.1 Experimental Setup

In this version, the player must select a dungeon from three options presented in the overworld scene in Figure 69. The figure also shows our player avatar (yellow robot) inside a dungeon room, in which there are three enemies that the player must kill.

For this setup, our Profile Selector container from the Profile Analyst system, described

[1] <https://github.com/LeonardoTPereira/Overlord-Project/releases/tag/v0.2.0>
[2] <https://overlord-pcg-en.web.app/>

Figure 68 – Data workflow for the offline experimental setup.




in Section 4.2.1, uses a simple weighted rule to give the player one of four possible profile types based on their answers to our pre-test questionnaire. They influence the quests' generation, presented in Section 4.3.5, and we employ these quests as a canvas for the other generators. The first profile is **A**chievement, 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 more leisurely. It influences the generators to create fewer enemies, fewer locked-door missions, and

Figure 69 – Game overworld (upper) and dungeon (bottom).

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 (YEE; DUCHENEAUT, 2018) player motivation, selecting the ones that our current game best-allowed players to experience.

We asked players to answer 12 questions in our pre-test questionnaire (part of the Questionnaire component, described in Section 4.2.1). They answered them only once when starting the game to define their profile. We adapted the questions from Rivera-Villicana et al.(RIVERA-VILLICANA *et al.*, 2018), Vahlo et al.(VAHLO *et al.*, 2017b), and Yee and Ducheneaut's(YEE; DUCHENEAUT, 2018) work 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. 70 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 Equation 5.1), ranked in descending order: the one with greater value is the profile most fitting to the player, and the one with the smaller value is 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

Figure 70 – Pre-Test Questionnaire.

- 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;

and enemy generators. In case of a draw, we use the order of the question enumeration, as in Equation 5.1: Mastery has preference over Creativity, then Achievement, and last Immersion. For Q3, we subtract three from the answer to make the medium difficulty equal 0. In this case, easier difficulties will reduce the Mastery weight, and harder ones will 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 with the world, and collecting items).

$$\begin{aligned} M &= Q_3 - 3 + Q_4 + Q_5 \quad & A &= Q_8 + Q_9 + 5 - Q_{12} \\ C &= Q_6 + Q_7 + 5 - Q_{12} \quad & I &= Q_{10} + Q_{11} + 5 - Q_{12} \end{aligned} \tag{5.1}$$

Section 4.3.5 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.

The Overlord System created a collection of levels and enemies from the four different

profiles. 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 4.3.2 and 4.3.4. Thus, there is a total of 42 possible enemies and 25 dungeons. Using a set of previously created content, we chose this offline approach 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 pre-test 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. 69. If the player dies in a dungeon and wants to select another, a new random dungeon-enemy combination of that questline 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 post-test questionnaire presented in Figure 71 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.

Figure 71 – Post-Test Questionnaire for the Offline Experiment.

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 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;

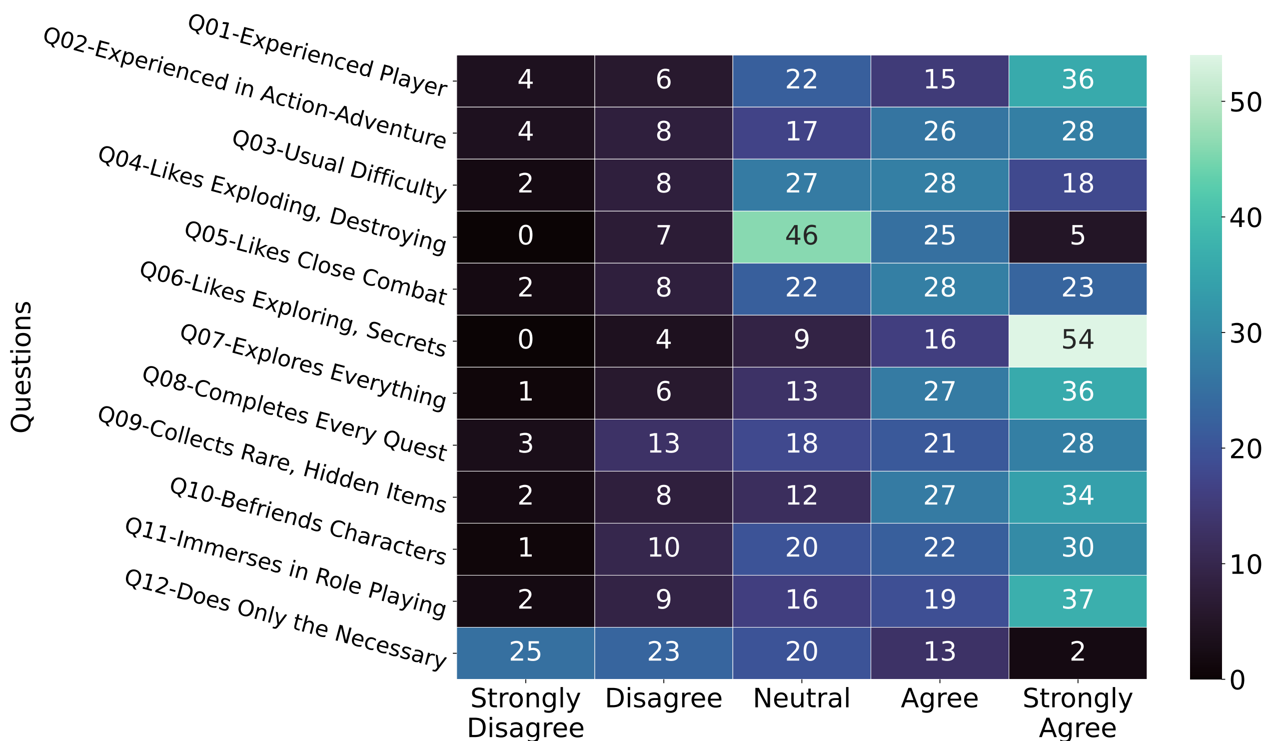
There is no repetition in the draws, so the player always plays a new dungeon for the first time and avoids bias from knowing it from a previous playthrough. We execute this loop until all dungeons are removed from the poll or the player decides to exit the game. If no dungeons are left, a message informs the player that the game ended. Thus, we are evaluating if the system can create content to be fun and diverse enough for players. These are grouped into four profiles based on their answers to the pre-test 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. 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 different gameplay sessions. 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).

### 5.2.2 Experimental Results

A total of 83 players answered the pre-test questionnaire and jointly played and evaluated (with the post-test questionnaire) 204 levels. Table 70 details the questions, and Fig. 72 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

Figure 72 – Heatmap grouping the answer count for each point in the 5-point Likert scale for the 12




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 of 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 game world's story, and its NPCs, 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 collected 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 pre-test 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 post-test question (from Fig. 71), grouped by the profile of the respondent, in Fig. 73. 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). This tendency continues when asked about 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.

Table 7 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.

As there were not enough players for each test and control group for a statistical analysis of the answers comparing each group, we will conduct an analysis considering the whole test group (players who played their actual profile, highlighted in Table 7, against the whole control group. This analysis may provide more robust proofs on the quality of our generators.

First, we must find the statistical power of our test, then, conduct a test to check whether the sample is Gaussian or not, only after this we can select the appropriate statistical hypothesis

Figure 73 – 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.

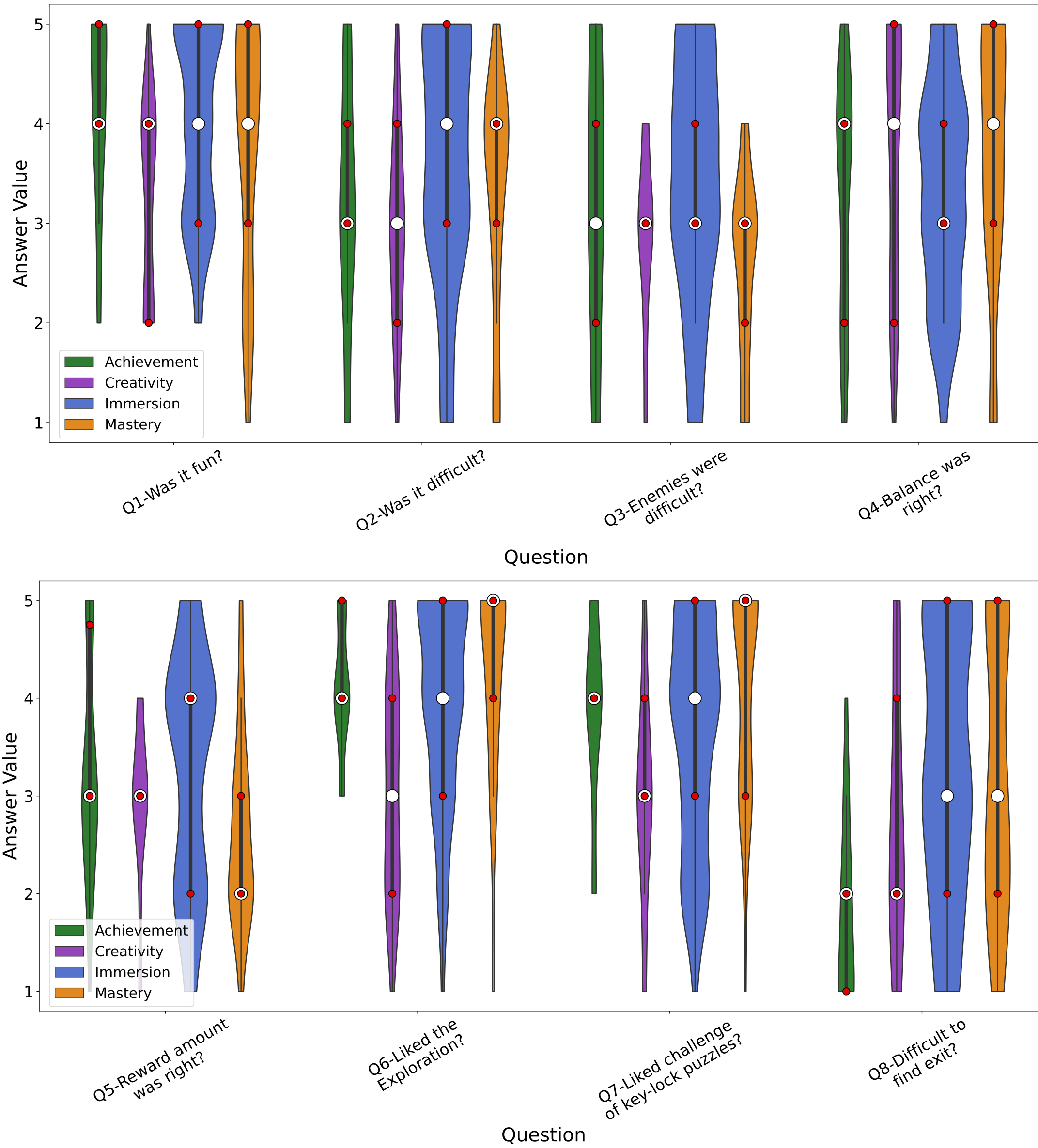



Table 7 – Post-test answers count, grouping players by the profile they played as vs. true profile. Control Group is colored.

| **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** |

test to verify whether both populations share the same mean or not. We use the *statsmodel* Python

library [3] for this, and *scipy.stats* library for the analysis [4].

For the power analysis we want an alpha of 5%, that is, there is only 5% chance that our results fall under the type I error of rejecting an actually true H0 (Null Hypothesis). We will work under a Cohen's D effect size of 0.5, that is, a medium effect, where the two groups differ by half a standard deviation, usually, this value is enough to measure a difference that can be discerned by the naked eye. Then, we have the ratio between our two population sizes: we have 143 post-test answers from people of matching profiles, versus 61 from different profiles, a ration of, approximately, 0.4266. We want, first, to not only compare if the populations are different, but, preferably, if one is greater or smaller than the other, so we consider the *greater* alternative. These inputs result in a statistical power of 0.947, that is, a little over 5% probability that our results fall under the type II error of not rejecting an actually false null hypothesis.

With only 5% chance of having any of the two error types, we can conclude that our sample is good enough for a robust statistical hypothesis test. Our two samples are independent, as the playthrough of both groups do not affect each other's results. We must then check whether the distribution of each answer is Gaussian or not. Using the Shapiro-Wilk test of normality, we find that the population of answers for the eight post-test questions are not Gaussian, with p-values very close to 0 in all of them. Therefore, we use the Mann-Whitney U test, which is recommended for independent yet non-Gaussian populations.

Therefore, we first conduct the Mann-Whitney U test, considering that rejecting the null hypothesis means that the answers from players with a matching profile are greater than those who played with a different profile. Table 8 shows the results. There, we observe that except for the enemy difficulty and the received reward, all answers for the group of matching profile were statistically greater than those with different profiles. This means that playing with content generated aimed by a profile matching the player's preference resulted in a self-evaluation of greater fun, harder overall difficulty and of finding the exit, more balanced challenged, better acceptance of exploration and finding the keys.

For the two non-rejected null hypothesis, we conduct the same analysis, but for the two-sided alternative, that is, to reject the H0 now means that both populations are different. We analyze the power of the test, using the same parameters as before, but for the two-sided alternative, resulting in a power of 0.902, that is, less than 10% probability of providing a type II error. Then, for the Mann-Whitney U test for question 3 we found a p-value of 0.603, failing to reject H0 and, therefore, concluding that the enemies' difficulty was perceived as the same for both groups. Then, for question 5, the same analysis provides a p-value of 0.027, meaning that both populations have a different mean and, therefore, the players with a different profile liked better the rewards than those of matching profiles.

[3] <https://www.statsmodels.org/devel/>
[4] <https://docs.scipy.org/doc/scipy/reference/stats.html>

Table 8 – Results of the Mann-Whitney U test, considering that rejecting the null hypothesis means that the answers from players with a matching profile are greater than those playing a different profile. We reject the hypothesis with a p-value equal or smaller than 0.05, meaning less than 5% chance of a type I error.

| Question | p-value | Result |
|---|---|---|
| **Q1—The level was fun to play** | **0.000** | **Reject H0** |
| **Q2—The level was difficult to complete** | **0.001** | **Reject H0** |
| Q3—The enemies were difficult to defeat | 0.302 | Don't reject H0 |
| **Q4—The challenge was just right** | **0.002** | **Reject H0** |
| Q5—The rewards were on the proper amount | 0.986 | Don't reject H0 |
| **Q6—I liked the amount of exploration on this level** | **0.000** | **Reject H0** |
| **Q7—I liked the challenge of finding the keys to this level** | **0.000** | **Reject H0** |
| **Q8—It was difficult to find the exit/goal of this level** | **0.000** | **Reject H0** |

## 5.3 Online Generator Results

We now present results for the whole system, which includes quests, dialogue systems, and NPCs through an online generation of gameplay. Our experiments generate the contents in real-time when the player answers the pre-test questionnaire. We used the EA approach to generate dungeons (described in Section 4.3.1), as it converged faster to a suitable solution, and the MAP-Elites for the enemy generation (shown in Section 4.3.4). Furthermore, we used the new version of the quest generation algorithm with Markov Chain to add variety and reduce repetition of the same type of quest in a given quest line, as described in Section 4.3.6.

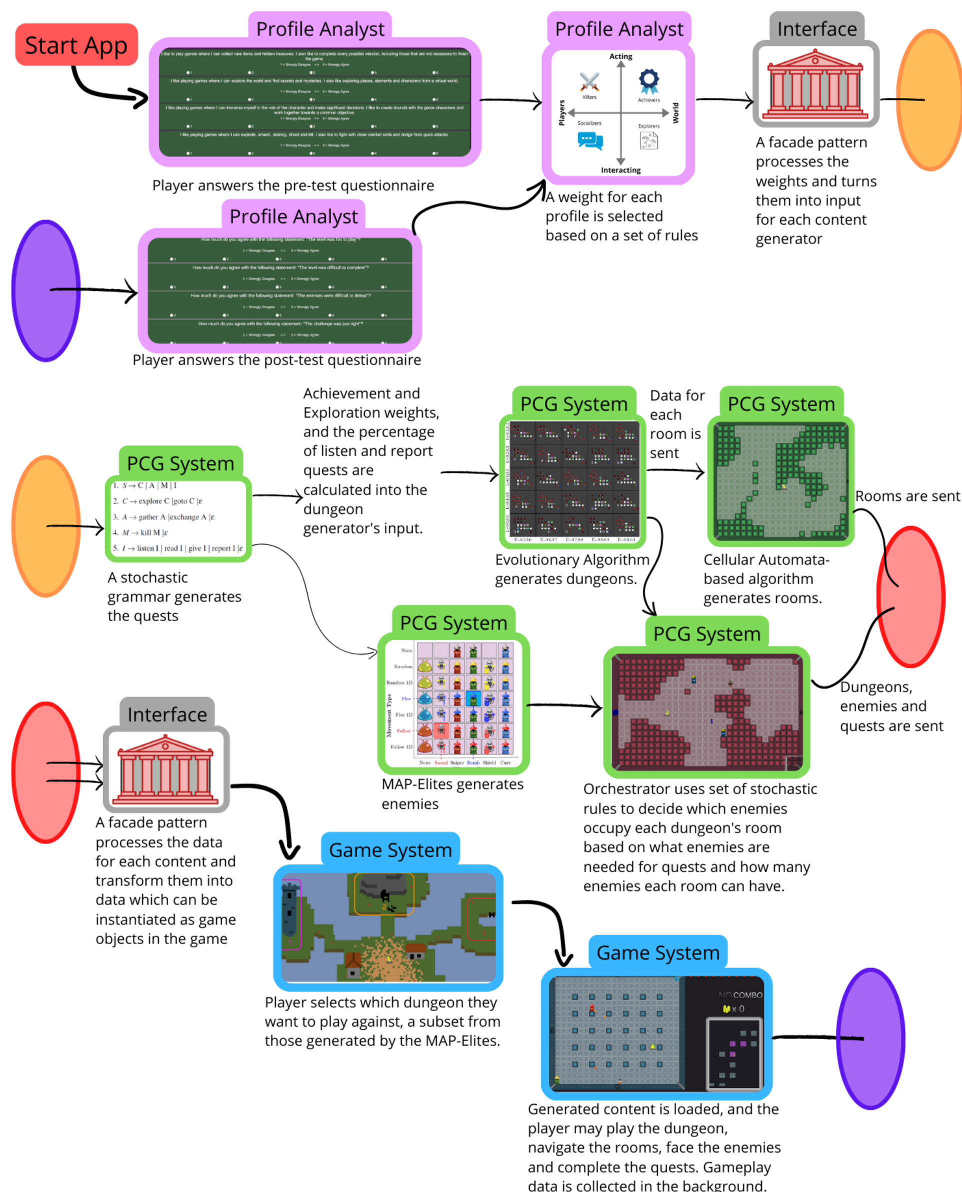
The summarized workflow is presented in Figure 74 and detailed throughout this section. The player answers a pre-test questionnaire and the profile analyst decides which are the corresponding weights for the quest generator for them. In this case, the quests, dungeons, and enemies are generated online, that is, as soon as the player answers the pre-test, the generation of a new set of contents begins. When the content was ready, the data was read and loaded as game objects, and the player could enjoy the content.

### *5.3.1 Experimental Setup*

In this setup, we got a small group of ten anonymous players to answer a new pre-test questionnaire with only four questions, one about each profile, translating into the player's affinity to each type of profile: Achievement, Creativity, Immersion, and Mastery. Figure 75 shows the questions.

Each player can play 12 different dungeons from a Level Selector, as shown in Figure 76. Each level will create new content for the game at the moment of the selection, passing the weight of each profile to the grammar generator, as shown in Section 4.3.6. The resulting quest line provided by the quest generator creates the dungeons, enemies, and rooms, all in real time. When the loading screen in Figure 77 completes, the player will play the created content by

Figure 74 – Data workflow for the online experimental setup.




pressing the mouse left-click button.

The 12 levels' input for the generator are some permutations of answering the pre-test questionnaire with 1, 2, 3, and 4 for each question. As these answers are converted to weights, normalized from 0.2 to 1.0, this equivalent to feeding the generators 0.2, 0.4, 0.6, and 0.8 as inputs. 1.0 corresponds to answering with 5 in any of the questions, and was discarded to reduce the number of alternatives and avoid outliers. The whole combination of inputs that each player could play are presented in Table 9.

A build with the online generator configuration can be played via browser in the following

Figure 75 – Pre-Test questionnaire for the online experiment. The questions are directly translated to the player's preference for the achievement, creativity, immersion and mastery profiles, respectively.

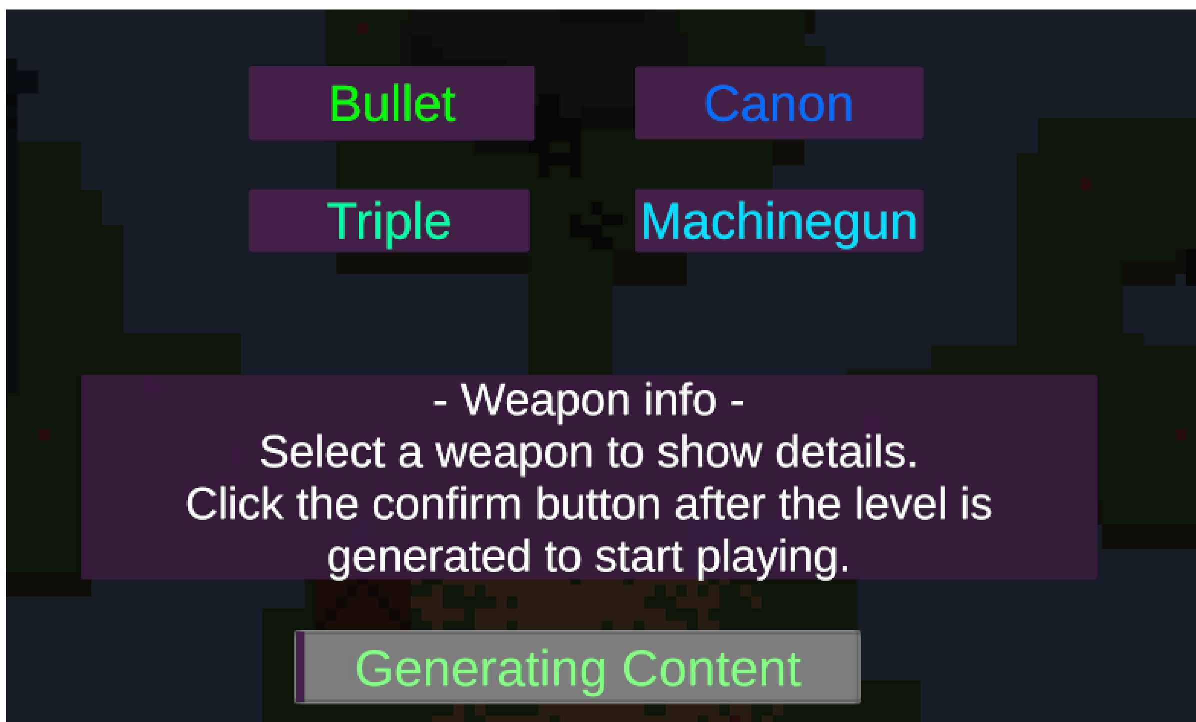
Q1 I like to play games where I can collect rare items and hidden treasures. I also like to complete every possible mission, including those that are not necessary to finish the game.

Q2 I like playing games where I can explore the world and find secrets and mysteries. I also like exploring places, elements, and characters from a virtual world.

Q3 I like playing games where I can immerse myself in the role of the character and make significant decisions. I like to create bonds with the game characters and work together towards a common objective.

Q4 I like playing games where I can explode, smash, destroy, shoot and kill. I also like to fight with close combat skills and dodge from quick attacks.

Figure 76 – Loading screen for the real-time level generation game prototype. The player can select any weapon they wish, and play after the content is created.




link [5], although taking directly into account the player's answer in the pre-test as the generator's input. Before presenting the results with the ten players who played this version of the game, we first show the new settings for the quest and dungeon generators and some computational results for each.

5 <https://overlord-thesis.web.app/>

Figure 77 – Loading screen for the real-time level generation game prototype. The player can select any weapon they wish, and play after the content is created.

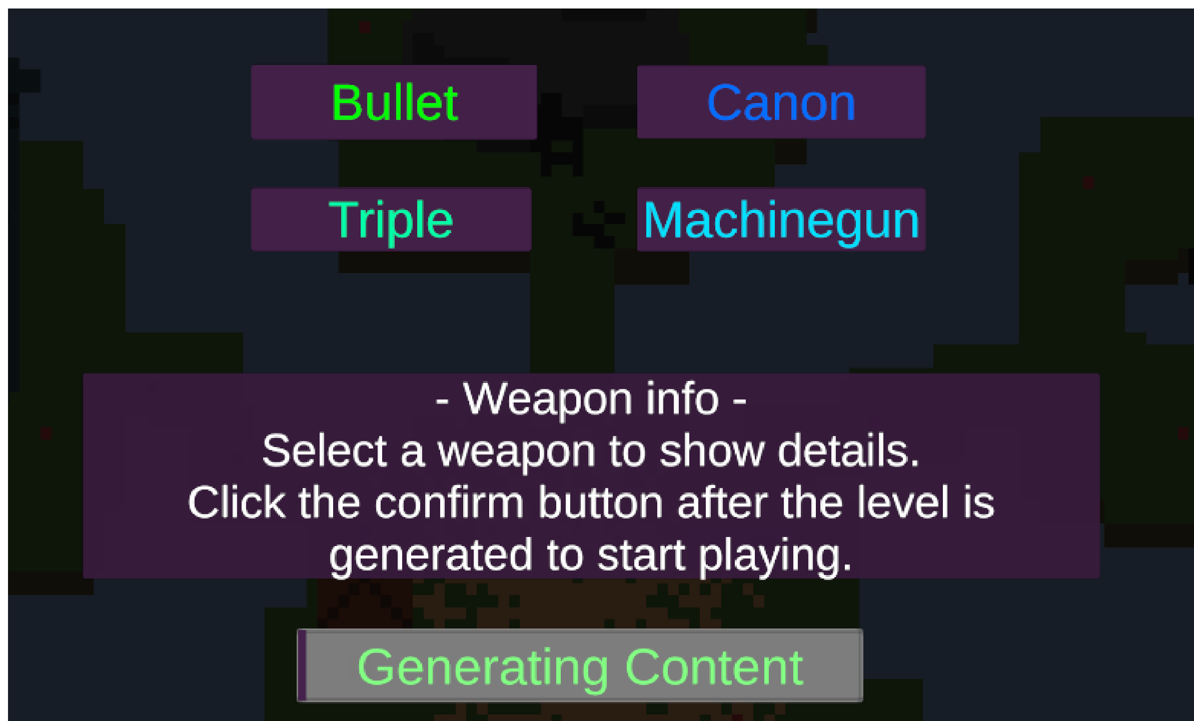



Table 9 – The 12 inputs used to create the content for the online-generation experiment, as well as what are the inputs for the pre-test questionnaire they represent and what weights each answer is converted when normalized as inputs for the quest generator.

| | Answer value → weight input for quest generator | | | |
|---|---|---|---|---|
| **Input Name** | **Achievement** | **Creativity** | **Immersion** | **Mastery** |
| A=1, C=3, I=2, M=4 | 1 → 0.2 | 3 → 0.6 | 2 → 0.4 | 4 → 0.8 |
| A=1, C=3, I=4, M=2 | 1 → 0.2 | 3 → 0.6 | 4 → 0.8 | 2 → 0.4 |
| A=1, C=4, I=3, M=2 | 1 → 0.2 | 4 → 0.8 | 3 → 0.6 | 2 → 0.4 |
| A=2, C=1, I=3, M=4 | 2 → 0.4 | 1 → 0.2 | 3 → 0.6 | 4 → 0.8 |
| A=2, C=1, I=4, M=3 | 2 → 0.4 | 1 → 0.2 | 4 → 0.8 | 3 → 0.6 |
| A=2, C=4, I=1, M=3 | 2 → 0.4 | 4 → 0.8 | 1 → 0.2 | 3 → 0.6 |
| A=3, C=2, I=1, M=4 | 3 → 0.6 | 2 → 0.4 | 1 → 0.2 | 4 → 0.8 |
| A=3, C=2, I=4, M=1 | 3 → 0.6 | 2 → 0.4 | 4 → 0.8 | 1 → 0.2 |
| A=3, C=4, I=2, M=1 | 3 → 0.6 | 4 → 0.8 | 2 → 0.4 | 1 → 0.2 |
| A=4, C=1, I=3, M=2 | 4 → 0.8 | 1 → 0.2 | 3 → 0.6 | 2 → 0.4 |
| A=4, C=2, I=1, M=3 | 4 → 0.8 | 2 → 0.4 | 1 → 0.2 | 3 → 0.6 |
| A=4, C=3, I=2, M=1 | 4 → 0.8 | 3 → 0.6 | 2 → 0.4 | 1 → 0.2 |

### 5.3.2 Computational Experiments

Using the grammar presented in Section 4.3.6, we conduct a statistical analysis over the content generated by the quest generator. For each of the 12 inputs presented in Table 9 we ran the whole content-generation process 110 times and analyzed the data. We will analyze data from the dungeon's fitness and its components, as well as how many quests of each type were generated. More specifically: the normalized distance from the dungeon's values to the

inputs (Equation 4.12), the normalized usage of rooms and locks (Equation 4.13), the normalized enemy sparsity (Equation 4.10), the normalized enemy standard deviation (Equation 4.11), the final fitness, the number of rooms, locks, keys, items, and enemies in the dungeon, the linearity, the total quests, and how many quests are in the quest line for each non-terminal (A, C, I, M) from Figure 34.

In this version of the Dungeon Generator, presented in Section 4.14, we changed some generator settings, especially its fitness and stop criteria. We maintained the 5% mutation chance, 100% crossover chance, elitism of the best individual, selection tournament of 2 individuals, 100 individuals per population. The algorithm stops if the best individual's fitness is lower than 0.1, if there is no decrease in the best fitness after 30 generations, or if 200 generations have evolved.

We conduct the same experimental setup as mentioned in Section 5.2: we found that having 110 samples in our population gave a statistical power of over 0.95 for our tests, that is, there is less than 5% chance that our results fall under the type II error of not rejecting an actually false null hypothesis. We used a Cohen's D effect size of 0.5, that is, a medium effect. As this is a controlled experiment, the ratio between populations was 1, as everyone was executed 110 times.

Our samples are independent, as the execution of the algorithm for one input does not affect the results for other inputs. We must then check whether the distribution of each answer is Gaussian or not, with the Shapiro-Wilk test. As we are experimenting on multiple populations, a One-Way ANOVA test is conducted for those that are Gaussian, and a Kruskal-Wallis h test is used for the ones that are not. For the sake of brevity, we found that some groups of populations were Gaussian and others not, but only for the normalized usage of rooms and locks we failed to reject the null hypothesis (p-value of 0.982). This means that for any input, the convergence was the same. For the others, we rejected the null hypothesis, but we do not know for which pair (or pairs) of inputs.

Therefore, we must conduct a post-hoc Dunn's test to check which pairs reject the null hypothesis (are different) and which do not (cannot be discarded as the same). For each test we present a table: the column name corresponds to the value of the quest-generator's input in the same order we maintain throughout the document: first the Achievement weight, then Creativity, Immersion and lastly Mastery. The letters had to be omitted for the table to fit in the page. The first column shows the name of the analyzed variable, as well as the same inputs, but now accompanied by their letters, for better understanding. As the matrix is square, we shaded the diagonals for a better reading. Then, every cell which we discard the null hypothesis (that both set of values come from the same population) we highlighted in red and underlined, to further easy the comprehension.

We first analyze if all inputs had the same level of the dungeon's convergence to the desired input, in Table 10. As the table shows, only two inputs did not converge in the same range as the others: the ones with low weights for achievement and creativity quests, but high

hates for immersion and mastery. They were different from all others except themselves. The lower creativity and achievement possibly caused the levels to be linear, have few keys and be small, which may have become a little easier for the EA to converge. There are two other outliers: two of the levels with the highest achievement rate were considered different from the ones with the highest mastery. Although the mastery does not affect directly the dungeon's inputs, having a higher weight on it means the others are lower. Higher achievement demands more branching, and combined with a lower immersion and somewhat large creativity values generates larger dungeons. These larger and more branched dungeons may cause the EA to converge to less optimal dungeons than the inputs of high mastery, creating this difference.

Table 10 – Dunn's test result for the normalized distance from the desired input for the levels generated for each quest generator's input.

| Distance from Input | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 1.000 | 0.211 | 0.000 | 0.000 | 0.448 | 1.000 | 0.558 | 0.890 | 0.726 | 0.007 | 0.023 |
| **A1C3I4M2** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.387 | 0.805 |
| **A1C4I3M2** | 0.211 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.107 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A2C1I3M4** | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 |
| **A2C1I4M3** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A2C4I1M3** | 0.448 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.258 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A3C2I1M4** | 1.000 | 1.000 | 0.107 | 0.000 | 0.000 | 0.258 | 1.000 | 0.334 | 0.558 | 0.448 | 0.003 | 0.010 |
| **A3C2I4M1** | 0.558 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.334 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A3C4I2M1** | 0.890 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.558 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C1I3M2** | 0.726 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.448 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C2I1M3** | 0.007 | 0.387 | 1.000 | 0.001 | 0.000 | 1.000 | 0.003 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C3I2M1** | 0.023 | 0.805 | 1.000 | 0.000 | 0.000 | 1.000 | 0.010 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

A somewhat similar issue appears in Table 11, where the levels created from the inputs with higher achievement were considered as having a different sparsity than the others. This may be related to the convergence overcompensating on this aspect for the slightly lower convergence to the desired input. As the usage parcel was not discarded as being similar, and in the standard deviation, shown in Table 12 the differences does not appear to be focused on a specific group of inputs, which is also shown in Table 13, comparing the resulting fitness, the overcompensation sounds plausible.

Table 11 – Dunn's test result for the normalized sparsity of enemies in the levels generated for each quest generator's input.

| Enemy Sparsity | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A1C3I4M2** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A1C4I3M2** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A2C1I3M4** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A2C1I4M3** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A2C4I1M3** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A3C2I1M4** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A3C2I4M1** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A3C4I2M1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 |
| **A4C1I3M2** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 |
| **A4C2I1M3** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 |
| **A4C3I2M1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 |

Table 12 – Dunn's test result for the normalized standard deviation of enemies' position in the levels generated for each quest generator's input.

| Enemy Std. Dev. | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A1C3I2M4 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.031 | 1.000 | 1.000 | 0.372 |
| A1C3I4M2 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.207 | 1.000 | 1.000 | 1.000 |
| A1C4I3M2 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.071 | 1.000 | 0.021 | 1.000 | 1.000 | 0.098 | 1.000 |
| A2C1I3M4 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.023 | 1.000 | 0.065 | 1.000 | 1.000 | 0.032 | 1.000 |
| A2C1I4M3 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.430 | 1.000 | 0.002 | 1.000 | 1.000 | 0.557 | 1.000 |
| A2C4I1M3 | 1.000 | 1.000 | 0.071 | 0.023 | 0.430 | 1.000 | 1.000 | 0.000 | 0.000 | 0.170 | 1.000 | 0.004 |
| A3C2I1M4 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 0.058 | 1.000 | 1.000 | 0.585 |
| A3C2I4M1 | 0.000 | 0.000 | 0.021 | 0.065 | 0.002 | 0.000 | 0.000 | 1.000 | 1.000 | 0.007 | 0.000 | 0.256 |
| A3C4I2M1 | 0.031 | 0.207 | 1.000 | 1.000 | 1.000 | 0.000 | 0.058 | 1.000 | 1.000 | 1.000 | 0.000 | 1.000 |
| A4C1I3M2 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.170 | 1.000 | 0.007 | 1.000 | 1.000 | 0.223 | 1.000 |
| A4C2I1M3 | 1.000 | 1.000 | 0.098 | 0.032 | 0.557 | 1.000 | 1.000 | 0.000 | 0.000 | 0.223 | 1.000 | 0.006 |
| A4C3I2M1 | 0.372 | 1.000 | 1.000 | 1.000 | 1.000 | 0.004 | 0.585 | 0.256 | 1.000 | 1.000 | 0.006 | 1.000 |

Table 13 – Dunn's test result for the resulting fitness of the levels generated for each quest generator's input.

| Fitness | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A1C3I2M4 | 1.000 | 1.000 | 1.000 | 1.000 | 0.208 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.208 | 1.000 |
| A1C3I4M2 | 1.000 | 1.000 | 1.000 | 1.000 | 0.655 | 1.000 | 1.000 | 0.466 | 1.000 | 1.000 | 0.655 | 1.000 |
| A1C4I3M2 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.153 | 0.581 | 1.000 | 1.000 | 1.000 |
| A2C1I3M4 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.003 | 0.021 | 1.000 | 1.000 | 0.335 |
| A2C1I4M3 | 0.208 | 0.655 | 1.000 | 1.000 | 1.000 | 1.000 | 0.153 | 0.000 | 0.000 | 0.418 | 1.000 | 0.014 |
| A2C4I1M3 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.003 | 0.023 | 1.000 | 1.000 | 0.358 |
| A3C2I1M4 | 1.000 | 1.000 | 1.000 | 1.000 | 0.153 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.153 | 1.000 |
| A3C2I4M1 | 1.000 | 0.466 | 0.153 | 0.003 | 0.000 | 0.003 | 1.000 | 1.000 | 1.000 | 0.720 | 0.000 | 1.000 |
| A3C4I2M1 | 1.000 | 1.000 | 0.581 | 0.021 | 0.000 | 0.023 | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 | 1.000 |
| A4C1I3M2 | 1.000 | 1.000 | 1.000 | 1.000 | 0.418 | 1.000 | 1.000 | 0.720 | 1.000 | 1.000 | 0.418 | 1.000 |
| A4C2I1M3 | 0.208 | 0.655 | 1.000 | 1.000 | 1.000 | 1.000 | 0.153 | 0.000 | 0.000 | 0.418 | 1.000 | 0.014 |
| A4C3I2M1 | 1.000 | 1.000 | 1.000 | 0.335 | 0.014 | 0.358 | 1.000 | 1.000 | 1.000 | 1.000 | 0.014 | 1.000 |

Continuing the discussion on the aforementioned Table 12, comparing the standard deviation of enemies, we note that specially levels with lower mastery were considered different from the others. This is possibly because a lower mastery results in less enemies and, therefore, they are harder to distribute equally through the dungeon. This hypothesis is corroborated when we check that the levels not discarded as different from the *A3C2I4M1* are the A2C1I3M4 and A3C4I2M1 and A4C3I2M1, the last ones being the ones with lower mastery, and the first one the level with high mastery but low creativity inputs. This means that this level has numerous enemies to distribute in a few rooms, and may also have difficulty to converge to a good setting.

Summarizing the discussion on the differences of the fitness-related metrics from the dungeons generated by the quests, Table 13 shows that dungeons with the lowest mastery were the ones that had most differences from others. They were considered different from some levels with 3 and 4 as input to mastery, although no clear trend on the other inputs can be seen as to a cause for the difference. Considering the previous tables, this difference arose, possibly, by the difficulty to converge to the optimal sparsity and standard deviation, as discussed for Table 11 and 12.

Now, we present the tables regarding the comparison of the elements in the generated dungeon. Table 14 shows the comparison between the number of rooms in the dungeons for each input. As the number of non-immersion quests and the creativity value increases the number of rooms, and the first one is stochastic, most of the configurations had differences between themselves. And the ones that could not be discarded as similar, have no clear pattern.

Table 14 – Dunn's test result for the total rooms in the levels generated for each quest generator's input.

| # of Rooms | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.446 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A1C3I4M2** | 0.000 | 1.000 | 0.802 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 | 0.055 | 1.000 | 0.000 | 0.000 |
| **A1C4I3M2** | 0.000 | 0.802 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 |
| **A2C1I3M4** | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 |
| **A2C1I4M3** | 0.000 | 0.002 | 0.000 | 0.000 | 1.000 | 0.041 | 0.001 | 0.000 | 0.000 | 0.001 | 1.000 | 0.000 |
| **A2C4I1M3** | 0.446 | 0.000 | 0.000 | 1.000 | 0.041 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.177 | 0.000 |
| **A3C2I1M4** | 1.000 | 0.000 | 0.000 | 1.000 | 0.001 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.005 | 0.000 |
| **A3C2I4M1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.014 | 0.000 | 0.000 | 1.000 |
| **A3C4I2M1** | 0.000 | 0.055 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.014 | 1.000 | 0.117 | 0.000 | 0.005 |
| **A4C1I3M2** | 0.000 | 1.000 | 1.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.117 | 1.000 | 0.000 | 0.000 |
| **A4C2I1M3** | 0.000 | 0.000 | 0.000 | 0.002 | 1.000 | 0.177 | 0.005 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 |
| **A4C3I2M1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.005 | 0.000 | 0.000 | 1.000 |

However, for the number of locks and keys, that should be similar, as the dungeon generator tries to create pairs, are shown in tables 15 and 16. Both are very similar, but have small differences, as the dungeon generator allows the existence of extra keys in a dungeon. As the number of keys and locks is mainly defined by the creativity, but capped if the number of rooms is too small, we see in both tables that most of the cells where the population could not be rejected as being the same are the ones with the same or similar creativity values. With this observation, we can see that the quest generator is passing these inputs to the dungeon generator that are aligned to the intended design. This same pattern and discussion is valid for the resulting linearity of levels, shown in Table 17, although the linearity itself is directly related to the creativity and achievement inputs.

Table 15 – Dunn's test result for the total keys in the levels generated for each quest generator's input.

| # of Keys | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 1.000 | 0.028 | 0.000 | 0.000 | 0.040 | 0.708 | 1.000 | 0.010 | 0.000 | 1.000 | 0.050 |
| **A1C3I4M2** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.003 | 1.000 | 0.001 |
| **A1C4I3M2** | 0.028 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.005 | 1.000 | 0.000 | 0.000 | 1.000 |
| **A2C1I3M4** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.177 | 0.000 | 0.000 |
| **A2C1I4M3** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.244 | 0.000 | 0.000 |
| **A2C4I1M3** | 0.040 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.007 | 1.000 | 0.000 | 0.000 | 1.000 |
| **A3C2I1M4** | 0.708 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.050 | 1.000 | 0.000 |
| **A3C2I4M1** | 1.000 | 1.000 | 0.005 | 0.000 | 0.000 | 0.007 | 1.000 | 1.000 | 0.002 | 0.000 | 1.000 | 0.010 |
| **A3C4I2M1** | 0.010 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.002 | 1.000 | 0.000 | 0.000 | 1.000 |
| **A4C1I3M2** | 0.000 | 0.003 | 0.000 | 0.177 | 0.244 | 0.000 | 0.050 | 0.000 | 0.000 | 1.000 | 0.005 | 0.000 |
| **A4C2I1M3** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.005 | 1.000 | 0.000 |
| **A4C3I2M1** | 0.050 | 0.001 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.010 | 1.000 | 0.000 | 0.000 | 1.000 |

When analyzing the enemies per dungeon in Table 18 we see a similar pattern to the one comparing the distance from the inputs (Table 10), however, this seems to be just a coincidence.

Table 16 – Dunn's test result for the total locks in the levels generated for each quest generator's input.

| # of Locks | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A1C3I4M2** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A1C4I3M2** | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| **A2C1I3M4** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| **A2C1I4M3** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| **A2C4I1M3** | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| **A3C2I1M4** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A3C2I4M1** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A3C4I2M1** | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| **A4C1I3M2** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| **A4C2I1M3** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A4C3I2M1** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |

Table 17 – Dunn's test result for the linearity in the levels generated for each quest generator's input.

| Linearity | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A1C3I4M2** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A1C4I3M2** | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| **A2C1I3M4** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| **A2C1I4M3** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| **A2C4I1M3** | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| **A3C2I1M4** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A3C2I4M1** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A3C4I2M1** | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| **A4C1I3M2** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| **A4C2I1M3** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A4C3I2M1** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |

As this value is heavily dependent not only in the mastery value, but the random value chosen for the number of enemies for each quest, it is expected that the number of enemies would not necessarily reflect the value of mastery. But this is not a problem for the player by itself, as the mastery also influences the difficulty of the enemies besides their amount. Nonetheless, making the range of enemies dependent on the mastery value could make their amount more different and possibly result in a greater variety of content for different values of mastery.

Table 18 – Dunn's test result for the total enemies in the levels generated for each quest generator's input.

| # of Enemies | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 1.000 | 1.000 | 0.000 | 0.001 | 1.000 | 1.000 | 1.000 | 0.914 | 1.000 | 1.000 | 1.000 |
| **A1C3I4M2** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A1C4I3M2** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A2C1I3M4** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A2C1I4M3** | 0.001 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A2C4I1M3** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.796 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A3C2I1M4** | 1.000 | 1.000 | 1.000 | 0.000 | 0.003 | 0.796 | 1.000 | 1.000 | 0.543 | 1.000 | 1.000 | 1.000 |
| **A3C2I4M1** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A3C4I2M1** | 0.914 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.543 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C1I3M2** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C2I1M3** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C3I2M1** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

For the final part of this evaluation, we check the number of quests created, shown in Table 19, and then analyze the number of quests generated by each non-terminal of the grammar in tables 20 to 23. Table 19 shows that for most combinations, the number of quests was different, majorly due to the stochastic nature of the quest generator. The same can be observed for the achievement and creativity quests, where both tables have the same pattern, with the majority having different numbers of the given quests (tables 20 and 21). For the immersion and mastery, most of them were considered similar, with, in both cases, the inputs *A2C1I3M4* and *A2C1I4M3* being, in most cases for the mastery and all for the immersion, different from others. This is possibly due to the randomness of the algorithm. Overall, these were expected results. But, further tweaking of probabilities may result in more different numbers, if necessary.

Table 19 – Dunn's test result for the total quests generated for each quest generator's input.

| # of Quests | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A1C3I2M4 | 1.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.796 | 0.000 | 0.000 | 1.000 |
| A1C3I4M2 | 0.000 | 1.000 | 0.000 | 0.078 | 0.054 | 0.000 | 1.000 | 0.320 | 0.000 | 0.078 | 0.036 | 0.000 |
| A1C4I3M2 | 1.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 |
| A2C1I3M4 | 0.000 | 0.078 | 0.000 | 1.000 | 1.000 | 0.000 | 0.009 | 1.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| A2C1I4M3 | 0.000 | 0.054 | 0.000 | 1.000 | 1.000 | 0.000 | 0.005 | 1.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| A2C4I1M3 | 1.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.433 |
| A3C2I1M4 | 0.000 | 1.000 | 0.000 | 0.009 | 0.005 | 0.000 | 1.000 | 0.054 | 0.000 | 0.009 | 0.245 | 0.000 |
| A3C2I4M1 | 0.000 | 0.320 | 0.000 | 1.000 | 1.000 | 0.000 | 0.054 | 1.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| A3C4I2M1 | 0.796 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.245 |
| A4C1I3M2 | 0.000 | 0.078 | 0.000 | 1.000 | 1.000 | 0.000 | 0.009 | 1.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| A4C2I1M3 | 0.000 | 0.036 | 0.000 | 0.000 | 0.000 | 0.000 | 0.245 | 0.000 | 0.000 | 0.000 | 1.000 | 0.003 |
| A4C3I2M1 | 1.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.433 | 0.000 | 0.000 | 0.245 | 0.000 | 0.003 | 1.000 |

Table 20 – Dunn's test result for the number of achievement quests generated for each quest generator's input.

| # of A Quests | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A1C3I2M4 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| A1C3I4M2 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| A1C4I3M2 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| A2C1I3M4 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| A2C1I4M3 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| A2C4I1M3 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| A3C2I1M4 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| A3C2I4M1 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| A3C4I2M1 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| A4C1I3M2 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| A4C2I1M3 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| A4C3I2M1 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |

Overall, the computational results over the quest generator and the content they create were positive, with the contents varying for different inputs, and most of the results behaving as expected. As the results with players, in Section 5.3.3 shows, they were considered different enough and tweaking the parameters may help futures tests, but this setting was enough to show difference to players, even if statistically it may not always be discarded as the same.

Table 21 – Dunn's test result for the number of creativity quests generated for each quest generator's input.

| # of C Quests | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A1C3I4M2** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A1C4I3M2** | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| **A2C1I3M4** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| **A2C1I4M3** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| **A2C4I1M3** | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| **A3C2I1M4** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A3C2I4M1** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A3C4I2M1** | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 |
| **A4C1I3M2** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| **A4C2I1M3** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |
| **A4C3I2M1** | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 |

Table 22 – Dunn's test result for the number of immersion quests generated for each quest generator's input.

| # of I Quests | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A1C3I4M2** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A1C4I3M2** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A2C1I3M4** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A2C1I4M3** | 0.000 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **A2C4I1M3** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A3C2I1M4** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A3C2I4M1** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A3C4I2M1** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C1I3M2** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C2I1M3** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C3I2M1** | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

Table 23 – Dunn's test result for the number of mastery quests generated for each quest generator's input.

| # of M Quests | 1324 | 1342 | 1432 | 2134 | 2143 | 2413 | 3214 | 3241 | 3421 | 4132 | 4213 | 4321 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A1C3I2M4** | 1.000 | 1.000 | 1.000 | 0.124 | 0.002 | 1.000 | 1.000 | 0.002 | 1.000 | 1.000 | 1.000 | 0.403 |
| **A1C3I4M2** | 1.000 | 1.000 | 1.000 | 1.000 | 0.147 | 1.000 | 1.000 | 0.114 | 0.186 | 1.000 | 1.000 | 0.010 |
| **A1C4I3M2** | 1.000 | 1.000 | 1.000 | 1.000 | 0.047 | 1.000 | 1.000 | 0.036 | 0.477 | 1.000 | 1.000 | 0.036 |
| **A2C1I3M4** | 0.124 | 1.000 | 1.000 | 1.000 | 1.000 | 0.010 | 0.016 | 1.000 | 0.000 | 0.183 | 0.285 | 0.000 |
| **A2C1I4M3** | 0.002 | 0.147 | 0.047 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.004 | 0.007 | 0.000 |
| **A2C4I1M3** | 1.000 | 1.000 | 1.000 | 0.010 | 0.000 | 1.000 | 1.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A3C2I1M4** | 1.000 | 1.000 | 1.000 | 0.016 | 0.000 | 1.000 | 1.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A3C2I4M1** | 0.002 | 0.114 | 0.036 | 1.000 | 1.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.003 | 0.005 | 0.000 |
| **A3C4I2M1** | 1.000 | 0.186 | 0.477 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| **A4C1I3M2** | 1.000 | 1.000 | 1.000 | 0.183 | 0.004 | 1.000 | 1.000 | 0.003 | 1.000 | 1.000 | 1.000 | 0.285 |
| **A4C2I1M3** | 1.000 | 1.000 | 1.000 | 0.285 | 0.007 | 1.000 | 1.000 | 0.005 | 1.000 | 1.000 | 1.000 | 0.183 |
| **A4C3I2M1** | 0.403 | 0.010 | 0.036 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 1.000 | 0.285 | 0.183 | 1.000 |

### 5.3.3 Player Feedback from Online Generation

We collected data from ten anonymous players who played up to the 12 levels we generated with the real-time generation setting. We collected data from 119 playthroughs in different dungeons. Some players only played through some dungeons, while others played more than once in a single dungeon, as they could replay when defeated by the enemies. When

Figure 78 – Post-test questionnaire for the online experiment, answered after completing (or giving up) each level.

Q1 The level was fun to play;

Q2 The level was difficult to complete;

Q3 The challenge was just right;

Q4 The enemies were difficult to defeat;

Q5 It was difficult to find the exit/goal of this level;

Q6 I liked the size of the dungeon;

Q7 I liked the amount of exploration available on this level;

Q8 I liked the challenge of finding the keys to this level;

Q9 The rewards were on the proper amount;

Q10 I had fun completing the quests;

Q11 The quests given to me were designed by a human;

discarding the data where players did not finish the dungeon, or did not answer all post-test questions, we collected 71 answers for the post-test questionnaire. First, we present a visual analysis of how each player's preferences affected their post-test answers in Section 5.3.3.1. Then, we conduct a statistical analysis, in Section 5.3.3.2.

The ten players identified themselves in the Pre-test Questionnaire as having the profiles shown in Table 24. We may observe that no player scored 4 on the creativity value, but all other combinations of parameters were contemplated.

Table 24 – Total players by profile according to their answers on the pre-test questionnaire during the experiment with the online generators.

| **Profile** | **A1C3I2M4** | **A2C1I3M4** | **A2C1I4M3** | **A3C2I1M4** | **A3C2I4M1** | **A4C1I3M2** | **A4C2I1M3** | **A4C3I2M1** |
|---|---|---|---|---|---|---|---|---|
| **Players** | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

After completing or giving up playing each dungeon, the player had to answer a Post-Test Questionnaire, similar to our experiment with the Offline setup. The questionnaire, shown in 78 had 11 questions, most remaining from the offline experiment, but with new ones, especially focused on the quest elements.

#### 5.3.3.1 Visual Analysis

Figure 79 shows that players enjoyed the levels, with most answers to the questions about having "Fun" playing the game having an average of 3.6, as shown in the rightmost bottom cell. We can also observe that our system generated diversity between the inputs, as the opinions varied for each dungeon. The only level evaluated as having an average amount of fun was the one with inputs *A=2, C=1, I=4, M=3*, possibly because it combined two contrasting profiles: immersion and mastery. We can also observe that dungeon *A=2, C=1, I=3, M=4* were the best evaluated, possibly because it were evaluated by two players who had a weight of 4 in mastery, and the other one, although having a score of 2 in mastery, had a score of 4 in achievement. Therefore, all players had a profile that tend to like challenge.

Figure 79 – Heatmap showing the mean value of each profile's answer to Q1 of the post-test questionnaire: how much fun they felt playing each dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.

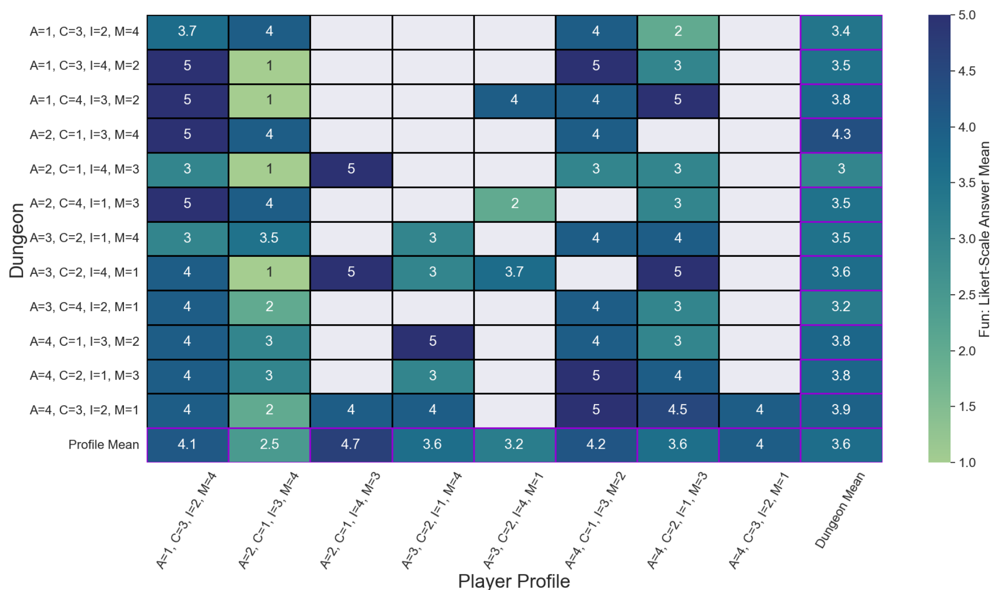
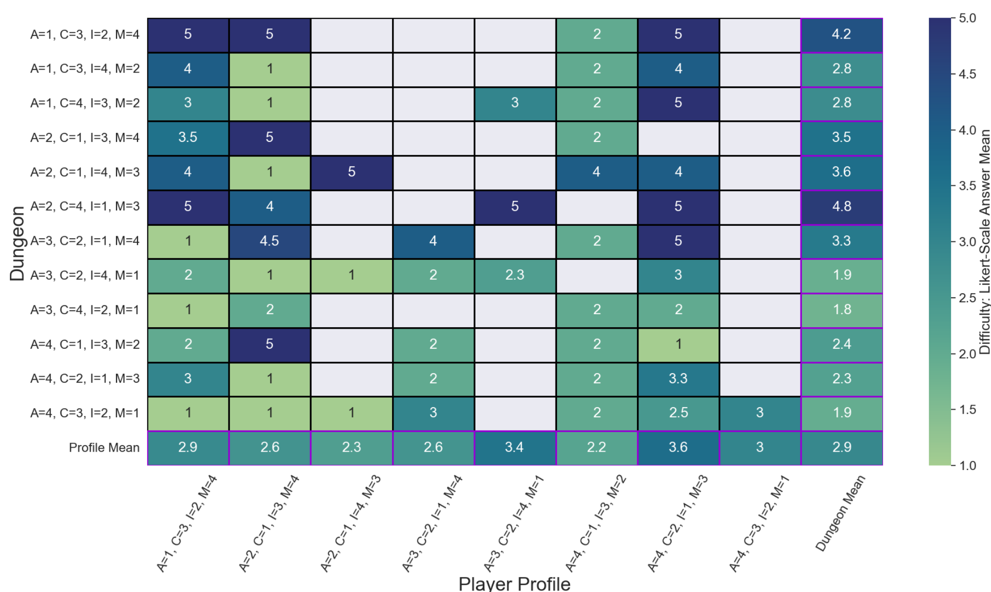



Figure 80 shows that the mastery setting was a significant factor in defining the difficulty of the levels, as expected. The creativity also had such relevance once it was weighted in the size of the dungeon. All dungeons with a mean value above 3 had great mastery and/or creativity inputs (rightmost column). We can also notice that dungeons with a mean lesser than 2 were all created with an input of 1 to mastery. These findings confirm our observations from the previous experiment, in Section 5.2, that large dungeons are difficult, mainly when the enemies are also at a harder level of difficulty.

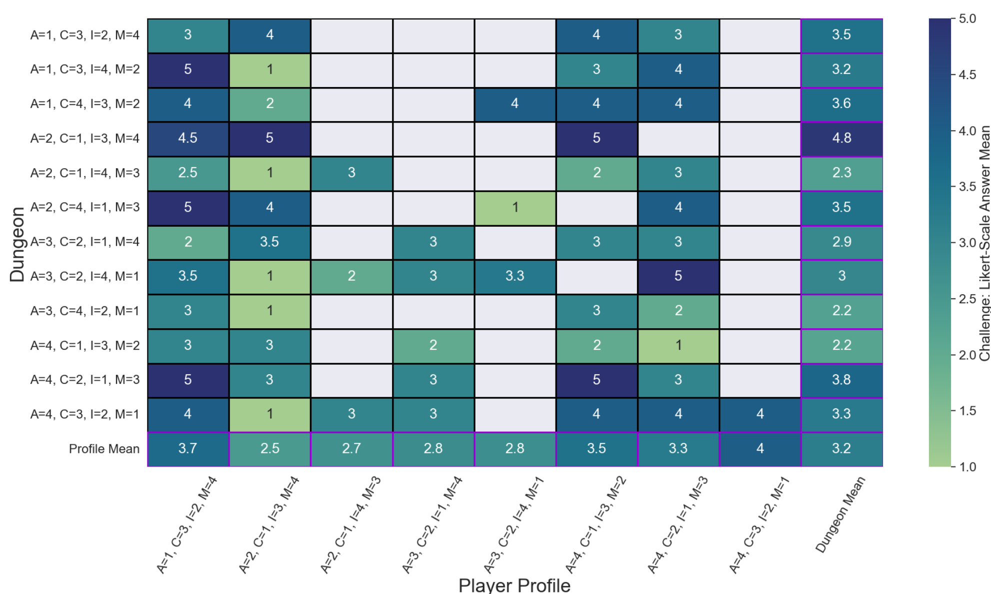
Overall, the system created challenging levels for almost all combinations, as seen in Figure 81, where the overall mean was 3.2 (rightmost bottom cell). Only three levels had a mean challenge below 3: the first one is the *A=3, C=4, I=2, M=1*, where the large dungeon with easy enemies may have seen as tedious for our players, more focused on the mastery aspect of games. The other is the *A=2, C=1, I=4, M=3*, the one with high immersion and mastery, which was

Figure 80 – Heatmap showing the mean value of each profile's answer to Q2 of the post-test questionnaire: how difficult was the dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.




already considered not very fun to play in Figure 79. And the last one is the *A=4, C=1, I=3, M=2*, seen as having easier enemies, which may have been the reason for the low score on the challenge aspect. A final observation is that the same level considered as the most fun, in Figure 79, was also the most challenging: *A=2, C=1, I=3, M=4*. This reinforces many game design theories that the right amount of challenge is perceived as the most fun for players.

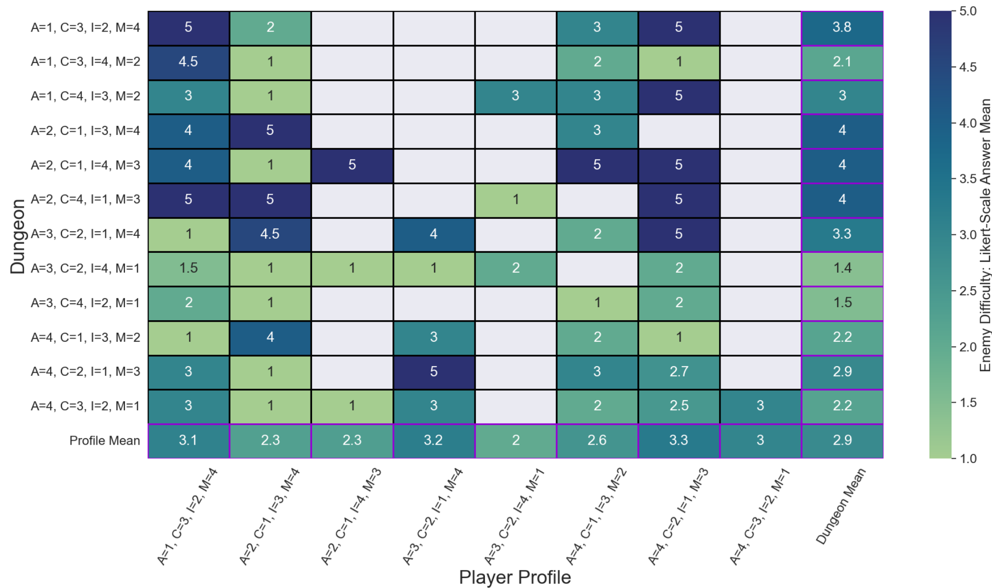
Figure 81 – Heatmap showing the mean value of each profile's answer to Q3 of the post-test questionnaire: how much challenge they felt playing each dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.




The enemy difficulty, in Figure 82, behaved as expected: dungeons with a mastery setting

3 or 4 were considered more difficult, especially when paired with weights providing larger dungeons, as seen in the mean values above 3 in the rightmost column, containing the mean for each dungeon. On the other hand, those created with the mastery set to 1 or 2 were considered easier, some with a mean value below 2.

Figure 82 – Heatmap showing the mean value of each profile's answer to Q4 of the post-test questionnaire: how difficult were the enemies in each dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.




We analyze exploration-related answers in figures 83 through 86. In Figure 83 we observe that dungeons with higher creativity inputs were evaluated as harder to navigate (mean value equal or greater than 3 on the rightmost column), except for the one with input *A=1, C=4, I=3, M=2* and *A=4, C=3, I=2, M=1*. These exceptions are due to their enemies being easier, so the player could explore the dungeon more freely. Another exception is the dungeon with input *A=3, C=2, I=1, M=4*, following the same reasoning: the enemies were harder, making it difficult for the player to explore.

Nonetheless, Figure 84 shows that all dungeons were considered of adequate size (mean value greater than 3, in the rightmost column), even if varying in the number of enemies, items, NPCs, keys, locks, and linearity. Users from all backgrounds answered positively, on average, to the dungeons. With two notable lower scores coming from players. The first one had a profile *A=2, C=1, I=3, M=4* and, when evaluating the dungeon *A=1, C=4, I=3, M=2*, gave it a score of 2. This is probably because the dungeon had a much more exploration-focused setup than what the player wanted. The second case was the evaluation of dungeon *A=2, C=1, I=4, M=3* by player *A=4, C=2, I=1, M=3*. In this case, the difference in the achievement and immersion factors may have made the dungeon with more talk-related quests, which the player did not favor, and an easy challenge to find the keys, as reported by the same player in Figure 83.

The same positive results are valid for the amount of exploration, in Figure 85: all

Figure 83 – Heatmap showing the mean value of each profile's answer to Q5 of the post-test questionnaire: how difficult was to navigate through each dungeon. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.

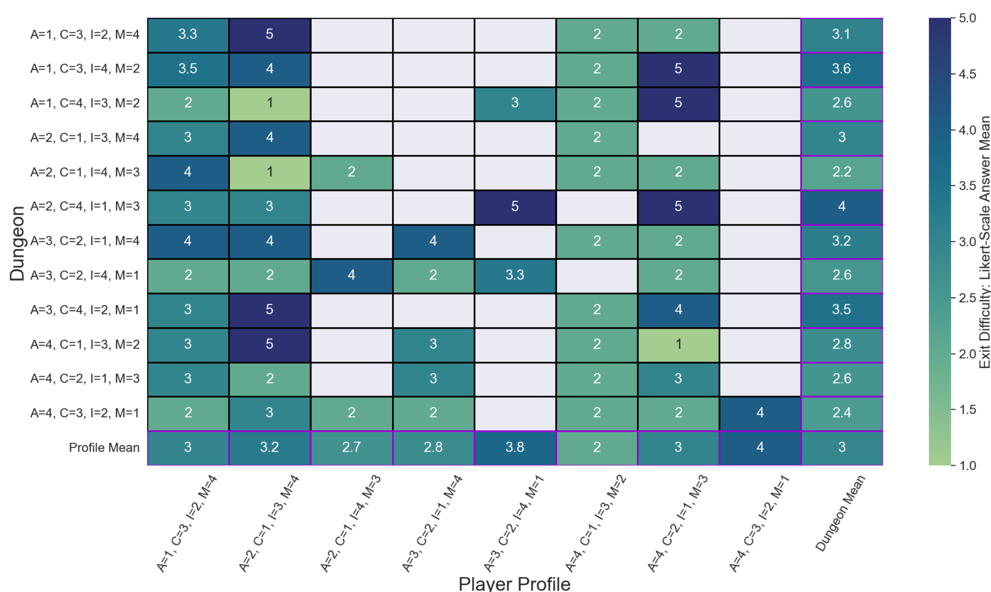
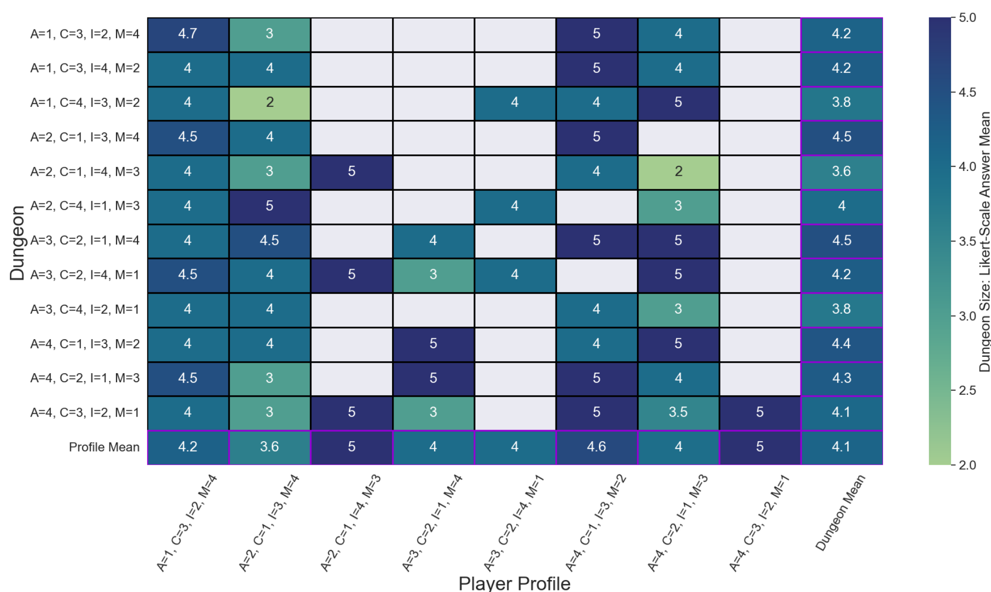



Figure 84 – Heatmap showing the mean value of each profile's answer to Q6 of the post-test questionnaire: how good they felt was the dungeon size. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.




dungeons were positively evaluated by the players, with just *A=2, C=1, I=4, M=3* having a neutral mean (rightmost column), possibly due to being a small dungeon, with few keys, but hard enemies. Finally, for the challenge in the locked-door puzzles, in Figure 86, most levels were also positively evaluated, except for *A=1, C=3, I=4, M=2*, with its mean lower than 3, possibly for being a somewhat large dungeon, linear, with few keys and somewhat easy enemies. These parameters usually create dungeons that are easy to find all the keys and traverse without dying,

meaning it's not very challenging.

Figure 85 – Heatmap showing the mean value of each profile's answer to Q7 of the post-test questionnaire: how good they felt was the dungeon's exploration. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.

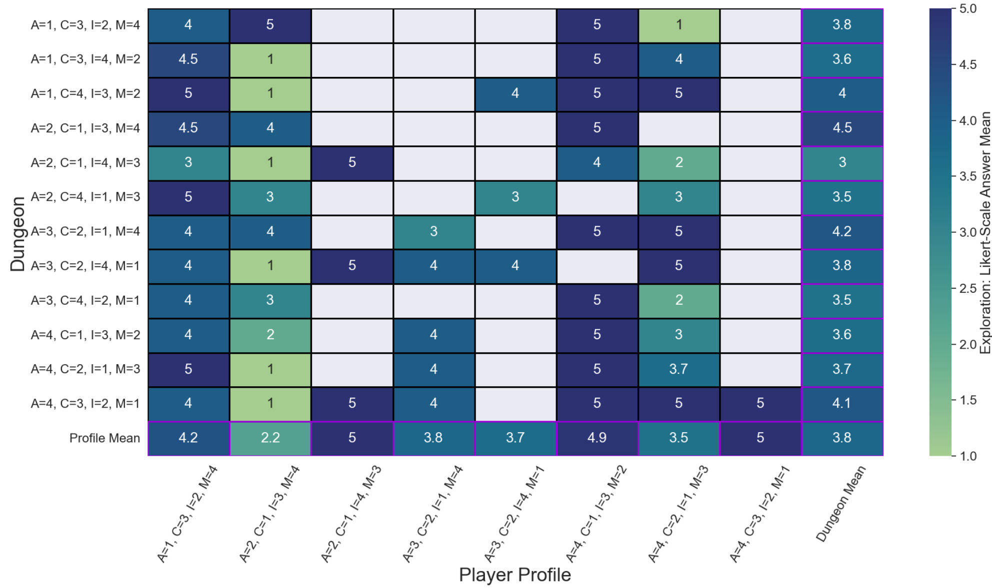



Figure 86 – Heatmap showing the mean value of each profile's answer to Q8 of the post-test questionnaire: how good they felt was the challenge to open the locked doors. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.

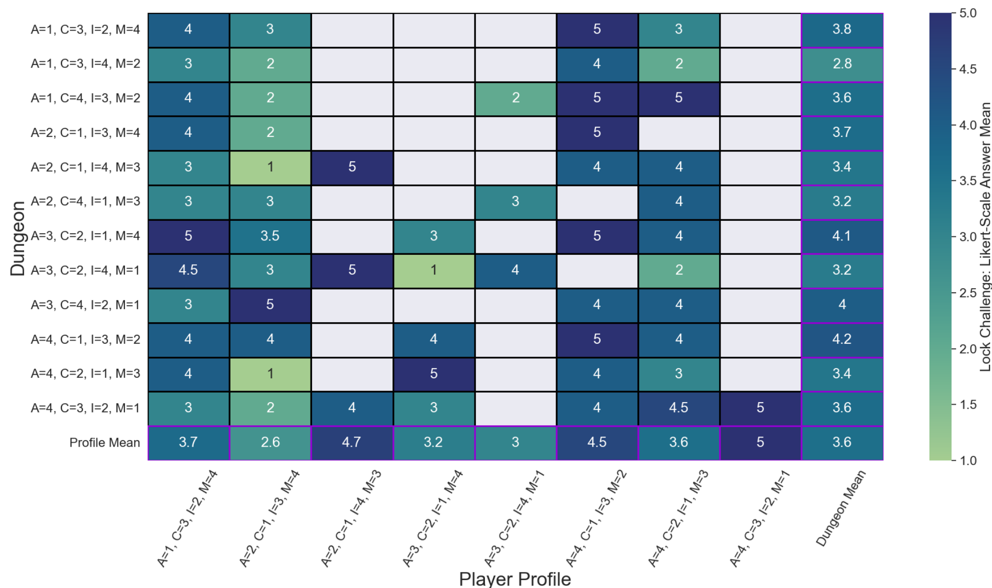
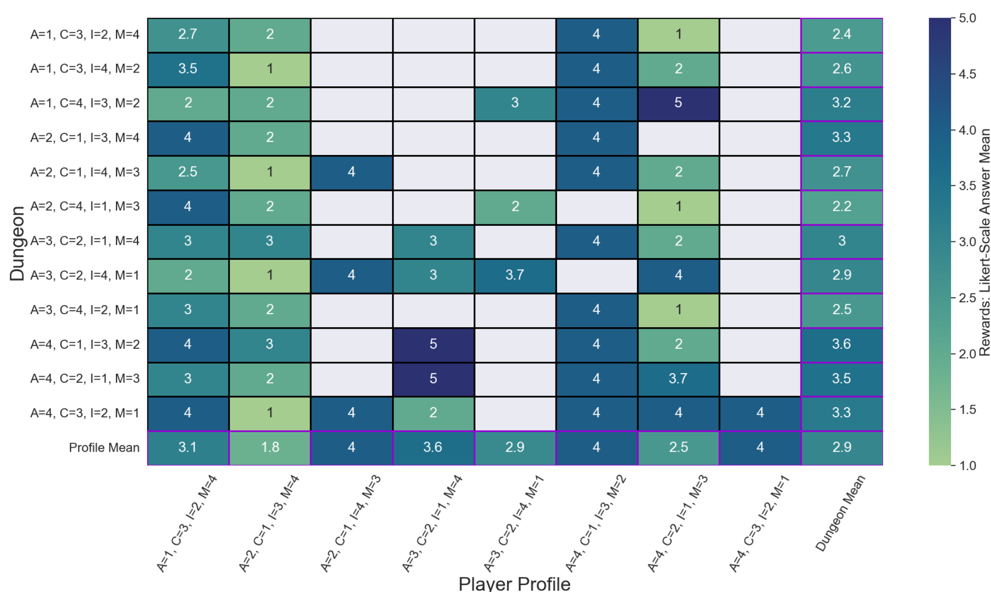



Figure 87 shows the user's evaluations of the items (rewards).They were received as neutral, whereas dungeons with higher achievement and smaller creativity inputs were slightly more positive than others. Possibly because they provided more fetch quests, giving items to players, in a smaller environment, increasing the density of items found per room traveled.

Figure 87 – Heatmap showing the mean value of each profile's answer to Q9 of the post-test questionnaire: how good they felt was the rewards (items) collected. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.




The players also did not have much fun with the quests, as shown in Figure 88, but there were not many negative evaluations. This feedback means we must make the implementation of the quests more interesting, with better rewards and more engaging dialogues. The players also had a neutral opinion if humans generated the quests, in Figure 89. This means that players could not discard the content as human-made, with results similar to those in (LIMA; FEIJó; FURTADO, 2019). Although, we reckon the dialogues were created from grammar, changing only the names of contents and amounts, making it much more believable as to be made by humans.

From the analysis of players' answers to the post-test questionnaire, we can conclude that our real-time generation led to positive results. The content was well evaluated compared to the experiments with the offline setup. The orchestration of the multiple contents was successful, as players enjoyed both the levels, the enemies, and the quests (although more neutral towards the latter). They perceived difficulty and challenge in order to the inputs provided to the quest generator.

However, we will also peek into gameplay data to further evaluate the quality of our content. We present a series of heatmap plots, grouping the data for players with different profiles for each dungeon and showing their performance against other metrics, in Figures 90 through 96. Although we had only one post-test questionnaire answer per profile for each dungeon, the gameplay data was collected at every attempt, disregarding if the player won or lost, in a total of 119 attempts.

First, we analyze the success rate for each player type in each dungeon, presented in Fig. 90. As expected, the dungeons with higher mastery input had lower success rates than

Figure 88 – Heatmap showing the mean value of each profile's answer to Q10 of the post-test questionnaire: how much fun they had completing the quests. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.

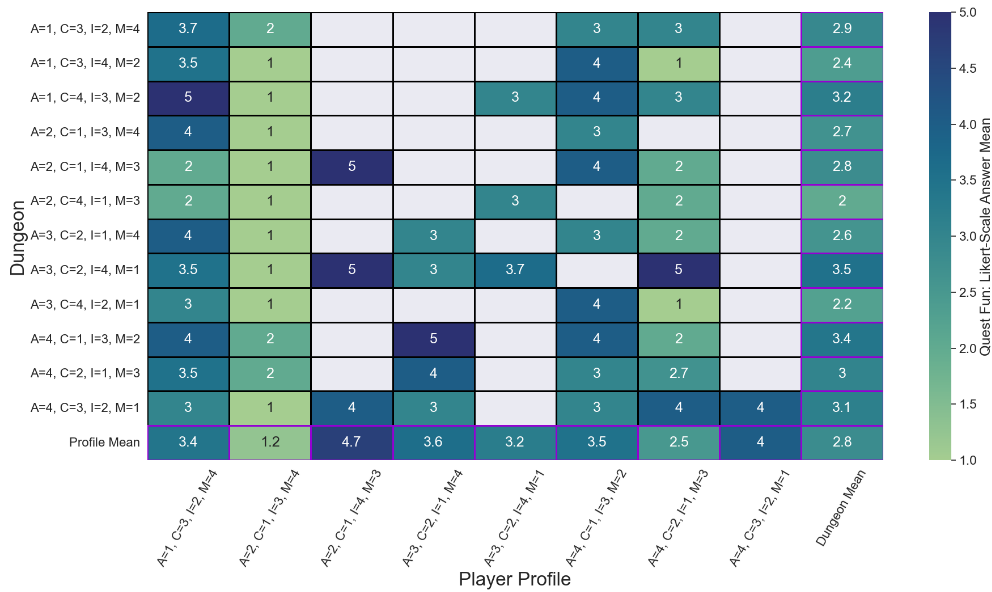
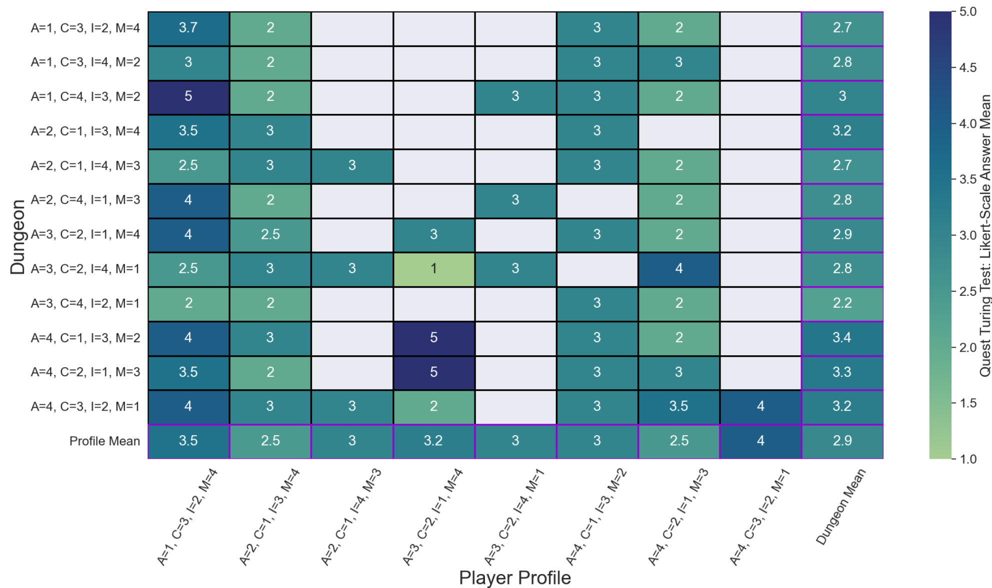



Figure 89 – Heatmap showing the mean value of each profile's answer to Q11 of the post-test questionnaire: how much they felt the quests were designed by humans. The bottom row shows the mean per profile, and the rightmost column the mean per dungeon.




others, and players who favored mastery did not always have the best results, but those with high achievement preferences went better. By looking at Fig. 91 we see that for the hardest dungeons, players killed fewer enemies. Some must have run away from them, while others died, as the previous figure showed the decrease in success rate in these levels. When comparing profiles, especially in dungeons *A=2, C=1, I=4, M=3* and *A=2, C=4, I=1, M=3* which had more attempts, we that players who favored mastery killed slightly more enemies than those who didn't.

Figure 90 – Heatmap plot, grouping the success rate for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator.

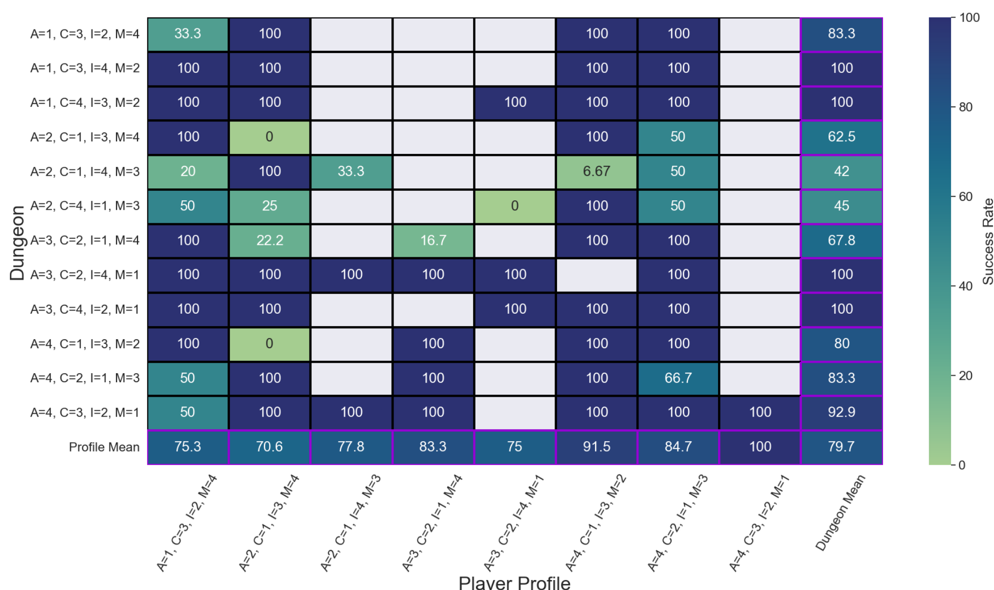
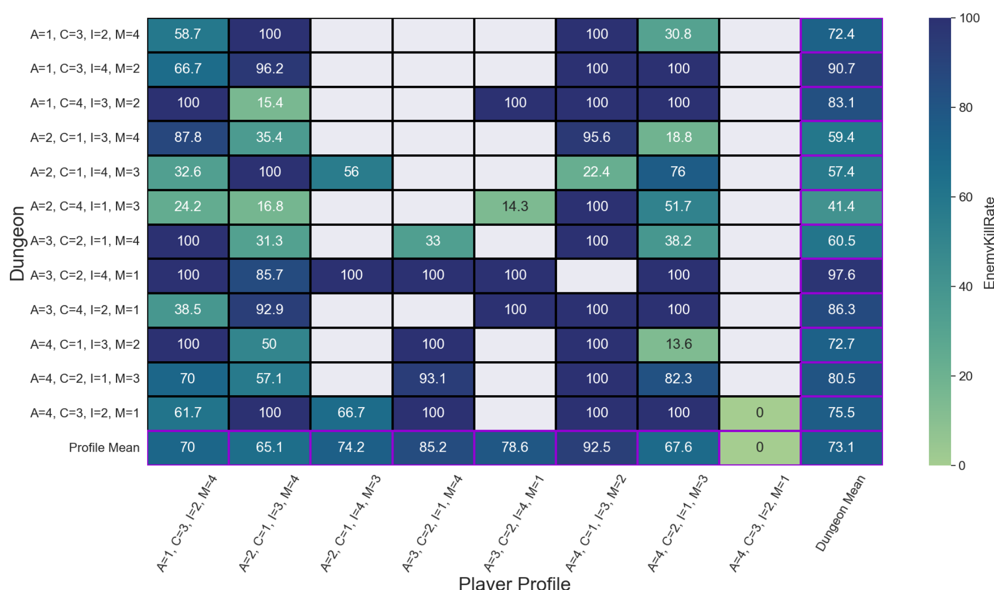


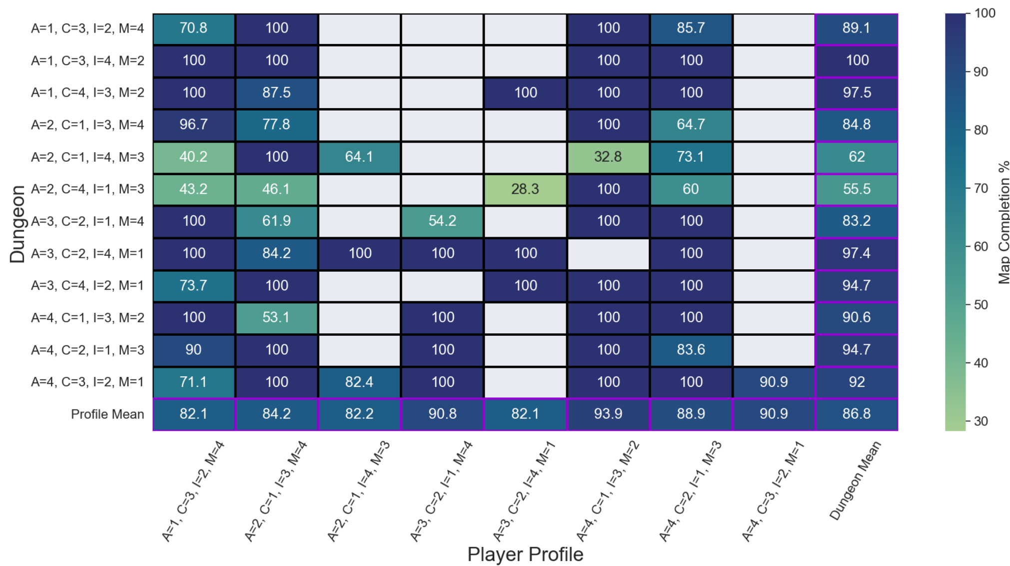

Figure 91 – Heatmap plot, grouping the percentage of enemies killed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator.




We analyze Figure 92 related to the map completion (unique rooms visited divided by the number of rooms in the dungeon). Their mastery input was a significant factor in their exploration: players died more before exploring much. As the enemies were a significant barrier to the exploration, and most players did not favor the creativity aspect in their profiles, we cannot conclude their preferences influenced the map's completion. But dungeons with higher mastery had a smaller completion rate.

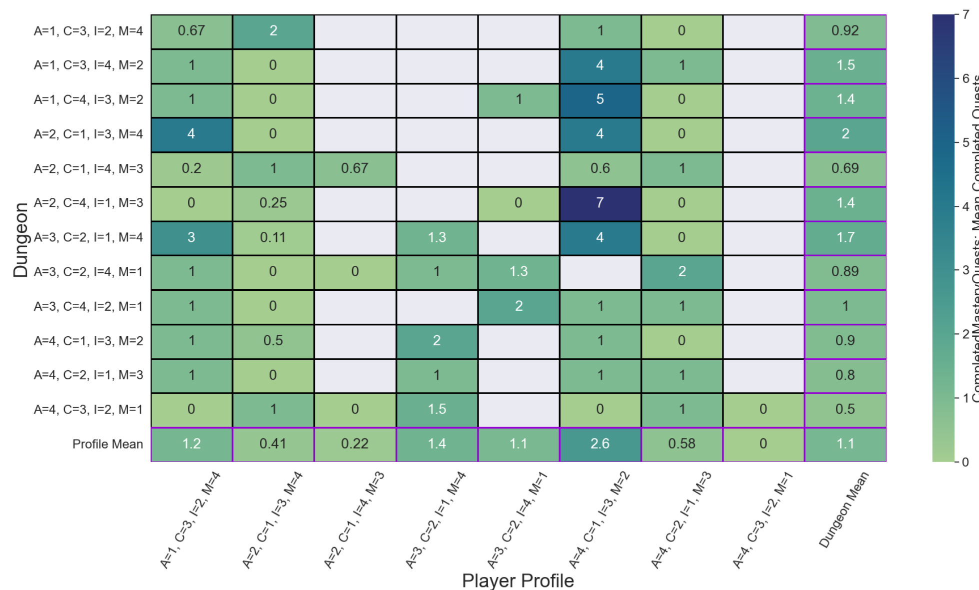
Now, we analyze how many quests of each type (that is, their non-terminal value) each player's group completed. First, we analyze the number of completed mastery quests, in Figure

Figure 92 – Heatmap plot, grouping the percentage of map completion for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator.




93. We see that for the hardest dungeons (high mastery input), players with profiles focused both on mastery and achievement completed more quests: in dungeons where players with an achievement profile of 3 or 4 played, we see that they completed almost as many quests as those with high mastery preferences, and more than those with lower preferences on both. Specially, two profiles completed the most quests: *A=1, C=3, I=2, M=4* and *A=4, C=1, I=3, M=2*, one with high mastery and the other with high achievement. Moreover, most dungeons with higher mastery input had more mastery quests completed than those with lower inputs, specially because the greater the input, the higher the chance to have more mastery quests.

Figure 93 – Heatmap plot, grouping the percentage of mastery quests completed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator.



When analyzing the completion of achievement quests, in Figure 94, we see a similar tendency: players with greater achievement and mastery preferences completed more achievement quests. Specially those with high preferences on both, except for profile *A=4, C=2, I=1, M=3*.

Figure 94 – Heatmap plot, grouping the percentage of achievement quests completed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator.

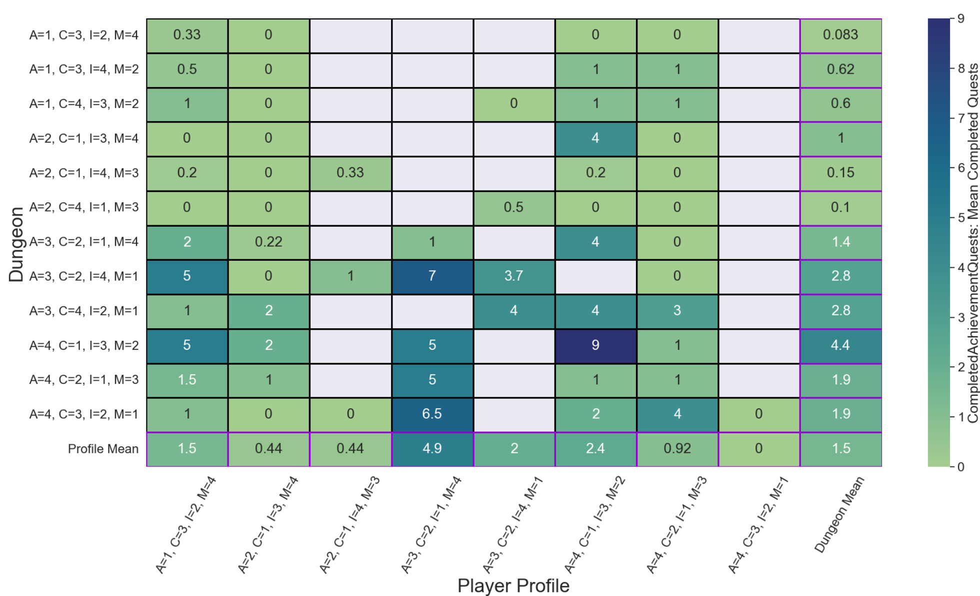



Figure 95 shows the players' completion of the creativity quests. We see that players with higher values for achievement also completed more quests of this type, but one of the profiles with high creativity and mastery, but lowest achievement weight also completed many quests, specially in one of the dungeons with the highest creativity input. This is a clue that our system can create contents matching different profiles, as when players played a dungeon created with similar inputs as the player's profile, they gave better scores to the dungeon in the post-test questionnaire.

Figure 96 shows the completion of immersion quests. Overall, profiles with higher immersion preferences completed more quests than those with lower preferences. The major exception was the profile *A=3, C=2, I=1, M=4*, that completed plenty of immersion quests, but possibly because of their achievement preference, to complete everything, and with a high mastery, possibly completing dungeons with ease and enabling them to finish all quests.

#### 5.3.3.2 Statistical Analysis

In this section, we run statistical tests over the 71 answers from the post-tests to know which can be discarded as not having the same average. This is a somewhat small dataset, so we must run a power analysis first. As the vast majority of players played levels that are not exactly made for their profile, we ease the rule of what is the same profile for our analysis. If the dungeon was generated with an input distant no more than 4 units from the player's profile, and

Figure 95 – Heatmap plot, grouping the percentage of creativity quests completed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator.

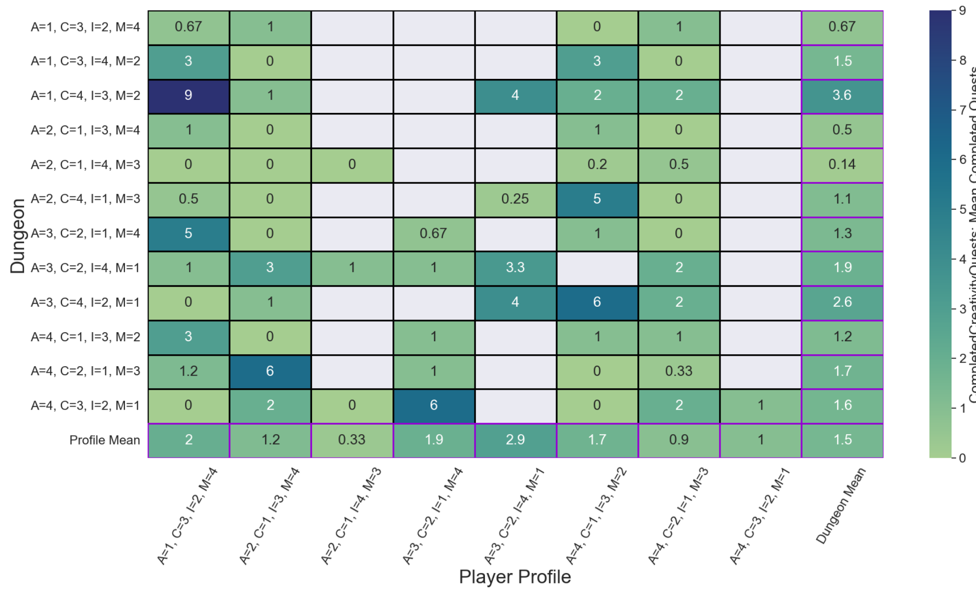
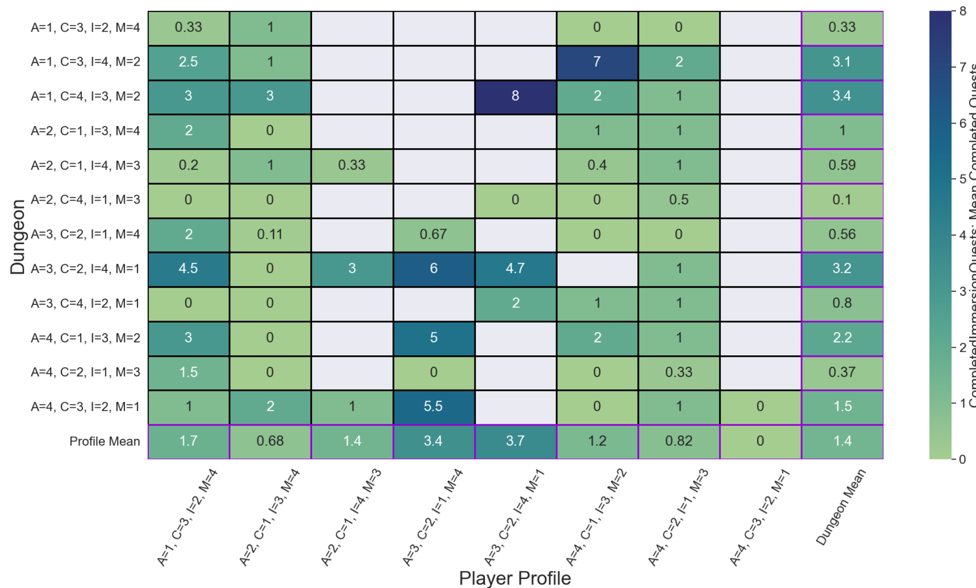



Figure 96 – Heatmap plot, grouping the percentage of immersion quests completed for each player profile in each dungeon. Dungeons are labeled as each profile's weight for the input generator.




where the distance of a single input is no greater than 1 unit, we consider it the same. This rule helps to balance the database while maintaining the profile and dungeon close enough so that the generated content may be good enough to match the player's profile.

For example, if the player has the profile *A=4, C=2, I=1, M=3*, the dungeon generated with input *A=4, C=2, I=1, M=3* is a clear match. The dungeon generated with input *A=3, C=2, I=1, M=4* is also a match because there is a distance of 2 units (*A=4* from *A=3*, and *M=3* from *M=4*), and both weights are distant only 1 unit between themselves. The same is not true for dungeon *A=2, C=1, I=3, M=4*, which has 7 units of distance from the profile. Neither for

dungeon *A=4, C=3, I=2, M=1*: although having 4 units of distance, the mastery weight has a difference of 2 units from the profile.

Applying this rule to the dataset, we find that from the 71 answers, 43 are from instances where the player's profile does not match the dungeon's, and 28 are from matching profiles. Running a power analysis on this data, with a ratio of 0.65 and 43 observations on the first population, and an alpha of 0.05, we find that, for an effect size with a 0.5 Cohen's D, the power is 0.653. This is not a significant result, and the analysis should not be considered for "visible" changes. However, for a Cohen's D of 0.8 we have a power of 0.947. Therefore, our following analysis can be considered statistically significant for easily visible changes only.

We run the same statistical setup from the experiments in Section 5.2: our two samples are independent, as the playthrough of both groups do not affect each other's results. We use the Shapiro-Wilk test of normality, finding that the population of answers for the eleven post-test questions are not Gaussian. Therefore, we use the Mann-Whitney U test, which is recommended for independent yet non-Gaussian populations.

We first conduct the Mann-Whitney U test considering that rejecting the null hypothesis means that the answers from players with a matching profile are greater than those who played with a different profile. Table 25 shows the results. There, we observe that the fun for both playing the game and completing quests, how much they liked the exploration and rewards, and also if the quests were designed by humans were all rejected as having the same mean for both populations, meaning players with a matching profile gave higher values for these answers. Not only does it show the content generated targeting the player profile is better evaluated, but it also feels more human-mand.

Table 25 – Results of the Mann-Whitney U test, considering that rejecting the null hypothesis means that the answers from players with a matching profile are greater than those playing a different profile. We reject the hypothesis with a p-value equal or smaller than 0.05, meaning less than 5% chance of a type I error.

| Question | p-value | Result |
|---|---|---|
| **Q1 - The level was fun to play** | **0.013** | **Reject H0** |
| Q2 - The level was difficult to complete | 0.127 | Don't Reject H0 |
| Q3 - The challenge was just right | 0.247 | Don't Reject H0 |
| Q4 - The enemies were difficult to defeat | 0.055 | Don't Reject H0 |
| Q5 - It was difficult to find the exit of this level | 0.663 | Don't Reject H0 |
| Q6 - I liked the size of the dungeon | 0.098 | Don't Reject H0 |
| **Q7 - I liked the exploration available on this level** | **0.021** | **Reject H0** |
| Q8 - I liked the challenge of finding the keys to this level | 0.151 | Don't Reject H0 |
| **Q9 - The rewards were on the proper amount** | **0.046** | **Reject H0** |
| **Q10 - I had fun completing the quests** | **0.036** | **Reject H0** |
| **Q11 - The quests given to me were designed by a human** | **0.019** | **Reject H0** |

For the questions with non-rejected null hypothesis, we conduct the same analysis, but for the two-sided alternative. The power of the test falls to 0.901, that is, less than 10% probability

of providing a type II error. Then, for the Mann-Whitney U test we still fail to reject H0 for all the questions. Therefore, we may not discard that the answers for these questions are the same for both groups: players playing a matching dungeon and playing different dungeons.

Conducting a statistical test on the gameplay data reported in the previous section, we have 119 samples to explore. Using the same criteria to select how many players have played on dungeons matching their profile, we have that 84 played on dungeons of different profiles, and 35 on matching dungeons. Therefore, the power of the analysis for an analysis using the same parameters as before, now with a ratio of 0.417, we have a power of 0.990 for an analysis checking if one population is greater than the other, and 0.976 for a two-sided analysis, both for a Cohen's D of 0.8.

Using the same Mann-Whitney U test, as no sample is Gaussian, we find that the null hypothesis cannot be rejected, for any parameter, both for a one-sided and two-sided analysis. Therefore, there are no statistically significant gameplay differences in our sample, considering players playing their matching dungeons or ones for a different profile, for the success rate, enemies killed, map completion, and quests of each profile completed.

#### 5.3.3.3 Final Remarks

Our system achieved the desired results considering the feedback from experiments. The system generates multiple contents in a real-time environment, fast enough for players to enjoy. The contents from all generators were well-received, with the combination as the final product. The players enjoyed navigating the dungeon and its rooms, solving the locked-door puzzles, fighting enemies created by the enemy generator and placed by the dungeon generator, and completing the quests generated by the quest generator. Our orchestrator algorithm was able to eliminate undesired contents in its post-processing step. Not only that, but as the quest generator was the frame for the other creators, we can also conclude that its quests were successful. Moreover, users with different profiles played the game differently and enjoyed the contents more (on average) when what they liked more had a higher input value in the quest generator. Thus, the system can adapt to different players' needs.

CHAPTER

6

# FINAL REMARKS

This doctorate thesis presented the research developed from January 2019 to December 2022. We advance the state-of-the-art in generating multiple contents by providing a system that simultaneously provides dungeons, enemies, quests, rooms, and the placement of enemies and locked-door puzzles inside the dungeons. We also made the system work in real-time, another challenge in the PCG, and more critical for multi-content generators.

The system was also able to create content for different player profiles, providing input values from 0 to 1, corresponding to the player's preference for each of our four profiles: achievement (for players that like to complete a goal), creativity (for the ones that like exploration), immersion (for those that like talking to NPCs and reading the game's lore), and mastery (for those that like combat). Both our offline results (Section 5.2) and online results (Section 5.3) showed that players enjoyed more the contents developed when considering their profile as input.

We also created a novel enemy-generating algorithm, as we did not find in the literature others able to generate as we did in our system. We tested this approach with players, and the algorithm was able to create diverse enemies with similar difficulties. Moreover, increasing the input parameter related to the difficulty raises the challenge for the players.

We answer our main research question: "*How can procedural content generation be effectively applied to simultaneously create multiple and coherent contents in a real-time environment and adapt them to different users' needs?*", by providing a detailed view of our systems' design, their architecture, and details on the major algorithms that compose each system. Furthermore, we tested it against metrics in the literature and players, validating the quality of our approach.

For the specific questions, we also have possible answers. "What type of architecture can effectively coordinate the generation of narrative, levels, and rules facets?": we present this architecture in Figure 59, and its variations used in the different experiments which occurred during the development of this thesis, in Figure 60, Figure 68, and 74. Our architecture is one answer to this question: an architecture which separates the profile analysis from the procedural

content generation and the game system. The architecture also separates each system by a facade, responsible to process and transform the data from one system to what the other can understand, decoupling them from profile, content and game representations. Another important feature is that each generative algorithm handles their content in such a way that the output is always feasible and matching the player's needs. Therefore, the content does not need a complex orchestrator algorithm to validate and guarantee the solution's feasibility and quality.

Our orchestrator works as a post-processing step, enhancing the solution and arranging smaller tasks rather than fixing it and validating sets of complex rules. Thus, we may answer another specific research question: "How can an orchestrator algorithm process the generated content to make them feasible and more enjoyable when placed together?". For our case, the answer was to use a more bottom-up approach, similar to *Jamming with Fake Sheets*, letting each generator handle the quality and feasibility of their contents. The quest generator acted as a frame, guiding the other generators by providing their input based on the generated missions. Then, the orchestrator filtered undesired enemies and tried to place the best combinations of enemies, considering the ones needed by the quests and how many were allocated for each room by the dungeon generator.

The latter two architectures, for the offline and online experiment, are the same, but with different algorithms being used to generate content and other small variations in the implementation. This also helps to answer our other specific research question: "How to build a modular architecture which can support different generating algorithms with as few changes as possible to other containers?". This was one of the questions we did not discuss previously in Section 4.6, as we did not present yet the results on using different algorithms with the same system's architecture. However, as seen in Sections 5.2 and 5.3, changing the algorithms was relatively easy and both provided positive results, although one was still in an offline environment, and the other was online. We used a classic EA and a MAP-Elites approach both for dungeon and enemy generations. We used a formal grammar and then a stochastic grammar for the quest generator. Not only that, we added the quest generator after the experiment with the enemy generator (Section 5.1), and the room generator after the offline experiment (Section 5.2). All without major changes in the system.

For the next question, "How can the orchestrator deal with simultaneous content generation, considering or learning from player or game designer profiles?": we found that the orchestrator did not have to handle this in detail, as the generator algorithms were enough to process the input and provide content matching different players' needs. We answer that a component that can process the profiles and serve them as inputs for the generators is enough to solve this issue. The answers to these four specific questions were tested both in our offline and online experiments (sections 5.2 and 5.3), and validated through peer-review by our publication in (PEREIRA; VIANA; TOLEDO, 2022).

For the last one, "How to adapt the simultaneous content generation within online

environments?": by using a mixture of grammars, evolutionary approaches with somewhat fast convergence, and some rule-based algorithms, aided by our architecture which enabled flexibility to change some system's rules with few changes, we were able to create our content online. Our architecture allowed us to easily select if the generators would have to serialize their contents or pass them directly to the game system, as well as permitted the game to load serialized content or the content directly received by the procedural generators. We then experimented with a small sample of players in an online setting. We can confirm that real-time generation of multiple contents is possible within our algorithms and architecture. The contents can be generated reasonably fast without losing quality while also adapting to different user's needs, as shown by our results in Section 5.3.

As future work, we will test our online setup with more players, aiming for data that enable us to run statistical analysis on the results and provide more robust answers on the content's quality. We will also validate it through peer review by sending it to relevant journals. We will later use this data to feed a machine-learning algorithm and use it in the online setup to create a theoretically infinite loop of content generation. A loop where the player's data from the previous dungeon will feed the algorithm and, in return, it will feed the generators with new input, so the player can play a new set of contents, better adapted to their profile's.

We also intend to create other experimental setups focusing on the comparative analysis of the generator algorithms: e.g., comparing the evolutionary algorithm against the MAP-Elites in better controlled environments; analyzing in more depth the enemy generators searching for patterns in the parameters and how they relate to increase the difficulty, specially the difficulty perceived by the players; and others.

Other future approaches are to create diverse NPCs to handle quests based on their profile and preferences. Furthermore, we can employ dynamic dialogues to reflect preferences, insert the item and NPCs placement in the dungeon generator, add a Sokoban-like puzzle generator to our system, and other procedural content generators. And, of course, implement these new functionalities in our game system.

At last, we plan on creating games from other genres and using our system to create content for them with as few changes as possible to test the generality of our solution, walking towards a framework for multiple content generations.

# BIBLIOGRAPHY


ADAMS, E.; ROLLINGS, A. **Fundamentals of game design**. [S.l.]: Upper Saddle River, NJ, 2006. Citation on page 40.

ALHUSSAIN, A. I.; AZMI, A. M. Automatic story generation: a survey of approaches. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 54, n. 5, p. 1–38, 2021. Citations on pages 34 and 78.

ALVAREZ, A.; DAHLSKOG, S.; FONT, J.; HOLMBERG, J.; JOHANSSON, S. Assessing aesthetic criteria in the evolutionary dungeon designer. 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/10.1145/3235765.3235810>. Citation on page 70.

Alvarez, A.; Dahlskog, S.; Font, J.; Togelius, J. Empowering quality diversity in dungeon design with interactive constrained map-elites. In: **2019 IEEE Conference on Games (CoG)**. [S.l.: s.n.], 2019. p. 1–8. ISSN 2325-4270. Citation on page 71.

ANTONIADES, T. **Hellblade: Senua's Sacrifice**. 2017. Microsoft Windows, PlayStation 4. Available: <http://www.hellblade.com/>. Citation on page 40.

ATMAJA, P. W.; PARLIKA, R. *et al.* A preliminary study on integrating procedural content generation into game development process. **IJCONSIST JOURNALS**, v. 1, n. 1, p. 27–34, 2019. Citation on page 33.

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**. New York, NY, USA: ACM, 2018. (FDG '18), p. 5:1–5:7. ISBN 978-1-4503-6571-0. Available: <http://doi.acm.org/10.1145/3235765.3235780>. Citation on page 67.

Baldwin, A.; Dahlskog, S.; Font, J. M.; Holmberg, J. Mixed-initiative procedural generation of dungeons using game design patterns. In: **2017 IEEE Conference on Computational Intelligence and Games (CIG)**. [S.l.: s.n.], 2017. p. 25–32. ISSN 2325-4289. Citations on pages 71, 72, and 74.

BARTLE, R. Designing virtual worlds. In: ____. [S.l.: s.n.], 2003. p. –768. ISBN 0131018167. Citations on pages 35 and 64.

BEAUPRE, S.; WILES, T.; BRIGGS, S.; SMITH, G. A design pattern approach for multi-game level generation. In: **Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference**. [S.l.: s.n.], 2018. Citation on page 83.

BECK, K. **Test-driven development: by example**. [S.l.]: Addison-Wesley Professional, 2003. Citation on page 83.

BELL, I.; BRABEN, D. **The Elite Home Page**. 1984. Available: <http://www.iancgbell.clara.net/elite/>. Citations on pages 46 and 70.

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**. New York, NY, USA: ACM, 2018. (FDG '18), p. 1:1–1:10. ISBN 978-1-4503-6571-0. Available: <http://doi.acm.org/10.1145/3235765.3235774>. Citations on pages 36, 67, 86, and 87.

BLESZINSKI, J. S. C. **Unreal**. 1998. Microsoft Windows; Mac OS. Available: <https://web.archive.org/web/20010331080920/http://www.unreal.com/index2.html>. Citation on page 44.

BONTCHEV, B.; GEORGIEVA, O. Playing style recognition through an adaptive video game. **Computers in Human Behavior**, v. 82, p. 136 – 147, 2018. ISSN 0747-5632. Available: <http://www.sciencedirect.com/science/article/pii/S0747563217307264>. Citations on pages 35 and 69.

BREAULT, V.; OUELLET, S.; DAVIES, J. Let conan tell you a story: Procedural quest generation. **Entertainment Computing**, v. 38, p. 100422, 2021. ISSN 1875-9521. Available: <https://www.sciencedirect.com/science/article/pii/S1875952121000197>. Citation on page 75.

BROWN, S. **The C4 model for visualising software architecture**. [S.l.]: Leanpub, 2022. Citation on page 57.

BROWNE, C.; MAIRE, F. Evolutionary game design. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE Computer Society, Los Alamitos, CA, USA, v. 2, n. 01, p. 1–16, jan 2010. ISSN 1943-0698. Citations on pages 80 and 87.

CHARITY, M.; GREEN, M. C.; KHALIFA, A.; TOGELIUS, J. Mech-elites: Illuminating the mechanic space of gvg-ai. In: **International Conference on the Foundations of Digital Games**. New York, NY, USA: Association for Computing Machinery, 2020. (FDG '20). ISBN 9781450388078. Available: <https://doi.org/10.1145/3402942.3402954>. Citation on page 71.

CHATZILYGEROUDIS, K.; CULLY, A.; VASSILIADES, V.; MOURET, J.-B. Quality-diversity optimization: A novel branch of stochastic optimization. In: ____. **Black Box Optimization, Machine Learning, and No-Free Lunch Theorems**. Cham: Springer International Publishing, 2021. p. 109–135. ISBN 978-3-030-66515-9. Available: <https://doi.org/10.1007/978-3-030-66515-9_4>. Citations on pages 23, 50, and 51.

CHEN, T.; GUY, S. J. Gigl: A domain specific language for procedural content generation with grammatical representations. In: **Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference**. [S.l.: s.n.], 2018. Citations on pages 76 and 77.

CIVIT, M.; CIVIT-MASOT, J.; CUADRADO, F.; ESCALONA, M. J. A systematic review of artificial intelligence-based music generation: Scope, applications, and future trends. **Expert Systems with Applications**, Elsevier, p. 118190, 2022. Citation on page 34.

COHEN, S.; GIMPEL, K.; SMITH, N. A. Logistic normal priors for unsupervised probabilistic grammar induction. **Advances in Neural Information Processing Systems**, v. 21, 2008. Citation on page 57.

COLLINS, S. **A/B Testing for Game Design Iteration: A Bayesian Approach**. 2014. <https://www.gdcvault.com/play/1020201/A-B-Testing-for-Game>. Citation on page 35.

COOK, M. Software engineering for automated game design. In: IEEE. **2020 IEEE Conference on Games (CoG)**. [S.l.], 2020. p. 487–494. Citation on page 85.

COOK, M.; COLTON, S. A rogue dream: Automatically generating meaningful content for games. In: **Tenth Artificial Intelligence and Interactive Digital Entertainment Conference**. [S.l.: s.n.], 2014. Citations on pages 80, 81, and 87.

COOK, M.; COLTON, S.; PEASE, A. Aesthetic considerations for automated platformer design. In: **Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**. [S.l.]: AAAI Press, 2012. (AIIDE'12), p. 124–129. Citations on pages 80, 81, and 87.

COOK, M.; COLTON, S.; RAAD, A.; GOW, J. Mechanic miner: Reflection-driven game mechanic discovery and level design. In: **EvoApplications**. [S.l.: s.n.], 2013. Citations on pages 80 and 87.

COOK, M.; ELADHARI, M.; NEALEN, A.; TREANOR, M.; BOXERMAN, E.; JAFFE, A.; SOTTOSANTI, P.; SWINK, S. Pcg-based game design patterns. **arXiv preprint arXiv:1610.03138**, 2016. Citations on pages 16, 84, and 85.

CORMEN, T. H. Section 22.3: Depth-first search. In: ____. **Introduction to algorithms**. [S.l.]: MIT Press, 2008. p. 540–549. Citations on pages 23, 53, 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**, v. 26, n. 2, p. 257–306, Jun 2016. ISSN 1573-1391. Available: <https://doi.org/10.1007/s11257-016-9170-1>. Citations on pages 35 and 70.

CUI, X.; SHI, H. A*-based pathfinding in modern computer games. **International Journal of Computer Science and Network Security**, v. 11, n. 1, p. 125;130, 2011. Available: <http://paper.ijcsns.org/07_book/201101/20110119.pdf>. Citation on page 52.

DEALS, T. **This engine is dominating the gaming industry right now**. 2016. Available: <https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now/>. Citation on page 46.

DESIGN, A. I. **Rogue for Amiga**. 1984. Available: <https://www.mobygames.com/game/rogue>. Citations on pages 46 and 70.

DEVELOPMENTS, F. **Elite:Dangerous**. 2014. Available: <https://www.elitedangerous.com/>. Citation on page 47.

DONG, J.; LIU, J.; YAO, K.; CHANTLER, M.; QI, L.; YU, H.; JIAN, M. Survey of procedural methods for two-dimensional texture generation. **Sensors**, MDPI, v. 20, n. 4, p. 1135, 2020. Citation on page 34.

DORAN, J.; PARBERRY, I. A prototype quest generator based on a structural analysis of quests from four mmorpgs. In: **Proceedings of the 2nd International Workshop on Procedural Content Generation in Games**. New York, NY, USA: Association for Computing Machinery, 2011. (PCGames '11). ISBN 9781450308724. Available: <https://doi.org/10.1145/2000919.2000920>. Citations on pages 74, 75, 120, and 121.

DORMANS, J.; BAKKES, S. Generating missions and spaces for adaptable play experiences. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 3, n. 3, p. 216–228, Sept 2011. ISSN 1943-068X. Citation on page 40.

EIBEN, A. E.; SMITH, J. E. **Introduction to Evolutionary Computing**. [S.l.]: SpringerVerlag, 2003. ISBN 3540401849. Citations on pages 15, 23, and 49.

ESCAYG, K. M. S. **Uncharted: The Lost Legacy**. 2017. PlayStation 4. Available: <https://www.unchartedthegame.com/en-us/games/uncharted-the-lost-legacy/>. Citation on page 40.

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 116.

FONT, J. M.; IZQUIERDO, R.; MANRIQUE, D.; TOGELIUS, J. Constrained level generation through grammar-based evolutionary algorithms. In: SQUILLERO, G.; BURELLI, P. (Ed.). **Applications of Evolutionary Computation**. Cham: Springer International Publishing, 2016. p. 558–573. ISBN 978-3-319-31204-0. Citation on page 71.

FOWLER, M. **Refactoring: improving the design of existing code**. [S.l.]: Addison-Wesley Professional, 2018. Citation on page 83.

FREDMAN, M.; TARJAN, R. Fibonacci heaps and their uses in improved network optimization algorithms. **25th Annual Symposium onFoundations of Computer Science, 1984.**, 1984. Citation on page 51.

FUJIBAYASHI, H. **The Legend of Zelda: The Minish Cap**. 2004. Game Boy Advance, Nintendo 3DS, Nintendo WiiU. Available: <https://www.nintendo.com/games/detail/Uq4mHxYWIHPUlTlOd967IkJ1lvpCuYak>. Citation on page 42.

____. **The Legend of Zelda: Breath of the Wild**. 2017. Nintendo Switch; Wii U. Available: <https://www.zelda.com/breath-of-the-wild/>. Citation on page 40.

GAMES, E. **Game Engine Technology by Unreal**. 2018. Available: <https://www.unrealengine.com/en-US/features>. Citation on page 45.

GAMES, F. **Civilization VI**. 2016. Available: <https://www.civilization.com/>. Citation on page 47.

GAMES, H. **No Man's Sky**. 2016. Citations on pages 47, 71, 73, and 74.

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. Citation on page 74.

GILBERT, É.; CONKLIN, D. A probabilistic context-free grammar for melodic reduction. In: **Proceedings of the International Workshop on Artificial Intelligence and Music, 20th International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2007. p. 83–94. Citation on page 77.

GIRAUD, M.; STAWORKO, S. Modeling musical structure with parametric grammars. In: SPRINGER. **International conference on mathematics and computation in music**. [S.l.], 2015. p. 85–96. Citations on pages 76 and 77.

GRAND, S.; CLIFF, D. Creatures: Entertainment software agents with artificial life. **Autonomous Agents and Multi-Agent Systems**, v. 1, p. 39–57, 03 1998. Citations on pages 71, 73, and 74.

GRAY, R. C.; ZHU, J.; ARIGO, D.; FORMAN, E.; nóN, S. O. Player modeling via multi-armed bandits. In: **International Conference on the Foundations of Digital Games**. New York, NY, USA: Association for Computing Machinery, 2020. (FDG '20). ISBN 9781450388078. Available: <https://doi.org/10.1145/3402942.3402952>. Citation on page 68.

GRAY, R. C.; ZHU, J.; ONTAñóN, S. Multiplayer modeling via multi-armed bandits. In: **2021 IEEE Conference on Games (CoG)**. [S.l.: s.n.], 2021. p. 01–08. Citations on pages 36 and 68.

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 80 and 87.

GREGORY, J. **Game Engine Architecture, Second Edition**. CRC Press, 2014. ISBN 9781466560062. Available: <https://books.google.com.br/books?id=LlLSBQAAQBAJ>. Citations on pages 44 and 45.

HALL, J. R. S. P. A. M. S. G. T. **Doom**. 1993. MS-DOS. Available: <https://www.mobygames.com/game/doom>. Citation on page 44.

HARRISON, B.; PURDY, C.; RIEDL, M. O. Toward automated story generation with markov chain monte carlo methods and deep neural networks. In: **Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference**. [S.l.: s.n.], 2017. Citation on page 54.

Hartsook, K.; Zook, A.; Das, S.; Riedl, M. O. Toward supporting stories with procedurally generated game worlds. In: **2011 IEEE Conference on Computational Intelligence and Games (CIG'11)**. [S.l.: s.n.], 2011. p. 297–304. ISSN 2325-4270. Citations on pages 80 and 87.

HARTSOOK, K.; ZOOK, A.; DAS, S.; RIEDL, M. O. Toward supporting stories with procedurally generated game worlds. In: **2011 IEEE Conference on Computational Intelligence and Games (CIG'11)**. [S.l.: s.n.], 2011. p. 297–304. Citations on pages 81 and 120.

HEIJNE, N. **Investigating the Relationship between FFM, Game Literacy, Content Generation and Game-play Preference**. Master's Thesis (Master's Thesis) — University of Amsterdam, 2016. Available: <https://esc.fnwi.uva.nl/thesis/centraal/files/f840179296.pdf>. Citations on pages 35, 36, 68, 82, 86, 87, and 95.

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**. New York, NY, USA: ACM, 2017. (FDG '17), p. 11:1–11:10. ISBN 978-1-4503-5319-9. Available: <http://doi.acm.org/10.1145/3102071.3102091>. Citation on page 68.

HENDRIKX, M.; MEIJER, S.; VELDEN, J. V. D.; IOSUP, A. Procedural content generation for games: A survey. In: . New York, NY, USA: ACM, 2013. v. 9, n. 1, p. 1:1–1:22. ISSN 1551-6857. Available: <http://doi.acm.org/10.1145/2422956.2422957>. Citation on page 47.

HOOVER, A.; CACHIA, W.; LIAPIS, A.; YANNAKAKIS, G. Audioinspace: Exploring the creative fusion of generative audio, visuals and gameplay. In: . [S.l.: s.n.], 2015. v. 9027, p. 101–112. ISBN 978-3-319-16497-7. Citations on pages 80, 81, and 87.

HOSOKAWA, J. L. M. T. **Metroid: Samus Returns**. 2017. Nintendo 3DS. Available: <https://metroidsamusreturns.nintendo.com/>. Citation on page 40.

IOSUP, A. Poggi: Generating puzzle instances for online games on grid infrastructures. In: . Chichester, UK: John Wiley and Sons Ltd., 2011. v. 23, n. 2, p. 158–171. ISSN 1532-0626. Available: <http://dx.doi.org/10.1002/cpe.1638>. Citation on page 47.

JAMESON, A. Adaptive interfaces and agents. In: ____. **The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications**. USA: L. Erlbaum Associates Inc., 2002. p. 305–330. ISBN 0805838384. Citations on pages 63, 69, and 144.

JAQUAYS, G. D. T. W. P. **Quake III Arena**. 1999. Microsoft Windows; Mac OSX; Linux; Dreamcast; PlayStation 2; Xbox 360; iOS. Available: <https://web.archive.org/web/20020718214708/http://quake3arena.com:80/>. Citation on page 44.

JONGE, M. de. **Horizon Zero Dawn**. 2017. PlayStation 4. Available: <https://www.guerrilla-games.com/play/horizon>. Citation on page 40.

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 116.

KANTHARAJU, P.; ALDERFER, K.; ZHU, J.; CHAR, B.; SMITH, B.; ONTAñóN, S. Modeling player knowledge in a parallel programming educational game. **IEEE Transactions on Games**, v. 14, n. 1, p. 64–75, 2022. Citation on page 67.

KARAVOLOS, D.; LIAPIS, A.; YANNAKAKIS, G. A multifaceted surrogate model for search-based procedural content generation. **IEEE Transactions on Games**, v. 13, n. 1, p. 11–22, 2021. Citations on pages 34, 82, and 87.

KEGEL, B. D.; HAAHR, M. Procedural puzzle generation: A survey. **IEEE Transactions on Games**, IEEE, v. 12, n. 1, p. 21–40, 2019. Citation on page 33.

KHALIFA, A.; LEE, S.; NEALEN, A.; TOGELIUS, J. Talakat: bullet hell generation through constrained map-elites. In: **GECCO ’18**. [S.l.: s.n.], 2018. Citations on pages 71, 72, 73, and 74.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J.; FORSGREN, N. **The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations**. Portland, USA: IT Revolution, 2021. Citation on page 83.

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**, v. 4, p. 36–50, 04 2014. Citation on page 69.

KORF, R. E. Depth-first iterative-deepening: An optimal admissible tree search. **Artif. Intell.**, v. 27, p. 97–109, 1985. Citation on page 53.

KOZLOVA, A.; BROWN, J. A.; READING, E. Examination of representational expression in maze generation algorithms. In: **2015 IEEE Conference on Computational Intelligence and Games (CIG)**. [S.l.: s.n.], 2015. p. 532–533. ISSN 2325-4289. Citation on page 53.

KREMINSKI, M.; DICKINSON, M.; MATEAS, M.; WARDRIP-FRUIN, N. Why are we like this?: The ai architecture of a co-creative storytelling game. In: **International Conference on the Foundations of Digital Games**. New York, NY, USA: Association for Computing Machinery, 2020. (FDG '20). ISBN 9781450388078. Available: <https://doi.org/10.1145/3402942.3402953>. Citation on page 83.

Kybartas, B.; Bidarra, R. A survey on story generation techniques for authoring computational narratives. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 9, n. 3, p. 239–253, Sep. 2017. ISSN 1943-068X. Citation on page 77.

Kybartas, B.; Verbrugge, C. Analysis of regen as a graph-rewriting system for quest generation. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 6, n. 2, p. 228–242, 2014. Citations on pages 77 and 95.

LABS, U. **State of Decay 2**. 2018. Citations on pages 72, 73, and 74.

LESTER, P. **A* Pathfinding for Beginners**. 2005. Available: <http://homepages.abdn.ac.uk/f.guerin/pages/teaching/CS1013/practicals/aStarTutorial.htm>. Citations on pages 15 and 53.

LEWIS, M. Game ai pro: collected wisdom of game ai professionals. In: ____. [S.l.]: CRC Press, 2013. chap. 36. Citation on page 57.

LI, B.; CHEN, R.; XUE, Y.; WANG, R.; LI, W.; GUZDIAL, M. Ensemble learning for mega man level generation. In: **The 16th International Conference on the Foundations of Digital Games (FDG) 2021**. [S.l.: s.n.], 2021. p. 1–9. Citation on page 54.

____. Ensemble learning for mega man level generation. In: **The 16th International Conference on the Foundations of Digital Games (FDG) 2021**. New York, NY, USA: Association for Computing Machinery, 2021. (FDG'21). ISBN 9781450384223. Available: <https://doi.org/10.1145/3472538.3472592>. Citation on page 76.

LIAPIS, A. Multi-segment evolution of dungeon game levels. In: **Proceedings of the Genetic and Evolutionary Computation Conference**. New York, NY, USA: ACM, 2017. (GECCO '17), p. 203–210. ISBN 978-1-4503-4920-8. Available: <http://doi.acm.org/10.1145/3071178.3071180>. Citations on pages 40, 71, 72, and 74.

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 108.

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 16, 34, 78, 79, 80, 86, 90, and 94.

LIMA, E. Soares de; FEIJó, B.; FURTADO, A. L. Procedural generation of quests for games using genetic algorithms and automated planning. In: **2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)**. [S.l.: s.n.], 2019. p. 144–153. Citations on pages 74 and 185.

LIN, L.; WU, T.; PORWAY, J.; XU, Z. A stochastic graph grammar for compositional object representation and recognition. **Pattern Recognition**, v. 42, n. 7, p. 1297–1307, 2009. ISSN 0031-3203. Available: <https://www.sciencedirect.com/science/article/pii/S0031320308004603>. Citation on page 76.

LIU, J.; SNODGRASS, S.; KHALIFA, A.; RISI, S.; YANNAKAKIS, G. N.; TOGELIUS, J. Deep learning for procedural content generation. **Neural Computing and Applications**, Springer, v. 33, n. 1, p. 19–37, 2021. Citation on page 33.

LOPES, P. L.; LIAPIS, A.; YANNAKAKIS, G. N. Framing tension for game generation. In: **ICCC**. [S.l.: s.n.], 2016. Citations on pages 80 and 87.

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, CHI PLAY 2018, Melbourne, VIC, Australia, October 28-31, 2018**. [s.n.], 2018. p. 531–538. Available: <https://doi.org/10.1145/3270316.3271526>. Citations on pages 35 and 69.

LUBAN, P. **Designing and Integrating Puzzles in Action-Adventure Games**. 2002. Available: <https://www.gamasutra.com/view/feature/131326/designing_and_integrating_puzzles_.php>. Citation on page 39.

MARTIN, R. C.; GRENNING, J.; BROWN, S.; HENNEY, K.; GORMAN, J. **Clean architecture: a craftsman's guide to software structure and design**. [S.l.]: Prentice Hall, 2018. Citation on page 83.

MARTIN, R. C.; NEWKIRK, J.; KOSS, R. S. **Agile software development: principles, patterns, and practices**. [S.l.]: Prentice Hall Upper Saddle River, NJ, 2003. Citation on page 90.

MARTINOVIC, A.; GOOL, L. V. Bayesian grammar learning for inverse procedural modeling. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2013. p. 201–208. Citations on pages 76 and 77.

MATHEW, G. E. Direction based heuristic for pathfinding in video games. **Procedia Computer Science**, v. 47, p. 262–271, 2015. Citation on page 52.

MAXIS. **Spore**. 2008. Citations on pages 71, 73, and 74.

MCMILLEN, E. **The Binding of Isaac: Rebirth**. 2014. Microsoft Windows, OS X, Linux, PlayStation 4, PlayStation Vita, Wii U, New Nintendo 3DS, Xbox One, iOS, Nintendo Switch. Available: <http://bindingofisaac.com/>. Citation on page 48.

MCMILLEN, E.; HIMSL, F. 2011. Available: <https://bindingofisaac.com/>. Citation on page 89.

____. **The Binding of Isaac**. 2011. Microsoft Windows, OS X, Linux. Available: <http://bindingofisaac.com/>. Citation on page 47.

MEDUNA, A. **Formal Languages and Computation**. [S.l.]: Taylor\& Francis Informa plc, 2014. Citation on page 55.

MELHART, D.; AZADVAR, A.; CANOSSA, A.; LIAPIS, A.; YANNAKAKIS, G. N. Your gameplay says it all: Modelling motivation in tom clancy's the division. **CoRR**, abs/1902.00040, 2019. Available: <http://arxiv.org/abs/1902.00040>. Citations on pages 35 and 66.

MIZUTANI, W. K.; DAROS, V. K.; KON, F. Software architecture for digital game mechanics: A systematic literature review. **Entertainment Computing**, Elsevier, v. 38, p. 100421, 5 2021. ISSN 1875-9521. Citation on page 83.

MONTFORT, N.; PÉREZ, R. P. y; HARRELL, D. F.; CAMPANA, A. Slant: A blackboard system to generate plot, figuration, and narrative discourse aspects of stories. In: **ICCC**. [S.l.: s.n.], 2013. p. 168–175. Citation on page 73.

MOTOKURA, K. **Super Mario Odyssey**. 2017. Nintendo Switch. Available: <https://www.nintendo.com/games/detail/super-mario-odyssey-switch>. Citation on page 40.

NINTENDO. **the official home for the legend of zelda**. 2020. Available: <https://www.zelda.com/>. Citation on page 89.

NORTH, B. **Diablo III**. 2012. Citations on pages 72 and 74.

OPENAI; BERNER, C.; BROCKMAN, G.; CHAN, B.; CHEUNG, V.; DęBIAK, P.; ...; ZHANG, S. **Dota 2 with Large Scale Deep Reinforcement Learning**. 2019. Citation on page 35.

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**. New York, NY, USA: Association for Computing Machinery, 2017. (CHI '17), p. 1015–1027. ISBN 9781450346559. Available: <https://doi.org/10.1145/3025453.3025577>. Citation on page 35.

ORJI, R.; TONDELLO, G. F.; NACKE, L. E. Personalizing persuasive strategies in gameful systems to gamification user types. In: **Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2018. (CHI '18), p. 1–14. ISBN 9781450356206. Available: <https://doi.org/10.1145/3173574.3174009>. Citations on pages 65 and 66.

PEDERSEN, R. **Game Design Foundations**. Jones & Bartlett Learning, 2009. 291 p. ISBN 9781449663926. Available: <https://books.google.com.br/books?id=0flChljb9IIC>. Citation on page 44.

PERCHY, S.; SARRIA, G. Musical composition with stochastic context-free grammars. In: **8th Mexican International Conference on Artificial Intelligence (MICAI 2009)**. [S.l.: s.n.], 2009. Citations on pages 76 and 77.

PEREIRA, L. T. **Procedural Generation of Dungeon Maps, Missions and Rooms**. Master's Thesis (Master's Thesis) — Universidade de São Paulo, 2018. Available: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-25032019-144917/en.php>. Citations on pages 97 and 149.

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 17, 23, 70, 97, 98, 102, 103, 104, 106, 110, 111, and 115.

PEREIRA, L. T.; PRADO, P. V. S.; TOLEDO, C. Evolving dungeon maps with locked door missions. In: **2018 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2018. p. 1–8. Citation on page 110.

PEREIRA, L. T.; VIANA, B. M.; TOLEDO, C. F. 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 34, 73, 74, 149, and 150.

PEREIRA, L. T.; VIANA, B. M. F.; TOLEDO, C. F. M. A system for orchestrating multiple procedurally generated content for different player profiles. **IEEE Transactions on Games**, p. 1–11, 2022. Citations on pages 87, 159, and 194.

PLURALSIGHT. **Unity, Source 2, Unreal Engine 4, or CryENGINE - Which Game Engine Should I Choose?** 2015. Available: <https://www.pluralsight.com/blog/film-games/unity-udk-cryengine-game-engine-choose>. Citation on page 46.

Prager, R. P.; Troost, L.; Brüggenjürgen, S.; Melhart, D.; Yannakakis, G.; Preuss, M. An experiment on game facet combinationİn: **2019 IEEE Conference on Games (CoG)**. [S.l.: s.n.], 2019. p. 1–8. Citations on pages 81 and 87.

PRODUCTIONS, M. **Middle-earth: Shadow of Mordor**. 2014. Citation on page 72.

RASHID, Z. **3 Game Engines Every Indie Developer should try**. 2016. Available: <https://www.linkedin.com/pulse/3-game-engines-every-indie-developer-should-try-zamilur-rashid-csm/>. Citation on page 46.

RICHARDS, M.; FORD, N. **Fundamentals of Software Architecture**. [S.l.]: O'Reilly Media, Inc., 2020. Citation on page 57.

RISI, S.; TOGELIUS, J. Increasing generality in machine learning through procedural content generation. **Nature Machine Intelligence**, v. 2, n. 8, p. 428–436, 2020. Citation on page 33.

_____. Increasing generality in machine learning through procedural content generation. **Nature Machine Intelligence**, Nature Publishing Group, v. 2, n. 8, p. 428–436, 2020. Citation on page 33.

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**. New York, NY, USA: ACM, 2018. (CHI PLAY '18), p. 417–428. ISBN 978-1-4503-5624-4. Available: <http://doi.acm.org/10.1145/3242671.3242700>. Citations on pages 36, 69, and 161.

ROBINETT, W. **Adventure**. 1979. Atari 2600. Available: <http://www.warrenrobinett.com/adventure/>. Citation on page 40.

RODRIGUES, L.; BRANCHER, J. Procedurally generating a digital math game's levels: Does it impact players' in-game behavior? **Entertainment Computing**, v. 32, p. 100325, 2019. ISSN 1875-9521. Available: <http://www.sciencedirect.com/science/article/pii/S1875952119300308>. Citation on page 33.

ROIG, C.; TARDóN, L. J.; BARBANCHO, I.; BARBANCHO, A. M. A non-homogeneous beat-based harmony markov model. **Knowledge-Based Systems**, v. 142, p. 85–94, 2018. ISSN 0950-7051. Available: <https://www.sciencedirect.com/science/article/pii/S0950705117305592>. Citations on pages 76 and 77.

ROLLINGS, A. E. A. **Andrew Rollings and Ernest Adams on Game Design**. [S.l.]: New Riders Publishing, 2003. 446 p. ISBN 1-59273-001-9. Citation on page 39.

ROSS, S. H. **Introduction to Probability Models**. 12. ed. Massachusetts, USA: Academic Press, 2019. Citation on page 54.

RYAN, M.-L. Beyond myth and metaphor: Narrative in digital media. **Poetics Today**, v. 23, n. 4, p. 581–609, 2002. Citation on page 39.

SHARIF, M.; ZAFAR, A.; MUHAMMAD, U. Design patterns and general video game level generation. **International Journal of Advanced Computer Science and Applications**, v. 8, n. 9, p. 393–398, 2017. Citations on pages 71, 72, and 74.

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. Citation on page 108.

SNODGRASS, S.; ONTAñóN, S. Procedural level generation using multi-layer level representations with mdmcs. In: **2017 IEEE Conference on Computational Intelligence and Games (CIG)**. [S.l.: s.n.], 2017. p. 280–287. Citations on pages 54 and 76.

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. Citations on pages 104, 105, and 106.

SUMMERVILLE, A.; MATEAS, M. Sampling hyrule: Multi-technique probabilistic level generation for action role playing games. In: **2015 AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**. [s.n.], 2015. Available: <https://www.aaai.org/ocs/index.php/AIIDE/AIIDE15/paper/view/11570>. Citation on page 40.

SUMMERVILLE, A.; SNODGRASS, S.; GUZDIAL, M.; HOLMGåRD, C.; HOOVER, A.; ISAKSEN, A.; NEALEN, A.; TOGELIUS, J. Procedural content generation via machine learning (pcgml). **IEEE Transactions on Games**, PP, 02 2017. Citation on page 33.

SUMMERVILLE, A. J.; BEHROOZ, M.; MATEAS, M.; JHALA, A. The learning of zelda: Data-driven learning of level topology. In: **Proceedings of the 10th International Conference on the Foundations of Digital Games**. [S.l.: s.n.], 2015. Citation on page 40.

SUPERGIANT GAMES. **Hades**. 2020. Citation on page 33.

SYEED, A.; RASHID, T. U.; DHALI, M. A.; FARID, H. M.; SADIK, A. M. A comprehensive and comparative study of maze-solving techniques by implementing graph theory. In: **Artificial Intelligence and Computational Intelligence, International Conference on(AICI)**. [s.n.], 2010. v. 01, p. 52–56. Available: <doi.ieeecomputersociety.org/10.1109/AICI.2010.18>. Citation on page 53.

TARO, Y. **Nier: Automata**. 2017. PlayStation 4; Microsoft Windows. Available: <https://www.niergame.com/>. Citation on page 40.

THUE, D.; BULITKO, V.; SPETCH, M.; WASYLISHEN, E. Interactive storytelling: A player modelling approach. **Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**, v. 3, n. 1, p. 43–48, Sep. 2021. Available: <https://ojs.aaai.org/index.php/AIIDE/article/view/18780>. Citation on page 74.

TOGELIUS, J.; KASTBJERG, E.; SCHEDL, D.; YANNAKAKIS, G. N. What is procedural content generation?: Mario on the borderline. In: **Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games**. New York, NY, USA: ACM, 2011. (PCGames '11), p. 3:1–3:6. ISBN 978-1-4503-0872-4. Available: <http://doi.acm.org/10.1145/2000919.2000922>. Citation on page 33.

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. Citation on page 90.

TOGELIUS, J.; YANNAKAKIS, G. N.; STANLEY, K. O.; BROWNE, C. Search-based procedural content generation: A taxonomy and survey. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE, v. 3, n. 3, p. 172–186, 2011. Citation on page 33.

TONDELLO, G. F.; WEHBE, R. R.; DIAMOND, L.; BUSCH, M.; MARCZEWSKI, A.; NACKE, L. E. The gamification user types hexad scale. In: **Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play**. New York, NY, USA: Association for Computing Machinery, 2016. (CHI PLAY '16), p. 229–243. ISBN 9781450344562. Available: <https://doi.org/10.1145/2967934.2968082>. Citation on page 66.

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**. New York, NY, USA: Association for Computing Machinery, 2012. (FDG '12), p. 18–25. ISBN 9781450313339. Available: <https://doi.org/10.1145/2282338.2282347>. Citations on pages 80, 81, and 87.

TREANOR, M.; ZOOK, A.; ELADHARI, M. P.; TOGELIUS, J.; SMITH, G.; COOK, M.; THOMPSON, T.; MAGERKO, B.; LEVINE, J.; SMITH, A. Ai-based game design patterns. Society for the Advancement of Digital Games, 2015. Citations on pages 16, 83, and 84.

____. Ai-based game design patterns. In: **Proceedings of the 10th International Conference on the Foundations of Digital Games 2015 (FDG 2015)**. USA: Society for the Advancement of Digital Games, 2015. ISBN 9780991398249. Available: <http://www.fdg2015.org/papers/fdg2015%5fpaper%5f23.pdf>. Citation on page 33.

VAHLO, J.; KAAKINEN, J.; HOLM, S.; KOPONEN, A. Digital game dynamics preferences and player types: Preferences in game dynamics. **Journal of Computer-Mediated Communication**, 02 2017. Citations on pages 35 and 65.

____. Digital game dynamics preferences and player types: Preferences in game dynamics. **Journal of Computer-Mediated Communication**, 02 2017. Citations on pages 36 and 161.

VALENCIA-ROSADO, L. O.; STAROSTENKO, O. Methods for procedural terrain generation: a review. In: SPRINGER. **Mexican Conference on Pattern Recognition**. [S.l.], 2019. p. 58–67. Citation on page 34.

VALTCHANOV, V.; BROWN, J. A. Evolving dungeon crawler levels with relative placement. In: **Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering**. New York, NY, USA: ACM, 2012. (C3S2E '12), p. 27–35. ISBN 978-1-4503-1084-0. Available: <http://doi.acm.org/10.1145/2347583.2347587>. Citation on page 40.

VALVE. **Left 4 Dead 2**. 2009. Citations on pages 72, 73, and 74.

VIANA, B. M.; PEREIRA, L. T.; TOLEDO, C. F. Illuminating the space of dungeon maps, locked-door missions and enemy placement through map-elites. **SSRN Electronic Journal**, 2022. Available: <https://ssrn.com/abstract=4180762>. Citation on page 70.

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 33 and 70.

VIANA, B. M. d. F. **Orchestrating and Adapting of Dungeon Levels, Locked-door Missions, and Enemies**. Master's Thesis (Master's Thesis) — Universidade de São Paulo, 2022. Available: <https://www.teses.usp.br/teses/disponiveis/55/55134/tde-19072022-164759/en.php>. Citations on pages 106 and 149.

VIANA, B. M. F.; PEREIRA, L. T.; TOLEDO, C. F. M. Illuminating the space of enemies through map-elites. In: **2022 IEEE Conference on Games (CoG)**. [S.l.: s.n.], 2022. p. 17–24. Citations on pages 73, 74, 119, and 149.

VINYALS, O.; BABUSCHKIN, I.; CHUNG, J.; MATHIEU, M.; JADERBERG, M.; CZARNECKI, W.; ...; SILVER, D. **AlphaStar: Mastering the Real-Time Strategy Game StarCraft II**. 2019. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>. Citation on page 35.

VO, N. N.; BOBICK, A. F. From stochastic grammar to bayes network: Probabilistic parsing of complex activity. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2014. p. 2641–2648. Citation on page 76.

YEE, N. Motivations for play in online games. **Cyberpsychology & behavior : the impact of the Internet, multimedia and virtual reality on behavior and society**, v. 9, p. 772–5, 01 2007. Citations on pages 35 and 65.

YEE, N.; DUCHENEAUT, N. Gamer motivation profiling: Uses and applications. **Games User Research**, Oxford University Press, Oxford, UK, p. 485–490, 2018. Citations on pages 36 and 161.

YU, K. K.; GUZDIAL, M.; STURTEVANT, N. The definition-context-purpose paradigm and other insights from industry professionals about the definition of a quest. **Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**, v. 17, n. 1, p. 107–114, Oct. 2021. Available: <https://ojs.aaai.org/index.php/AIIDE/article/view/18897>. Citation on page 74.

APPENDIX

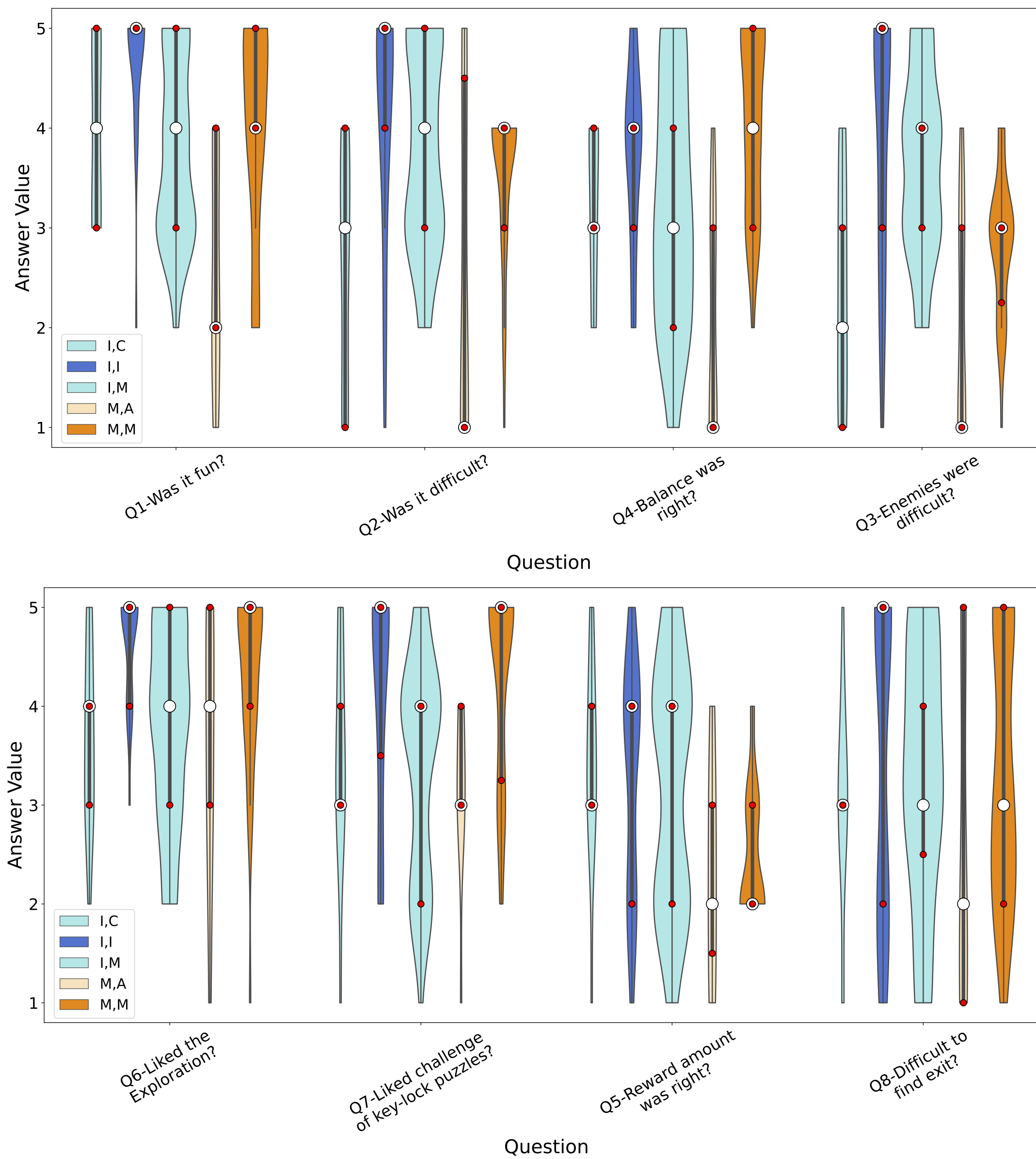
A

# OFFLINE EXPERIMENTAL SETUP EXTENDED RESULTS

To offer the reader a more detailed and visual representation of our data for the offline experiment, specially regarding how the post-test answer for each group of player varied, we present in this appendix Fig. 97. This is the same Violin Plot as Fig. 73, 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. As shown in Table 7 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 need more 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 7).

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 complicated, 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).

Figure 97 – 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 post-test question. The plot width increases with the total answers. The white dot is the median, red ones the quartiles.



ICMC USP
SÃO CARLOS