Faculty of Technology and Society
Department of Computer Science and Media Technology

Master Thesis Project 15p, Spring 2017

Mixed-Initiative Procedural Generation of Dungeons Using Game Design Patterns

By
Alexander Baldwin & Johan Holmberg

Supervisors
José Font & Carl Magnus Olsson

Examiner
Johan Holmgren
Contact Information

Authors:
Alexander Baldwin
E-mail: alexander.d.baldwin@gmail.com

Johan Holmberg
E-mail: johan.holmberg@mah.se

Supervisors:
José Font
E-mail: jose.font@mah.se
Malmö University, Department of Computer Science and Media Technology

Carl Magnus Olsson
E-mail: carl.magnus.olsson@mah.se
Malmö University, Department of Computer Science and Media Technology

Examiner:
Johan Holmgren
E-mail: johan.holmgren@mah.se
Malmö University, Department of Computer Science and Media Technology
Abstract

Procedural content generation (PCG) can be a useful tool for aiding creativity and efficiency in the process of designing game levels. Mixed-initiative level generation tools where a designer and an algorithm collaborate to iteratively generate game levels have been used for this purpose – taking advantage of the combination of computational efficiency and human intuition and creativity. However, it can be difficult for designers to work with tools that do not respond to the common language of games: game design patterns.

It has been demonstrated that game design patterns can be integrated into PCG algorithms, but formally-defined and hierarchically-arranged game design patterns have not yet been used as a means of increasing gameplay-based control in mixed-initiative dungeon generators. We present a method for evolving dungeon rooms using multi-level game design patterns in the objective function of a genetic algorithm, as well as an instantiation of this method in a mixed-initiative dungeon design tool. Our results show that we are able to control the frequency and type of design patterns in generated rooms using pattern-related input parameters, enabling us to create dungeon rooms containing a wide variety of patterns on different levels of abstraction.

Results from a small-scale user study of professional game developers suggest that the use of game design patterns in mixed-initiative level design tools can be a promising way of providing a good starting point when designing a level, as well as offering meaningful gameplay-related feedback throughout the design process. We also identify challenges that will need to be faced if game design pattern-based mixed-initiative level design tools are to become a part of the game designer’s toolkit.
Popular Science Summary

Modern video game development is time-consuming and costly. As games become larger and consumers expect them to contain varied and frequently updated content, the burden on the creators of this content steadily increases. In the 1980s, games like Rogue and Elite pioneered algorithmic approaches to the automatic creation of game content, known as procedural content generation (PCG), allowing for game worlds vastly larger or more varied than those their developers could have created by hand. Since then, whole genres like the Rogue-inspired roguelike have been built around PCG and algorithmic approaches to game content generation are common in many areas of game development.

Recently, researchers have begun to explore the potential of PCG techniques as an aid to human game designers, rather than as purely automated processes. Combining the time-saving and raw computational power of PCG with the intuition and creativity of humans might allow us to take the best of both worlds and produce higher quality game content faster.

An open question that we address in this thesis is how best to facilitate the collaboration between human designers and PCG algorithms in the domain of game level generation. We argue that, in order to meet this goal, level generation algorithms should create content with clearly identifiable gameplay elements. Furthermore, the algorithm should be controlled in terms of these gameplay elements. One way of expressing gameplay formally is the concept of game design patterns, which refers to gameplay elements that occur frequently in games and it is through the use of these patterns that we approach the problem.

Our contribution is a new method for generating game levels that combines the idea of game design patterns with a mainstay of artificial intelligence techniques: the genetic algorithm. Genetic algorithms mimic the biological process of evolution, but, in our case, instead of evolving creatures to survive, we evolve game levels to contain interesting and diverse game design patterns. We demonstrate the use of this method in a tool allowing designers to iteratively evolve game levels, which was tested with a small number of professional game developers, whose feedback gives rise to a number of further challenges for future research on PCG-based game design tools.
Acknowledgements

We would like to thank our supervisors José and Carl for all of their advice and cooperation during the writing of this thesis. Thanks also to Steve for the many discussions we’ve had about game design patterns over the past few months. Finally, thanks to the people who tested Evolutionary Dungeon Designer.
## Contents

1 Introduction .................................................. 1  
  1.1 Research Goals ........................................... 1  

2 Related Work ................................................ 3  
  2.1 Procedural Content Generation .......................... 3  
  2.2 Search-Based PCG ........................................ 3  
  2.3 Game Design Patterns .................................... 4  
  2.4 Dungeons .................................................. 5  
  2.5 Generating Dungeons ..................................... 5  
  2.6 Mixed-Initiative Level Generation ..................... 6  

3 Research Method .............................................. 8  
  3.1 Design Science ............................................ 8  
  3.2 Evaluating Generators ................................... 10  
  3.3 Limitations & Threats to Validity ..................... 11  

4 Evolutionary Dungeon Designer ............................ 12  
  4.1 Overview .................................................. 12  
  4.2 FI-2pop GA ............................................... 13  
  4.3 Pattern Detection ........................................ 14  
  4.4 Spacial Micro-Patterns .................................... 15  
  4.5 Inventorial Micro-Patterns ............................... 18  
  4.6 Meso-Patterns ............................................. 21  
  4.7 Evaluating the Fitness of Feasible Individuals ...... 28  
  4.8 Evaluating the Fitness of Infeasible Individuals ...... 30  
  4.9 User Interface and Workflow in EDD ................... 31  

5 User Study ..................................................... 35  
  5.1 Study Design .............................................. 35  
  5.2 Results .................................................... 37  
  5.3 Conclusions ............................................... 42  

6 Experiments .................................................... 44  
  6.1 Micro-Pattern Generation ............................... 44  
  6.2 Meso-Pattern Generation ............................... 52  

7 Discussion and Conclusions ................................ 61  
  7.1 Future Work ............................................... 64  

References ......................................................... 66  

Appendix A User Study Questionnaire ....................... 70
<table>
<thead>
<tr>
<th>B.1</th>
<th>Introductory question</th>
<th>71</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.2</td>
<td>Questions on patterns</td>
<td>71</td>
</tr>
<tr>
<td>B.3</td>
<td>Questions on the mixed-initiative tool</td>
<td>71</td>
</tr>
<tr>
<td>B.4</td>
<td>Final question</td>
<td>72</td>
</tr>
</tbody>
</table>
1 Introduction

As the expectations of consumers increase, computer games are constantly becoming larger and more time consuming to produce [33]. A solution that has been suggested for this problem is the algorithmic creation of game content: procedural content generation (PCG). PCG has already been applied in many areas, both in the game development industry and in academic research for the generation of weapons, maps, levels and even the rules for entirely new games [33].

An area that has seen a lot of interest from researchers is the generation of dungeons, where dungeons are confined game levels usually consisting of a network of rooms and corridors filled with treasure and enemies [4]. While many approaches to dungeon generation have proven successful, each approach brings its own limitations. Constructive approaches rely on an algorithm that is carefully tuned before generation [36] – a task that can be difficult for game designers if there is not a clear mapping between familiar game concepts and the algorithm’s control parameters.

Numerous efforts have been made to make procedural generation of levels more designer-friendly, often focussing on using the concepts and language of game design [8,11,22,37]. These concepts can be described as game design patterns: “commonly reoccurring parts of the design of a game that concern gameplay” [2]. Game design patterns provide a common vocabulary with which to describe game elements that may be an aid to game designers if integrated into PCG level generation. So far, most of the use of gameplay-related concepts in level generation has treated these concepts as building blocks or inputs to generators, but Dahlskog et al. [5] argue that there may be more value in seeing game design patterns as objectives for generators and the authors describe a search-based approach to level generation using game design patterns. Recent work by Dahlskog et al. [4] catalogues game design patterns for dungeon levels at three different abstraction levels, with the goal of providing a level of detail that future generators can be built upon.

Search-based approaches to generating levels such as evolutionary algorithms can also be integrated into mixed-initiative generation tools where a game designer and a PCG algorithm work together to iteratively create levels based on desired gameplay, as shown by Liapis et al. in their application Sentient Sketchbook [23]. Such approaches may enhance overall creativity in the level generation process while giving designers more control. Attaining a better understanding of how this might be applied to the area of mixed-initiative dungeon generation using game design patterns is the motivation for this research and our goals are presented in the following section.

1.1 Research Goals

PCG as a tool for aiding in the creative process of designing levels is starting to be explored, but research shows that designers lack effective ways of controlling the process to produce desired results [38]. While gameplay-related concepts have been used in the generation of dungeons and other game levels, formally defined and hierarchically organised game design patterns have not yet been used as part of a mixed-initiative dungeon generator. A recent catalogue of dungeon design patterns compiled by Dahlskog et al. [4] can serve as the starting point for the construction of such a generator. As such, the goal of this research
is to investigate how game design patterns can be applied to the procedural generation of game dungeons and to evaluate the usefulness of such an approach to game designers. This goal gives rise to the following research questions:

**RQ1** How can game design patterns be used to control procedural generation of dungeons?

**RQ2** How can game design patterns in dungeons be detected algorithmically?

**RQ3** How can the use of game design patterns in a mixed-initiative generator help designers realise their design goals?
2 Related Work

In this section, we present an overview of recent literature relevant to our research. We start with a broad overview of PCG before moving on to our specific areas of interest: search-based methods for procedural level generation, mixed-initiative level generation and how PCG can be combined with game design patterns.

2.1 Procedural Content Generation

Developing computer games is expensive and the cost of developing mainstream games has increased almost constantly since computer games were invented. More developers are required and more man hours must be spent to satisfy the demands of consumers as graphical fidelity increases and game worlds become larger and larger. Much of this increased development time is spent on creating content, which includes most of what makes up games: levels, maps, art, stories, music, characters etc. [33].

Reducing the time (and hence cost) taken to develop computer games is one of the primary motivations for PCG: “the algorithmic creation of game content with limited or indirect user input” [33]. By automating the work traditionally done by artists and designers, development times can be reduced, potentially enabling much smaller development teams to compete with the games produced by large studios.

Besides cost reduction, there are numerous other potential benefits to using PCG. If content is generated in real-time as players play a game, the content can be tailored to suit the way they like to play, leading to a more personalised experience [40]. PCG could also enhance design creativity by suggesting options a human designer would not have thought of. This can be used in mixed-initiative content generation where a human designer ‘collaborates’ with a PCG tool to create content together. Additionally, some game types, such as roguelikes, are built around the idea of being highly replayable and rely on PCG as a means of continuously creating new content (in this case usually levels) to keep players entertained [33].

2.2 Search-Based PCG

Togelius et al. introduce the term search-based procedural content generation to describe a set of PCG techniques using search-based methods such as genetic algorithms [36]. The non-deterministic, guided evolution of the content makes for a diverse set of potential end results. These techniques allow for content to be generated and refined towards a predefined set of goals, using a construct known as a fitness function or evaluation function [36]. Typically, a set of content artefacts are generated, altered by means of non-deterministic mutation and interbreeding, and then evaluated according to the fitness function. Those individuals that are deemed fitter than their competitors are allowed to survive and form the basis for a new generation of artefacts. This process is repeated for a fixed number of generations. However, as Togelius et al. point out, there is no guarantee that a search-based approach will actually produce a good enough solution [36]. To combat this, a fairly high number of generations maybe be needed to lower the possibility of ending up with bad content.
Liapis et al. propose an approach to applying search-based PCG to the domain of map-based games, such as strategy games, and introduce a set of fitness criteria with corresponding fitness functions [22]. By combining these fitness functions with a playability criterion, they are able to produce well-balanced maps through constrained optimisation using a feasible-infeasible two-population genetic algorithm (FI-2pop GA) as the basis for their generator. Whereas a traditional genetic algorithm may discard invalid solutions, a FI-2pop GA maintains separate populations of feasible and infeasible individuals with the goal of preserving potentially high quality genetic material from infeasible individuals. After mutation and cross-breeding, these individuals may become feasible again and increase the diversity of the feasible population (and therefore also the diversity of generated maps).

The evaluation of map-based levels can pose a problem if a fitness function needs to evaluate the entire level with e.g. an A* search function, as the search space tends to grow exponentially rather than linearly [36]. For real-time or interactive applications, this may lead to impractically long running times. One efficient countermeasure used in the mixed-initiative tool Sentient Sketchbook [24] is to generate and evaluate low resolution levels, which are then rendered or expanded at a later stage in the process.

2.3 Game Design Patterns

Design patterns are usually described as solutions to commonly occurring problems [3, p.32] and have been applied in architecture as well as widely in software development. Björk and Holopainen argue that the traditional definition of design patterns is not appropriate for describing game design due to the focus on solving problems and instead define game design patterns as: “semiformal interdependent descriptions of commonly re-occurring parts of the design of a game that concern gameplay” [2]. Björk and Holopainen present a large collection of general game design patterns and a number of additional efforts have been made to collect patterns for specific game genres, including platform games [2], first person shooters [14] and role playing games (RPGs) [4].

Recent research has started to combine game design patterns with PCG. Pantaleev used game design patterns as inputs to a search-based algorithm with the goal of discovering new patterns for RPG skill systems [30]. Smith et al. used design patterns in a mixed-initiative tool for the design of 2D platform game levels, where the patterns were limited to those describing the geometry of the level’s terrain [34]. While the previous examples use design patterns as inputs to an algorithm or components to be combined, Dahlskog and Togelius [6] argue that, rather than using design patterns as building blocks for levels, they can be used as objectives for a PCG generator. The authors present a method of generating 2D platform game levels using an evolutionary algorithm, where fitness is based on the number of patterns found in generated levels.

2.3.1 Micro-, Meso- and Macro-Patterns

In their discussion of game design patterns, Björk and Holopainen suggest that patterns can form hierarchies, wherein a pattern is formed by the instantiation of a number of other patterns. These patterns can be considered to exist on higher or lower levels of
abstraction [2] p.37]. Dahlskog and Togelius expand upon this idea when describing the division of game design patterns into three levels of abstraction: micro-, meso- and macro-patterns [7]. In their work, micro-patterns describe thin level slices in a 2D platform game and can be seen as fundamental building blocks of levels, without much specific gameplay meaning (they are compared to tiles in dungeon generation, but later work by Dahlskog et al. [4] makes a distinction between tiles and micro-patterns, which can be composed of multiple tiles). A meso-pattern combines multiple micro-patterns into a larger structure the authors describe as a “feature which requires or affords some player action”, emphasising a stronger connection to the type of gameplay enabled or required by the level layout. Macro-patterns are sequences of meso-patterns that enable some kind of progression to be modelled into the gameplay of a whole level (for example increasing difficulty). The authors suggest that by taking into account meso- and macro-patterns, procedurally generated levels can feel more ‘designed’.

2.4 Dungeons

The dungeon is a popular level design archetype found in games spanning multiple genres including action-adventures and RPGs such as the Legend of Zelda [28]. van der Linden et al. describe dungeons as “labyrinthic environments, consisting mostly of interrelated challenges, rewards, and puzzles, tightly paced in time and space to offer highly structured gameplay progressions” [38]. In practice this is usually implemented as a set of rooms, connected by corridors or hallways [33]. Shaker et al. suggest that it is the constraints imposed by these interconnected quests and spaces that allow for sophisticated gameplay pacing and progression, which in turn makes the archetype appealing to game designers and PCG researchers [33].

Dahlskog et al. present a collection of game design patterns found in RPG dungeons, motivated by the need for finer granularity and more detail than normally found in collections of patterns when generating levels using search-based PCG [4]. As with Dahlskog’s previous work, patterns are divided into micro- meso- and macro-abstraction levels. An example of a micro-pattern is Corridor: a narrow vertical or horizontal sequence of ground tiles. An example of a meso-pattern is Special room: a room that contains some kind of interesting content, such as items or enemies. An example of a macro-pattern is Backtracking level: a sequence of rooms where the player must return back the same way after completing some objective. In this case, the micro-patterns represent low-level building blocks of dungeons, while meso-patterns and macro-patterns describe higher-level gameplay, such as (in the case of macro-patterns) the progression through a sequence of rooms or the layout of a whole dungeon. It is however not clear whether or not there can be considered a strict distinction between meso- and macro-patterns.

2.5 Generating Dungeons

Due in part to their inclusion in popular games and the unique design constraints they impose, dungeons have been a popular focus for PCG research and numerous different approaches have been presented for generating dungeon levels, including constructive methods (which produce a single output that is not evaluated and regenerated) like space
partitioning [33], agent-based methods [33], cellular automata [15] and grammars [8, 37]. According to van der Linden et al. [38], such constructive methods for level generation require that designers fully understand the generation process in order to map control parameters to gameplay goals – something that is not always straightforward. This problem also affects search-based generation methods, since designers must understand the generation process in order to decide which fitness function to use. However, a recent survey of procedural dungeon generation methods by van der Linden et al. suggests that new approaches in search-based dungeon generation using gameplay-informed fitness functions can offer PCG solutions that are easier for game designers to use [38].

van der Linden et al. aim to give game designers more control over the PCG process by using gameplay-based graph grammars to generate dungeons [37]. The authors argue that the spaces in games should result from the desired gameplay, not the other way round and they achieve this by allowing the designer to construct gameplay grammars, where gameplay refers to actions the player can perform in the game. van der Linden et al.’s approach is similar to one described by Dormans, who uses generative graph grammars made up of game-related terms to generate missions, which are then mapped to levels using a shape grammar [8].

While the aforementioned grammar-based approaches to dungeon generation rely on grammars created by the designer, Font et al. describe how a search-based approach can be used to evolve grammars for generating dungeons [9]. A two-step approach is used, first generating a high-level acyclic graph of the level and then more detailed cyclic graphs where each node represents one of the dungeon’s rooms. The evolutionary algorithm evaluates the fitness of generated individuals using gameplay-related features such as progression: the level should offer a higher density of rewards as the player progresses. The designer is however not given control over the generation process beyond choosing the number of rooms in each section of the dungeon.

2.6 Mixed-Initiative Level Generation

While automatic PCG may be sufficient in some cases, there exists a desire for human input in the process [30, 36, 39]. A computer, not being held back by limited human imagination, has the potential to produce large amounts of novel material. Togelius et al. note that this novel material can help inspire designers to improve their products [36]. Lubart describes a class of human-computer relationship as Computer as colleague [25], in which computer programs contribute with new ideas. The role of the human agent in such a relationship is to provide feedback and point out which directions to go forward with. The developers behind the dungeon-based RPG Dwarf Quest [10] used a mixed-initiative tool for generating the game’s levels [16], indicating a willingness from parts of the game development community to embrace this kind of tool.

Smith et al.’s platform game level design tool Tanagra [34] emphasises the iterative nature of mixed-initiative design processes – an approach that is shared with another mixed-initiative level generator, Sentient Sketchbook [22], a tool using evolutionary algorithms to generate maps for strategy games. In Sentient Sketchbook, the fitness functions used are inspired by game design patterns such as resource fairness and base safety. These
concepts, which should be familiar to designers of strategy games, provide a meaningful understanding of the fitness of a generated level without requiring an understanding of how the algorithm is implemented. By combining this with an interactive level authoring tool, designers can actively and iteratively influence the types of game design pattern present in the resulting map.
3 Research Method

In this section, we discuss the research methods we have used to answer our research questions, starting with a brief overview of the general research approach, design science, followed by discussions of the specific methods that were used for the production and evaluation of artefacts relevant to our research.

3.1 Design Science

Our research uses the design science research approach. March and Smith describe the process of design science as consisting of two basic activities: build and evaluate [26]. Building is concerned with the creation of artefacts, which may be constructs, models, methods or instantiations. Our research is concerned with two artefacts: a method and an instantiation. The method is the application of dungeon game design patterns in the objective function for a search-based PCG algorithm. The instantiation is a working system demonstrating the use of the method, in this case a mixed-initiative tool for creating dungeon levels.

Hevner et al. situate design and creation in a framework for information systems research, where the construction and evaluation of artefacts is informed by the environment and the knowledge base [12]. The environment consists of people, organizations and technology and in Hevner’s model, the needs and affordances of these groups form an input to the research process, ensuring that the research is relevant. The knowledge base consists of the foundations and methodologies existing in the scientific literature, upon which design and creation should be based in order to support a rigorous research approach. The results of the research should in turn add to both the environment, through appropriate application, and the knowledge base, through the creation of new knowledge. Since PCG research is balanced between reacting to the real needs of game developers (and game consumers) and drawing from a robust and extensive knowledge base of research on AI and algorithms, we consider the design science approach ideally suited to our research.

In our case, we have provided an overview of the relevant parts of the knowledge base in section 2 and it is this research that the design, creation and evaluation of our artefacts is based upon. We also described in section 2 the motivations for the creation of new PCG methods in terms of the needs of game designers. The design science paradigm as described by Hevner fits our research well, since we actively address the needs of the environment by presenting game designers with a new approach for level design based on their needs and expertise. While addressing the needs of the environment, the research also contributes a new method to the knowledge base upon which further research can be based, as well as an instantiation that can be used to support or facilitate this future work.

An alternative research approach may have been to augment the literature review with an exploratory case study at one or more game development companies, in order to gather more detailed and in-depth data about the game development industry’s needs. This could then be used as an input to the design science process. However, we chose to rely on the analyses of the needs presented in recent literature in order to limit the scope of the study. Another approach that could further strengthen the relevance of the work
to the game development industry is to conduct action research in collaboration with a game development company. However, this presents a number of challenges in terms of organising such a collaboration in such a way that it is valuable to all involved parties and may also limit generalisability by tying the study to a particular company and/or game.

In Hevner’s framework, the creation and evaluation of artefacts is an iterative process of assessment and refinement [12]. The work described in this paper constitutes results of three major iterations of such an iterative process:

1. The first major iteration consisted of performing a literature review in order to catalogue the state of current research on dungeon generation. This was used to inform the direction taken in the later iterations of the research project, by identifying a gap in the existing research – the lack of studies of multi-level game design patterns in search-based dungeon generation – and finding suitable methods to address that gap. We present the results of this literature review in section 2 but the literature also informs the contents of the entire text. In addition to this original iteration, additional literature searches have been conducted throughout the entire project to ensure our work reflects the state of the art in PCG research.

From a starting point of a number of papers already familiar to us [9, 23, 4], we isolated a number of search terms, including “game design patterns”, “dungeons”, “search-based PCG”, “mixed-initiative” and combinations of these. We conducted a search in the ACM and IEEE databases for these terms and, based on an initial reading of the results, identified new search terms, including “FI-2pop GA” and “expressive range”. We identified relevant literature by conducting an initial reading of abstracts and introductions/conclusions, followed by a thorough reading of those relevant papers, which in turn revealed more literature.

2. The second major iteration consisted of the production of a first version of our instantiation, containing a subset of our final functionality (an implementation of our game design pattern-based generation method using only micro-patterns). We then evaluated the capability of our instantiation to generate dungeons featuring our target game design patterns. We present the results of this evaluation in section 6.1.

3. The third and final major iteration consisted of the production of a complete instantiation of our method (encompassing both micro- and meso-patterns) in a tool with a mixed-initiative workflow. We evaluated the instantiation through both a qualitative user study, presented in section 5 and a second-set of quantitative experiments, presented in section 6.2. The final method and instantiation composed of the work carried out in iterations two and three are presented in section 4.

Within each of these major iterations, we conducted smaller, informal iterations, consisting mainly of development and not evaluation. Since these steps were not formally evaluated, we opt not to present their results here.
3.2 Evaluating Generators

The second of March and Smith’s basic activities is evaluation [26]. How best to evaluate the success of approaches to PCG is an open question. As recently as 2014, Horn et al. described the current state of evaluating PCG generators as “largely ad-hoc”, stating that the evaluation of many PCG artefacts lacks rigour and cannot be generalised [13]. Dahlskog provides insight into the question of evaluation, with a focus on game design pattern-driven PCG systems, by asking “How can pattern-based PCG systems be evaluated?” [3]. Dahlskog’s research applies two primary evaluation methods: user studies [3, pp. 225-241] [6] and expressivity metrics [13]. These approaches are also echoed by Shaker et al. in their discussion of methods for evaluating content generators [33], though they argue that “regardless of the method followed, generators are evaluated on their ability to achieve the desired goals of the designer”. As such, we evaluate our generator from the perspective of how well it is able to meet our goal of generating dungeon rooms based on the types and quantities of game design patterns they contain.

3.2.1 Quantitative Evaluation

Expressive range is a measure of the size of the space of levels that the generator can potentially create [33]. Evaluation of expressive range consists of choosing metrics (most commonly linearity and leniency), generating many levels, evaluating them according to these metrics and then plotting the resulting values as a heat map. This heat map allows the expressivity of the generator to be visualised as well as showing how changes to the generator’s input parameters affect the space of potential levels. Game design pattern-based metrics have not however been heavily explored in expressive range analysis, except by Horn et al. who use the metrics pattern density, based on the number of patterns found in a level, and pattern variation, based on the diversity of the patterns found. This provides a method for evaluating the distribution of game design patterns in the space of generated levels.

In our case, we do not consider it a priority to perform an expressive range analysis, since we are more interested in whether or not we can control the occurrence of design patterns rather than examining the space of all potential rooms we can generate. As such, we define a series of experiments in which we observe the proportions and/or numbers of game design patterns generated using different configurations of input parameters, as well as the fitness of generated rooms. In doing this, we are also able to make conclusions about the expressivity of the generator: if we can show that we are able to control the occurrence of design patterns, it follows that we are able to generate diverse rooms by making use of this control.

Specific details of the design and results of these experiments can be found in section 6.

3.2.2 User evaluation

User evaluation is Shaker et al.’s second method of evaluating generators [33]. While Shaker et al. suggest that users in this case should be players, in the case of mixed-initiative tools it may be more interesting to study designers as done by Liapis et al.
their user evaluation of Sentient Sketchbook [23]. In a mixed-initiative tool, the designer and the generator are both responsible for the generated level, so it may be difficult to separate the influence of the designer from the generator when evaluating with players. In our case, we are interested in the level of control the generator offers the designer as well as how it enables their design goals to be achieved, so it is natural to study how designers interact with the tool.

While our quantitative evaluation is mainly concerned with whether or not our method is able to control the generation of rooms that contain specific game design patterns, our user study is necessarily more exploratory in nature. While some studies of mixed initiative design tools have been conducted [23, 34], there remain many open questions about the requirements of such a tool and how it can fit into the work process of a professional game designer. In particular, it is not clear what level of control a designer requires in order to be productive when using a mixed-initiative design tool. Since some of the criticism of today’s PCG level generation approaches is directed at the complexity of generation algorithms and the process of tweaking their parameters to produce desirable results [38], we have designed a user study which limits the amount of explicit access to the parameters of our underlying algorithm and focusses instead on how the software and designer can iteratively work together to generate creative levels with interesting gameplay.

Since the area is relatively unexplored, we have opted for a primarily qualitative approach to user evaluation, with the goal of providing a foundation on which future work can be built. As such, we conducted a small scale user study with professional game developers, including a mix of designers, programmers and artists, with the goal of learning about how our generation approach might be useful for users that (though familiar with professional game development in general) have varying degrees of experience with level design in particular. The details of this study are described in detail in section 5.

3.3 Limitations & Threats to Validity

While our aim is to produce results that are as generalisable as possible, there are of course limitations to the conclusions we will be able to draw from our research. Since we have studied the generation of dungeons, we do not claim to be able to draw conclusions about the procedural generation of other artefacts or even other types of game levels, though we hope to provide the foundation for similar investigations in these areas.

Our study is closely linked to the field of computational creativity, which has been widely studied and is relevant. However, since we are studying the topic of mixed-initiative level generation, it is unlikely that our results could be seen as a direct contribution to the field of computational creativity. This may however be an avenue for future research.

Hevner’s design science framework addresses the validity of design and creation based research by demanding that the creation and evaluation of research artefacts is grounded in existing theories and methods from scientific literature [12]. As such, we have based the development of our artefacts on the existing literature presented in section 2 which ensures that our prototype is built upon sound principles.

Further threats to validity associated with the evaluation of our artefacts are presented in conjunction with those evaluations in sections 5 and 6.
4 Evolutionary Dungeon Designer

In this section we discuss the artefacts developed during our design research process: a method for the procedural generation of dungeon rooms using game design patterns and an instantiation of that method in the form of a mixed-initiative dungeon design tool: Evolutionary Dungeon Designer (EDD).

4.1 Overview

EDD is the continuation of work done by Font et al. [9] on generating high-level dungeon layouts by evolving grammars, as discussed in section 2.5. While Font et al.’s Evolutionary World Designer generates dungeon layouts, it does not generate the contents of individual rooms. EDD is intended to fill this gap. As such, we do not generate entire dungeons, but rather the layout of individual rooms in a connected-rooms style dungeon like those found in The Legend of Zelda [28] or The Binding of Isaac [27]. This multi-step approach can be compared to Liapis’ two-step evolutionary dungeon generation method [20].

A room in EDD is defined by a rectangular $m \times n$ grid of tiles chosen from six types: floor, wall, enemy, treasure, entrance and door, shown in fig. 1. Walls are barriers to movement and are impassable, whereas all other tile types are considered passable (a game character can walk on them). A given room has between one and four doors, of which one is the entrance: the door a character will enter through when reaching the room for the first time. For a room to be playable, there must exist paths between the entrance and all other doors, treasures and enemies and the room must contain at least one enemy and treasure.

![Figure 1: The graphical representation of the six different tile types currently used to render images of the generated rooms.](image)

EDD is a Java application consisting of two main components: a generator that, given a set of input parameters, generates dungeon rooms according to the method we describe in subsequent sections and a graphical user interface that enables users to iteratively generate, edit, save and load rooms using a mixed-initiative workflow described in section 4.9. Owing to a high level of autonomy between the components, we are also able to run the generator separately from the GUI, including running multiple instances in parallel – allowing for increased efficiency when performing the experiments described in section 6.

Additionally, the tool contains components for saving generated rooms, rendered images of rooms and data about the generation process at each step of that process. Rooms saved this way are serialised as text files, in which the rooms are saved as a grid. In this
grid, each tile is represented by a hexadecimal digit. This allows us to represent up to sixteen different tile types, making the solution expandable to a certain degree. Serialising rooms allows us to analyse these rooms with external tools or save and load rooms for inspection in the GUI.

The central component of EDD, the generator, and the method it employs are described in the subsequent sections.

4.2 FI-2pop GA

The generator evolves rooms through constrained optimisation, using an implementation of the feasible-infeasible two population genetic algorithm (FI-2pop GA) first presented by Kimbrough et al. [18]. Unlike a standard genetic algorithm, a FI-2pop GA maintains two populations of solutions (rooms in our case): feasible and infeasible. Instead of discarding rooms that do not fulfil our playability constraint (that all doors, treasures and enemies are reachable from the main entrance), they are instead moved to the infeasible population where they are separately optimised to bring them closer to feasibility. The goal of this approach is to increase the overall diversity of the population, since maps that do not satisfy the playability constraint may still contain useful data (such as design patterns), that may later re-enter the feasible population.

Our implementation of the FI-2pop GA is largely based on the method Kimbrough et al. describe [18], though we instead use two-point crossover and a very high rate of mutation: 90%. Each mutation has a 20% chance to rotate the room 180°, inspired by Liapis et al. [23], or an 80% chance to mutate a single tile into a random different tile type. The rotation mutation serves to increase the chance of rotationally symmetrical features appearing in maps and, as discussed in section 6.1.6, it functions as expected. Additionally, we end the algorithm after a fixed number of generations, rather than on the basis of a fitness threshold, to ensure that the application’s response time is consistent across different sets of input parameters. In an interactive application like EDD it may sometimes be better to present a suboptimal solution than to run for more generations.

An individual’s chromosome consists of a two-dimensional array of integers, in which each integer corresponds to a tile type (so floor = 0 and so on, see fig. [2]). At the start of

![Figure 2](a) A chromosome and (b) its corresponding phenotype.

![Figure 2](a) A chromosome and (b) its corresponding phenotype.

0 2 0 0 0
0 0 1 1 1
4 0 0 0 5
0 1 0 3 0
0 1 4 0 0

Figure 2: (a) A chromosome and (b) its corresponding phenotype.
generation, feasible and infeasible populations are randomly generated based on a given average proportion of each tile type (mainly floor to make the playability constraint easier to fulfil).

The feasible population is optimised to contain game design patterns through the method described in the subsequent sections and full descriptions of the algorithm’s fitness functions can be found in sections 4.7 and 4.8.

4.3 Pattern Detection

In order to optimise a room for the occurrence of game design patterns, we must first be able to detect these patterns. We limit ourselves to the study and detection of micro- and meso-patterns, while excluding macro-patterns, since macro-patterns are generally on the scale of entire levels (or large parts of levels) and encapsulate larger sections of gameplay than can be found in a single dungeon room. As meso-patterns are built upon combinations of micro-patterns, detecting these patterns is a process in multiple stages in which we first detect micro-patterns and then use these to identify the meso-patterns that contain them.

In addition to the distinction between micro- and meso-patterns, we further differentiate between micro-patterns that deal with the layout of space (in our case, floor tiles) in the level and those that deal with the contents of that space – entities or objects that can occupy floor tiles. In the former category – *spacial micro-patterns* – we limit ourselves to considering the most fundamental of Dahlskog et al.’s micro-patterns *corridor*, *connector* and *room*, though we refer to room as *chamber* to avoid confusion with our previous use of room. *Space* is also one of Dahlskog et al.’s dungeon patterns, but we do not include it in our discussion of pattern detection since a space is equivalent to a passable tile. The second category – *inventorial micro-patterns* – consists of our remaining passable tile types: *door*, *treasure* and *enemy*. We describe these patterns and the process of detecting them in sections 4.4 and 4.5.

Meso-patterns encapsulate more complex types of gameplay than micro-patterns and consist of combinations of multiple micro-patterns (or other meso-patterns). For example, while a chamber micro-pattern denotes an open area giving the player more freedom of movement, the dead-end meso-pattern might reveal that a whole section of the room containing several chambers does not need to be visited in order to traverse the room (which may or may not be desirable depending on the designer’s goals and/or the type of content found in the dead end). We discuss meso-patterns and their detection in section 4.6.

Due to a lack of detail in Dahlskog et al.’s pattern definitions, we have redefined or amended each pattern with extra detail to allow consistent detection by our generator. Each detected pattern is also assigned an associated *quality* value between 0 and 1 as a measure of how well it conforms to some desired control parameters. Our new pattern definitions, quality metrics, and detection algorithms are presented below.
4.4 Spacial Micro-Patterns

Spacial micro-patterns describe structures in the layout of floor and wall tiles in a dungeon room and give some indication of the type of gameplay that might be found in those spaces. For example, a room containing many corridors and few chambers (open spaces) may feel claustrophobic and make retreating from combat difficult. Furthermore, if these corridors are made up of many twists and turns instead of being long and straight, there is potential for enemies to hide behind corners and it may be harder to fight with ranged weapons. The distribution of these patterns in a room also has a direct impact on the aesthetics of the room. For example, if the dungeon is part of a man-made structure, regular, rectangular spaces and straight corridors may be appropriate, while more irregular spaces might fit better in a naturally-occurring network of caves.

![Figure 3: Examples of each micro-pattern detected by the generator.](image)
(a) Chamber  (b) Corridor  (c) Intersection  (d) Intersection  (e) Turn

4.4.1 Corridor

According to Dahlskog et al. “A Corridor is a series of horizontal or vertical [passable tiles]. The end points of a Corridor can be connected to other Spaces.” [4]. We extend this definition to assume that the series of passable tiles is at least two tiles long and is enclosed by impassable tiles (walls) on either side. A three tile long example of a valid corridor is shown in fig. 3(b). Corridors are detected as follows:

- Mark all passable tiles as potential corridors if either the tiles to the north and south are both impassable or the tiles to the east and west are both impassable.
- For each potential corridor tile, find all connected (adjacent) corridor tiles using a flood fill. If there are at least two, these constitute a corridor, otherwise, the tile is not part of a corridor.
Using the function $Area(p)$, which returns the number of tiles included in a pattern (for a corridor, being one-dimensional, this is equivalent to the length), the quality $Q(p)$ of a corridor $p$ is calculated as

$$Q(p) = \min \left\{ 1.0, \frac{Area(p)}{T_{\text{corridor length}}} \right\}.$$  \hspace{1cm} (1)

where $T_{\text{corridor length}}$ is the user-defined target minimum length for a corridor, so only corridors at least as long as this are assigned maximum quality. Using this quality measure, we are afforded some control over the length of corridors in generated rooms. Consequently, we can choose to, for example, generate rooms containing many short corridors or a few long corridors.

### 4.4.2 Connector

According to Dahlskog et al. “A Connector is a 1 [passable tile] long Corridor that is used to let passages in dungeons turn or allow intersections by being connected to other Spaces.” \cite{4}. This conflicts with our definition of a corridor, which cannot be only one tile long. We remain true to the spirit of Dahlskog et al.’s definition and still consider a connector a type of corridor, while identifying two specific classes of connector with slightly different properties: *turn* and *intersection*. A turn is a passable tile with exactly two passable neighbours in orthogonal directions, see fig. 3(e). An intersection is a passable tile with exactly three or four passable neighbours to the north, east, south or west, see fig. 3(c) & 3(d).

The quality of a connector is determined by its type – intersection or turn – and all connectors of a particular type have the same quality. The quality of each type is a user-defined parameter, allowing the user to balance the ‘straightness’ of corridors by rewarding one connector type more or less than another. The impact of such balancing can be seen in fig. 4.

### 4.4.3 Chamber

According to Dahlskog et al. “A Room consists of several [passable tiles] but [is] wider than a Corridor and allow[s] for more freedom in movement.” \cite{4}. As mentioned above, we refer to this concept as *chamber* rather than room. A more rigid definition is given by Ashlock et al. \cite{1} in the form of a chamber identification algorithm. Ashlock et al. define a chamber as a set of passable tiles, consisting of a core block of 3 by 3 tiles, see fig. 3(a), which is iteratively expanded. Any passable tile with three or more neighbours already in the chamber is also considered to belong to that chamber. Given the definition offered by Ashlock et al., detecting chambers is a straightforward process. We detect chambers by locating a core block and then expanding that block by using a four-way flood fill algorithm.
Figure 4: Rooms generated with different quality parameters for turns and intersections. (a) shows an optimal solution for intersection quality 0.95 and turn quality 0.05. (b) shows an optimal solution for intersection quality 0.05 and turn quality 0.95 (note that these do not need to sum to 1).

Figure 5: The impact of different squareness values for three rooms with target area 25 and a target chamber ratio of 1.
The quality of a chamber $p$ is determined by a linear combination of two functions, $Squareness(p)$ ($BoundingBoxArea(p)$ returns the area of the rectangular bounds of a pattern)

$$Squareness(p) = \frac{Area(p)}{BoundingBoxArea(p)}.$$  \hfill (2)

which is a measure of the proportion of the tiles in a chamber’s bounding box that are also in the chamber – and $Size(p)$

$$Size(p) = \max\left\{0, 1 - \left|\frac{Area(p)}{T_{chamberarea}} - 1\right|\right\},$$ \hfill (3)

which is a measure of how far a chamber’s area is from a user-defined target area. Combined, with user defined weights $w_{squareness}$ and $w_{size}$, the quality becomes

$$Q(p) = w_{squareness} \cdot Squareness(p) + w_{size} \cdot Size(p).$$ \hfill (4)

Three results of varying the value of $w_{squareness}$ (and, consequently, $w_{size}$, since the weights must sum to 1) for a constant target area are shown in fig. 5. A lower weight tends to create more roughly shaped chambers that might be suitable if the room is part of a complex of underground caves, whereas high squareness values tend to create straight edges that more closely resemble man-made structures. Figure 6 shows three rooms generated with varying chamber target areas, demonstrating the potential for controlling the size of the open areas in a room.

4.5 Inventorial Micro-Patterns

In addition to spacial micro-patterns, we define inventorial micro-patterns as entities or objects that exist in spaces, by occupying floor tiles (and, as such, will belong inside other micro-patterns). In our case, these correspond to a room’s passable tile types: treasure,
enemy and door. Since they are a fundamental part of our rooms’ chromosomes, these patterns do not need to be detected, but the position and frequency at which they are generated will still be influenced by their quality metrics as for all our other patterns.

4.5.1 Treasure

While it does not map directly to any particular micro-pattern Dahlskog et al. describe, we define treasure as some kind of valuable item that can be collected by the player. As such, it could represent a wide array of different rewards, such as currency, consumable items, weapons or armour. This corresponds most closely to Dahlskog et al.’s container: “Containers allow caches of Items to be accessed from the Tiles they are placed in.” [4]

For treasure, we use quality to govern the approximate overall proportion of floor tiles in the room that contain treasure. The positioning of these treasures is then determined by the quality of other patterns, such as door, treasure chamber and guarded treasure. Quality for a treasure \( p \) is calculated as the extent to which the proportion of treasure in the room deviates from a user-defined target proportion, divided by the number of treasures in the room:

\[
Q(p) = \frac{1}{N_T} \left( 1 - \frac{1}{N_T \max\{T_{treasure}, 1 - T_{treasure}\}} \left| \frac{N_T}{N_P} - T_{treasure} \right| \right). \tag{5}
\]

\( N_T \) is the number of treasures in the room, \( N_P \) is the number of passable tiles in the room and \( T_{treasure} \) is the user-defined target treasure density.

4.5.2 Enemy

Dahlskog et al. define enemies as “Agents whose primary behavior is to attack players’ avatars.” [4] As with treasure, for enemies we let the quality govern only the proportion of enemy tiles in the room and leave the positioning of these enemies to other patterns: door, ambush, guard chamber and guarded treasure.

As such, we calculate the quality of an enemy in the same way as a treasure:

\[
Q(p) = \frac{1}{N_E} \left( 1 - \frac{1}{N_E \max\{T_{enemies}, 1 - T_{enemies}\}} \left| \frac{N_E}{N_P} - T_{enemies} \right| \right). \tag{6}
\]

\( N_E \) is the number of enemies in the room, \( N_P \) is the number of passable tiles in the room and \( T_{enemies} \) is the user-defined target enemy density.

4.5.3 Door

Dahlskog et al. define a door as “a barrier that has two states: “open” or “closed”. Open doors are Passable and Seethru while closed door[s] are neither. Connection to other Spaces are typically in a north-south or west-east direction.” [4] In our case, doors act purely as transition points to other rooms and all doors can be considered open. We do however make the distinction between a room’s entrance, through which the player will enter the room for the first time and other doors, since, while it might be appropriate to
place an enemy guarding a door, it might be less desirable to meet an enemy immediately upon entering a room.

Since door positions are fixed for an evolutionary run, doors are not actively affected during the generation process, but the quality of a door may affect the generation of other patterns in the room. When calculating the quality of doors, we are only concerned with the entrance – all other doors are assigned maximum quality. The entrance quality is defined as a linear combination of four separate measures: entrance safety quality, entrance greed quality, average treasure safety quality and treasure safety variance quality. Each feature is optimised to decrease its distance from a target value, specified by the user.

We use entrance safety as a representation of the size of the largest circular area centred on the main entrance that contains no enemies. Easier rooms provide larger safe areas, while harder rooms usually place enemies next to the main entrance, thus providing tiny safe areas. The safe area $Safe(p)$ returns the number of tiles traversed by a flood fill algorithm starting at a door $p$ before an enemy is found. We use this to calculate the entrance safety quality as

$$Q_{esafe}(p) = 1 - \left| \frac{Safe(p)}{NP} - T_{esafe} \right|,$$

where $NP$ is the number of passable tiles in the room and $0 \leq T_{esafe} \leq 1$ is the user-defined target safe area density.

Entrance greed is analogous to entrance safety fitness, but for treasure instead of enemies and serves to control how close to the main entrance treasures will be placed. $Greed(p)$ returns the number of tiles traversed by a flood fill algorithm starting at a door $p$ before a treasure is found. We calculate the entrance greed quality as

$$Q_{egreed}(p) = 1 - \left| \frac{Greed(p)}{NP} - T_{egreed} \right|,$$

where $0 \leq T_{egreed} \leq 1$ is the user-defined target entrance greed.

The treasure safety $TreasureSafety(t)$ measures how safe a treasure tile $t$ is, with regards to the distance to the main door $d_{t,d}$ (calculated using the A* search algorithm) and to every enemy tile $d_{t,e}$ in the room. A treasure can be considered safe for the player to pick up when it is close to the main entrance and far from every enemy in the room. This is calculated as

$$TreasureSafety(t) = \min_{1 \leq e \leq NE} \left\{ \max \left\{ 0, \frac{d_{t,e} - d_{t,d}}{d_{t,e} + d_{t,d}} \right\} \right\}.$$

This metric has been adapted from the safety metric described by Liapis et al. [22], translating concepts from strategy maps (resources and player bases) to dungeon rooms (entrance and enemies). In this implementation, $TreasureSafety(t)$ provides a value closer to 0 the closer treasure $t$ is to the enemies and the further it is from the main entrance. When the treasure is close to the main entrance and far from enemies, the value approaches 1.
After calculating $TreasureSafety(t)$ for every treasure tile in the room, we calculate the arithmetic mean $\overline{TreasureSafety(t)}$ and the variance $\sigma^2$. The former provides the average challenge for the players to get the treasures, which is indicative of the average level of difficulty for the room. The latter indicates whether there is a progression in the challenge level shown by treasures in the room. Higher variances show that the treasure safety values for a given room are dispersed around the mean, indicating higher levels of progression.

Finally we obtain the average treasure safety quality and the treasure safety variance quality as

$$Q_{atsafe}(p) = 1 - \left| \overline{TreasureSafety(t)} - T_{atsafe} \right|$$

$$Q_{vtsafe}(p) = 1 - \left| \sigma^2 - T_{vtsafe} \right|,$$

where $T_{atsafe}$ and $T_{vtsafe}$ are the user-defined target values for the average and the variance, respectively.

Finally, these four quality metrics are combined to give the quality of an entrance $p$ as:

$$Q(p) = \frac{1}{5}Q_{esafe}(p) + \frac{1}{5}Q_{egreed}(p) + \frac{1}{5}Q_{atsafe}(p) + \frac{2}{5}Q_{vtsafe}(p).$$

### 4.6 Meso-Patterns

In order to detect meso-patterns, which are defined by how micro-patterns or other meso-patterns relate to each other in space, or by what inventorial micro-patterns they contain, we require a means of analysing these properties of the detected micro-patterns. To do this we construct a pattern graph (using a breadth-first search starting at the room’s entrance), where vertices are spacial micro-patterns and edges represent adjacency between the spacial patterns’ tiles. The constructed graph (see fig. 7 for a visual representation of one such graph) can be cyclic and two vertices (patterns) may be connected by multiple edges, signifying that there exist multiple direct paths between the two patterns. Each edge is weighted according to the number of adjacent tiles and will generally have a weight of either one or two. All inventorial micro-patterns are associated with the spacial micro-patterns that contain them, but this relationship is not stored in the pattern graph.

By analysing the pattern graph, we can draw interesting conclusions about, for example, the types of paths by which the room can be traversed. The use of the pattern graph to detect meso-patterns is described in following sections, in particular section 4.6.4 and section 4.6.5. While we define micro-patterns by the tiles they contain, meso-patterns are instead defined by the micro-patterns or other meso-patterns they are constructed from (e.g. a treasure chamber consists of a chamber and at least two treasures). Detected meso-patterns are highlighted by icons representing the meso-pattern type, such as a treasure chest for a treasure chamber. The full set of icons is shown in fig. 8.

The first three meso-patterns we describe are all variations of Dahlskog et al.’s *special rooms* pattern, defined as “[chambers] created together with specific content such as Items
Figure 7: The pattern graph is visualised in EDD as a collection of colour-coded nodes (red for corridors, yellow for connectors, blue for chambers and grey for areas not included in a pattern) corresponding to spatial pattern locations, connected by black lines that indicate adjacency. In the lower right of the image, a thicker edge of weight two can be seen indicating that the connected chambers have two adjacent tiles. Meso-pattern icons are overlaid on nodes where necessary: in this case a treasure chamber and a guard chamber at bottom centre and bottom right, respectively. Nodes marked with a black dot are in dead ends and, as such, do not need to be visited in order to move between the room’s doors.

Figure 8: Icons shown in EDD to represent the presence of four of our five meso-pattern types. The icons are drawn in the location of the chamber contained in the pattern or, in the case of (d) Guarded Treasure, which includes multiple chambers, the icon is drawn in the included treasure chamber. Note that dead ends are not represented by an icon, since they can cover a large area of the map. They instead have an alternative visualisation shown in fig. [12]
or Enemies in them as well as possibly having restrictions on access to them.” [4]. All three
are detected in the same fashion: the generated room is processed to assign inventorial
micro-patterns to the spacial patterns containing them. Then, all chamber micro-patterns
are examined to determine whether or not they match the criteria (described below) for
one of the meso-patterns.

![Figure 9](image)

(a) shows a room that contains a treasure chamber: a chamber containing at
least two (and in this case exactly two) treasures. (b) shows the pattern overlay in EDD
for this room, indicating the treasure chamber with a treasure chest icon.

### 4.6.1 Treasure Chamber

We define a **treasure chamber** as a chamber containing two or more treasure tiles and no
enemies. While this can be interpreted as a chamber containing multiple discrete treasures,
this pattern could also be used to denote a room containing a single particularly valuable
or useful item and could be seen as similar to The Binding of Isaac’s item rooms, which
are guaranteed to contain a special item for the player to collect [27].

Quality for treasure chambers is defined by how close the number of treasures in the
chamber is to a user-defined target number, $T_{chambertreasure}$ and is calculated (for a treasure
chamber $p$) as:

$$Q(p) = \max \left\{ 0, 1 - \left\lfloor \frac{N_T - T_{chambertreasure}}{T_{chambertreasure}} \right\rfloor \right\}.$$  \hspace{1cm} (13)

By adjusting the target number of treasure tiles in the whole room and the desired
number of treasures in a treasure chamber, the user is given some control over how treasure
is distributed throughout the room.
4.6.2 Guard Chamber

We define a guard chamber as a chamber containing two or more enemy tiles and no treasures. As with the treasure chamber pattern, this could be interpreted literally as a number of discrete enemies or be used to represent particularly challenging enemies (such as bosses or mini-bosses), depending on the number of enemies in the chamber.

Quality for guard chambers is defined by how close the number of guards in the chamber is to a user-defined target number, $T_{chamberenemies}$ and is calculated (for a guard chamber $p$) as:

$$Q(p) = \max \left\{ 0, 1 - \left| \frac{N_E - T_{chamberenemies}}{T_{chamberenemies}} \right| \right\}. \tag{14}$$

4.6.3 Ambush

We define an ambush as a chamber containing both the room’s main entrance and at least one enemy. Conceptually, this represents a room in which the player will have to engage in combat (or flee) immediately upon entering and, depending on the number of enemies, may significantly increase the challenge of the room.

Quality for ambushes is defined by how close the number of enemies in the chamber is to a user-defined target number, $T_{ambushenemies}$ and is calculated (for an ambush $p$) as:

$$Q(p) = \max \left\{ 0, 1 - \left| \frac{N_E - T_{ambushenemies}}{T_{ambushenemies}} \right| \right\}. \tag{15}$$

![Figure 10](image-url)

Figure 10: (a) shows a room that contains a guard chamber: a chamber containing at least two (and in this case three) enemies. (b) shows the pattern overlay in EDD for this room, indicating the guard chamber with a shield icon.
Figure 11: (a) shows a room that contains an ambush: a chamber containing at least one (and in this case exactly one) enemy and the room's entrance. (b) shows the pattern overlay in EDD for this room, indicating the ambush with an icon depicting three enemies behind a shield. Note that the room also contains a treasure chamber on the left.

4.6.4 Dead End

Dahlskog et al. define dead ends as “locations from which players must move through previously explored areas” [4]. While this definition invites a number of potential interpretations, we consider a spacial micro-pattern to be part of a dead end if and only if every path from that pattern to the room’s critical path passes through a single pattern (where the critical path for the pattern graph is the shortest path connecting all the room’s doors). To find dead ends, we first calculate the pattern graph’s critical path. Then, for every pattern in the critical path, we execute a breadth first search of adjacent patterns not in the critical path. For each such pattern, if the edge traversed to reach it is not in a cycle, then all other connected unvisited patterns not in the critical path make up a dead end.

Dahlskog et al. also note that an area might be considered less of a dead end if it contains interesting gameplay [4]. We encapsulate this idea in the quality measure for dead ends by rewarding dead ends that contain other meso-patterns (such as a dead end that contains a treasure chamber or a guard chamber). However, it may in some cases be desirable to create dead ends that do not contain anything interesting, in order to create a maze-like room layout. For this reason we allow the user to specify the proportion of patterns in a dead end that should also be contained in other meso-patterns and use this to define the function $Density(p)$ for a dead end $p$ as:

$$Density(p) = 1 - \frac{1}{\max\{T_{deadend\_density}, 1 - T_{deadend\_density}\}} \left| \frac{N_{mp}}{N_{sp}} - T_{deadend\_density} \right|, \tag{16}$$

where $N_{mp}$ is the number of meso-patterns present in the dead end and $N_{sp}$ is the
Figure 12: (a) shows a room generated with no penalty for dead ends. The pattern graph for this room, shown in (b), indicates that most of the room’s patterns are included in dead ends (shown by the black dot in the centre of these patterns). There is only one path between the three doors and large parts of the room are optional. Since these optional parts do not contain other meso-patterns, they may be undesirable (though it should be noted that our collection of meso-patterns is far from exhaustive). On the other hand, a designer may wish to have numerous dead ends, only some of which contain interesting gameplay.

Figure 13: (a) shows a room generated with a high penalty for dead ends. The pattern graph for this room, shown in (b), indicates that the room contains no dead ends (which would otherwise be marked by a black dot in the position of a pattern that is part of a dead end) and we can see that there are multiple possible paths between the three doors, allowing players to bypass the enemies if they choose.
Figure 14: a) shows a room containing the guarded treasure meso-pattern: a treasure chamber in a dead end such that the player must pass through a guard chamber to access the treasure chamber. b) shows the pattern overlay in EDD for this room, where the guard chamber in the centre and the treasure chamber in the upper right (here marked with the guarded treasure icon) make up a guarded treasure pattern. The black dots in the graph indicate that these patterns are part of a dead-end.

4.6.5 Guarded Treasure

In addition to the previous meso-patterns based on Dahlskog et al.’s definitions, we define one original meso-pattern, guarded treasure, with the goal of investigating a meso-pattern constructed purely from other meso-patterns, rather than micro-patterns or a combination of the two. Patterns like this are a step closer to Dahlskog et al.’s more abstract macro-patterns, which describe extended sections of gameplay dependent on multiple lower-level patterns [4]. A guarded treasure is a treasure chamber situated in a dead end, such that all paths from the treasure chamber out of the dead end pass through at least one guard chamber (see fig. 14). The pattern encapsulates the idea of risk vs. reward: the player
does not have to enter the dead end in order to access the room’s doors, but can choose to take a risk fighting the enemies in the guard chamber in order to collect the treasure.

We detect a guarded treasure by locating treasure chambers inside previously detected dead ends. From each such treasure chamber, we execute a breadth-first search on the pattern graph, searching for a node that is not part of the dead end. If all such paths pass through a guard chamber, the treasure chamber is considered a guarded treasure.

Quality for guarded treasure is defined by how close the number of enemies that must be passed in order to access the treasure is to a user-defined target number, \( T_{\text{deadenemies}} \), and is calculated (for a guarded treasure \( p \)) as:

\[
Q(p) = \max \left\{ 0, 1 - \left| \frac{N_E - T_{\text{deadenemies}}}{T_{\text{deadenemies}}} \right| \right\}.
\] (18)

### 4.7 Evaluating the Fitness of Feasible Individuals

The genetic algorithm optimises feasible individuals (rooms) to increase their fitness, which is a value between zero and one. This fitness acts as a measure of how well the room satisfies the user’s requirements on the types and quality of patterns present in the room. While Liapis et al.’s Sentient Sketchbook generates maps based on six different fitness dimensions (in addition to novelty), these dimensions are only optimised one at a time, in six different evolutionary runs, so a map cannot be optimised based on two or more of these dimensions simultaneously [23]. By treating all of the elements of our rooms as design patterns, we are able to treat all of these in a standard manner and optimise multiple patterns simultaneously.

The fitness is a linear combination of two functions \( f_{\text{inventorial}}(r) \) and \( f_{\text{spacial}}(r) \), which evaluate a room \( r \)’s inventorial and spacial patterns respectively. The reason for this separation is that the fitness of spacial patterns is weighted according to how much of the available area of a room those patterns take up (so a two tile long corridor has less impact on the overall fitness of the room than an 8x8 chamber), while the fitness of inventorial patterns is not.

Since meso-patterns coincide with spacial micro-patterns, they are not evaluated in isolation, but instead their qualities are used to modify the quality of the spacial micro-patterns they contain (described in section 4.7.2). After this modification, the quality of a spacial pattern is a linear combination of its own quality and the sum of the qualities of any meso patterns it is in (capped at 1). So, for example, a chamber that is not also part of a meso-pattern has its quality capped at 0.9, while if that chamber is part of the treasure-chamber meso-pattern, it may reach a quality of 1.
4.7.1 Inventorial Pattern Fitness

The inventorial pattern fitness $f_{\text{inventorial}}(r)$ for a room $r$ is calculated as a linear combination of the sums of the qualities of each pattern type:

$$f_{\text{inventorial}}(r) = \frac{1}{5} \sum_{p \in D} Q(p) + \frac{2}{5} \sum_{p \in E} Q(p) + \frac{2}{5} \sum_{p \in T} Q(p), \quad (19)$$

where $D$, $E$ and $T$ are the sets of all doors, enemies and treasures, respectively.

4.7.2 Spacial Pattern Fitness:

Spacial Pattern fitness $f_{\text{spacial}}(r)$ is a measure of the frequency and quality of design patterns detected in a room and is a linear combination of two other functions, $f_{\text{chamber}}(r)$, which evaluates the fitness of detected chambers and $f_{\text{corridor}}(r)$, which evaluates the fitness of detected corridors and connectors. This fitness can be seen as a measure of how closely a room adheres to user-defined target values for the proportion and quality-parameters of each design pattern. The final quality of a spacial micro-pattern $p$ is a linear combination of its own quality and the quality of the meso-patterns it appears in, $Q_{\text{meso}}(p)$, which is defined as

$$Q_{\text{meso}}(p) = \min \left\{ 1, \sum_{m : p \in m} Q(m) \right\}. \quad (20)$$

$WeightedChambers(r)$ then calculates the proportion of passable tiles in a room $r$ that are in chambers, weighted by the quality of those chambers (see eq. (4)) and the meso patterns that include the chambers as

$$WeightedChambers(r) = \sum_{p_{\text{chamber}} \in r} \left( \frac{9}{10} Q(p_{\text{chamber}}) + \frac{1}{10} Q_{\text{meso}}(p_{\text{chamber}}) \right) \frac{\text{Area}(p_{\text{chamber}})}{N_P}, \quad (21)$$

where $p_{\text{chamber}}$ is a detected chamber pattern and $N_P$ is the number of passable tiles in the room. This is used to calculate the chamber fitness as the degree to which the proportion of chambers differs from a user-defined target proportion $T_{\text{chamber}}$ as

$$f_{\text{chamber}}(r) = 1 - \frac{\max \{ WeightedChambers(r) - T_{\text{chamber}}, 1 - T_{\text{chamber}} \}}{\max \{ T_{\text{chamber}}, 1 - T_{\text{chamber}} \}}. \quad (22)$$
WeightedCorridors(r) calculates the proportion of passable tiles in a room that are parts of corridors or connectors, weighted by the quality of those corridors/connectors (see eq. (1)) as

\[
\text{WeightedCorridors}(r) = \sum_{p_{\text{corridor}} \in r} \left( \frac{9}{10} Q(p_{\text{corridor}}) + \frac{1}{10} Q_{\text{meso}}(p_{\text{corridor}}) \right) \frac{\text{Area}(p_{\text{corridor}})}{N_P} + \sum_{p_{\text{connector}} \in r} \left( \frac{9}{10} Q(p_{\text{connector}}) + \frac{1}{10} Q_{\text{meso}}(p_{\text{connector}}) \right) \frac{\text{Area}(p_{\text{connector}})}{N_P},
\]

(23)

where \(p_{\text{corridor}}\) is a detected corridor pattern and \(p_{\text{connector}}\) is a detected connector pattern. This is used to calculate the corridor fitness as the degree to which the proportion of corridors (including connectors) differs from a user-defined target proportion \(T_{\text{corridor}}\) as

\[
f_{\text{corridor}}(r) = 1 - \left| \frac{\text{WeightedCorridors}(r) - T_{\text{corridor}}}{\max(T_{\text{corridor}}, 1 - T_{\text{corridor}})} \right|.
\]

(24)

Combining these two fitness functions (using weights determined through a process of experimentation), we arrive at the spacial pattern fitness

\[
f_{\text{spacial}}(r) = \frac{3}{10} f_{\text{chamber}}(r) + \frac{7}{10} f_{\text{corridor}}(r).
\]

(25)

### 4.7.3 Feasible fitness

Combining the inventorial pattern fitness with the spacial pattern fitness, we are able to calculate the overall fitness for a feasible individual as

\[
f_{\text{feasible}}(r) = \frac{1}{2} f_{\text{inventorial}}(r) + \frac{1}{2} f_{\text{spacial}}(r).
\]

(26)

### 4.8 Evaluating the Fitness of Infeasible Individuals

The infeasible fitness score evaluates individuals that do not satisfy the playability constraint introduced in 4.1. Rooms that are closer to fulfilling the playability constraint will be awarded higher fitnesses. The fitness \(f_{\text{infeasible}}(r)\) is calculated as

\[
f_{\text{infeasible}}(r) = 1 - \frac{1}{3} \left( \frac{N_{\text{path}}}{N_E} + \frac{N_{\text{path}}}{N_T} + \frac{N_{\text{path}}}{D} \right),
\]

(27)

where \(N_{\text{path}}, N_{\text{path}}, \) and \(N_{\text{path}}\) are the number of enemies, treasures, and doors respectively for which there is no path to the main entrance and the best fitness is 1. \(D\) is the number of doors excluding the main entrance.
4.9 User Interface and Workflow in EDD

As already seen in tools like Sentient Sketchbook [23] and Tanagra [34], the goal of a mixed-initiative generator is to take advantage of the combination of artificial intelligence techniques with the natural intuition and creativity of human designers. This section details the user interface through which designers can interact with and influence our generator.

A primary consideration in the construction of this kind of tool is what degree of granularity to provide in the level of control over the generation. We could have chosen to expose all of the algorithm’s weights and control-parameters for direct manipulation by the game designer, but since part of the motivation for this work is to reduce the need for laborious and potentially unintuitive tweaking of generation algorithms, we have opted to employ an almost opposite approach. We do so in order to investigate whether an adequate level of control over room generation can be provided without any direct access to the algorithm’s input parameters. These parameters are instead determined ahead of time in pre-made configuration files and then manipulated automatically at runtime as necessary. The extent to which this approach is successful is evaluated and discussed in section 5.

![Diagram of the basic workflow supported by EDD.](image)

**Figure 15:** Illustration of the basic workflow supported by EDD. Upon starting the application, room suggestions are generated. The user can choose either to generate more suggestions or choose one and begin editing it. Then, room editing consists of an iterative process of manual editing and generating new suggestions from the current room (mutation) and selecting them. When the user is happy with the created room, it can be saved.

We provide a workflow in which the user is presented with a number of widely varied suggestions of possible maps and can then follow an iterative process of regeneration and manual refinement to create a room they are happy with. An overview of this workflow can be seen in fig. 15. As highlighted by Liapis et al. in Sentient Sketchbook [23], one of the most powerful aspects of a mixed initiative generation tool is the potential for providing information about the properties of a generated room. As such, we also provide extensive visual indication of the type of game design patterns found in a generated or edited room and how these patterns relate to each other.
The tool is divided into two distinct interfaces: the room suggestion view and the room editing view, both of which are described in detail in the following sections.

4.9.1 Room Suggestion View

After opening EDD, six rooms are generated from parallel runs of the generator using different configuration files selected from a small and varied pool (see fig. 16), with the goal of acting as a starting point for the mixed-initiative generation process. If none of the generated rooms are interesting, six new ones can be generated. After clicking on one of the suggestions, it is loaded in the room editing view.

Configuration files contain values for all of the room’s target values for pattern generation as described in section 4.3 and the following sections. For example, a configuration might specify that a room should contain 40% corridors, 60% chambers, 2% treasure, no dead ends, etc.

![Figure 16: EDD’s room suggestion view. Six rooms are generated from a selection of predefined configuration files, offering a variety of different micro- and meso-patterns. In (a) maps are loading, which may take several seconds (or longer for large rooms). In (b) the maps have loaded and the user can either select one to edit or generate six new suggestions.](image)

4.9.2 Room Editing View

EDD’s room editing view is the main interface for generating new room suggestions, manually editing rooms and viewing the game design patterns contained in a generated or edited room. The basic functionality of this view is outlined in fig. 17. This view can be considered to serve two main roles: firstly, to enhance designer creativity by suggesting new rooms based on an existing room and, secondly, to provide feedback about the types of game design patterns present in a room.

Room suggestions are generated based on the current room in two ways. Two rooms are generated using a new configuration based on the proportion and type of design patterns
Figure 17: EDD’s room editing view, showing: (1) A larger, manually editable version of the selected room. (2) A toolbox of tiles that can be applied to the room. Note that only floor, walls, treasure, and enemies can be placed in this fashion – door positions are fixed and cannot be changed. (3) The Show patterns button overlays the room with a view of the patterns it contains – see fig. 18 for more details. (4) The Generate maps button will generate four rooms based on the room being edited. (5) Newly generated rooms appear here and can be selected to replace the currently chosen map in order to edit them. (6) A legend explaining the meaning of the icons shown in the pattern overlay.

detected in the current room (so, if the current room was generated using a configuration that aimed for 40% chambers, but the generated room actually contains 47% chambers, the new configuration will aim for 47% chambers). The remaining two suggestions are also generated using a configuration computed in the same way, but each run is seeded with individuals that are mutated versions of the current room (50% of tiles are mutated randomly to different tile types). By employing these two types of mutation we aim to produce two different types of suggestions: rooms containing a similar distribution of game design patterns to the original room for the former approach and rooms that visually resemble the original room and contain a similar distribution of game design patterns for the latter approach.

As an aid to the understanding of the gameplay afforded by generated rooms, users can view a pattern overlay, as shown in fig. 18. This overlay is automatically updated as the room is edited, and edits can be made with the overlay visible, providing rapid feedback on how changes to the layout of the room affect the resulting gameplay. A legend of pattern icons indicates the meaning of the graphics in the overlay.

Rooms are edited by changing existing tiles to one of the other tile types: floor, wall, treasure and enemy (see fig. 19). Door positions are locked and cannot be changed (except by starting a new generation process) or overwritten. In the case that the user creates an unplayable room (that is, the room does not satisfy the playability constraint defined
in section 4.1, the room is highlighted with a red border, but new rooms may still be generated based on this unplayable room.

Figure 18: EDD’s room editing view, with pattern overlay enabled. Spacial micro-patterns are displayed as blocks of colour, blue for chambers, red for corridors and yellow for connectors. Black lines show the pattern-graph for the room, where meso-pattern icons are overlaid, as described in section 4.6.

Figure 19: EDD’s room editing view, showing: (a) a room that has been extensively edited and (b) an infeasible room. Since manual editing is allowed, it is possible for users to create rooms that do not satisfy the playability constraint (described in section 4.1). A red border is added to the edited room to indicate that it is not feasible. In this case the room is infeasible because some floor tiles are not connected to the rest of the room.
5 User Study

In order to assess the relevance of a mixed-initiative PCG component in a level editor, a user study was conducted using EDD. The goal of the study was to answer whether the users would feel that they were given a sufficient level of control of the level generation process, and whether the users would deem mixed-initiative tools to be a potentially useful addition to a game designer’s toolbox.

5.1 Study Design

The user study consisted of five user tests, in which the users designed a set of dungeon levels using EDD. The users were then interviewed in order to gather their impressions and thoughts regarding the process of working together with an AI-based mixed-initiative tool in general and EDD in particular.

5.1.1 Users

We invited five professional game developers working at a major local game development studio to test EDD in person. The users were mainly recruited through personal contacts. As such, three of the users were known to the authors and colleagues prior to the study. The last two users were recruited by the original three users. A breakdown of the users looks as follows:

User 1 has been working as a first person shooter (FPS) level designer for two years. While he hasn’t designed dungeon style levels, he has played games containing them. He has no prior experience working with AI-assisted design tools and is unfamiliar with the game design patterns concept.

User 2 works as a senior animator. He has been involved in developing RPGs with dungeon style levels earlier in his career. Game design patterns were a new concept to him. While he hasn’t used AI-assisted design tools, he expresses appreciation for any tool that could potentially speed up the design process.

User 3 works as a senior tech and tools programmer, building tools to assist the game developers, including level design and development tools. He hasn’t worked with level design, nor has he played any games with dungeon style levels. He has used procedural generation tools before.

User 4 is a senior user researcher and analyst with a background as a level designer with two other game studios. He claims prior knowledge of games with dungeon style levels by having played Spelunky \[41\]. Prior to trying EDD, he has used Tanagra \[34\], which didn’t impress him, and Sentient Sketchbook \[23\], which he liked. He has prior knowledge of game design patterns, although his definition differed slightly from Björk and Holopainen’s.

User 5 is a trained game level designer. He has built automated tools for level design, specifically for games with dungeon style levels. While he remembers reading about
game design patterns, they aren’t something that he has worked with. Instead, he uses the concept of lenses \[22\] in his daily work. He doesn’t consider the tools he has used before to be AI-assisted, due to a lack of a clear definition on what constitutes AI.

5.1.2 Process

Each test session took between thirty minutes and an hour and was divided into the following phases:

**Pre questionnaire** The user filled in a short questionnaire about their game development responsibilities and previous relevant experience (see section \[A\]).

**Introduction** One of the authors gave a brief introduction to EDD, describing its function, intended audience, and superficial details of the underlying implementation.

**Demonstration** One of the authors gave a short demonstration of portions of the standard EDD workflow (see section \[4.9\]). The following steps were covered:

- Generating map suggestions, i.e. starting up a new session.
- Selecting a suggestion. This step included an introduction to the room editing view and a demonstration of the pattern overlay. Any pattern visible in the overlay was described during this step.
- Manually editing the map. The different tile types available were also described.
- Generating and selecting new map suggestions, based on the currently visible map.
- Saving the current map.

The demonstration included a brief introduction to the patterns, including the more complex meso patterns, recognised by the tool. This was typically done by means of manual editing of the maps by one of the authors.

**Room generation** The user was asked to design three rooms that would be part of the same dungeon level, suggesting that some kind of progression (e.g. in difficulty) between the rooms might be a desirable goal. Each room was modelled as a square of 11 by 11 tiles with three doors. The nature of the dungeon and the rooms was left to the user’s imagination. No further information was given on the layout of the individual rooms. Similarly, no information was given regarding the nature of the enemies and treasures in advance. Upon inquiry, the authors would tell the user that the enemies and treasures were to be considered generic place holders, giving the user the opportunity to make of them what they wanted. All the generated maps were saved for later discussion, which was conducted with the user after this phase, and analysis. One of the authors kept notes of what the user was doing, providing additional data for the analysis.
Interview  The authors conducted a short, semi-structured interview with the user, using the questions listed in section B as a starting point. During the interview, one of the authors took the lead in asking questions, while a colleague took written notes. The audio of the interview was recorded for later analysis.

5.1.3 Threats to Validity

While the study may provide some preliminary insights, it is important to keep in mind that the user study was relatively small with only five users, and out of these only two are currently working as game designers. However, out of the remaining three users, one has worked as a level designer in the past, and one works as a game development tools programmer. These should therefore be considered highly relevant for the user study as well.

It can be argued that having five users from the same company could be considered a threat to validity. However, seeing that the subjects all have different backgrounds and all took different approaches to the task at hand, we feel confident that the sample of users is good enough to fulfil the study’s objectives.

EDD provides a base for level editing, but still lacks some functionality, e.g. more a diverse tile set and a more direct way of influencing the parameters of the fitness function. A related issue is the relative lack of user friendliness in the tool. Those two factors might have lead to frustrations with the tool amongst the users.

Each user spent roughly fifteen minutes with the tool, which might be argued to be too short a time to get properly familiarised with the tool and its way of working.

It is possible that our interpretation of the interview and questionnaire data could differ from the participants' desired meaning. The participants were asked for clarification if something they said during the interviews was unclear. In a similar manner, the participants were asked for clarification if the questionnaire was unclear. However, from the perspective of internal validity, a better approach may have been to ask the participants to review our data after it had been collected and analysed.

5.2 Results

The data collected from the questionnaires, interviews, and observations was categorised according to a number of themes: usage of room suggestions, usage of the pattern overlay feature, and impressions of the tool and mixed-initiative workflow. Using these categories, we were able to analyse the data based on our areas of interest. This analysis is presented in the following sections.

5.2.1 Questionnaire

Two of our users currently work as game/level designers. One has worked as a level designer in the past and currently works as a senior user researcher and analyst. One works as an animator, but has some limited experience of designing dungeons in the past. One is a programmer involved in developing tools for level design.
Figure 20: (a) User 1’s third room and (b) the pattern overlay for the room. While the user created two chamber-like structures containing treasure protected by enemies, since they do not conform to our chamber or guarded treasure definitions, they are not identified as such in the pattern overlay (and will also not be recognised for the purpose of generating further rooms suggestions).

The term game design pattern was only familiar to one of the users (though others were able to speculate about its meaning or remembered hearing the term but not what it refers to). His definition “solutions to reoccurring problems” is however closer to the classic idea of design patterns in architecture/software engineering than the game design pattern definition suggested by Björk and HoloPainen [2]. While game design patterns certainly serve as a means of describing gameplay elements, this unfamiliarity with the term might suggest a need to more deeply investigate the principles and language used in professional level design situations in order to communicate these concepts in a language level designers understand.

Only one of the users had prior experience of using artificial intelligence-assisted level design tools like EDD, which may have had an impact on their understanding of our expectations for the level design workflow in EDD.

5.2.2 Room Generation Task & Interviews

We asked each user to create three rooms from a dungeon level, preferably with a degree of progression between the different rooms taken into account. It is interesting to note that the approach taken to the task varied considerably between the users. Whereas users 1 and 5, both professional level designers, spent a lot of time analysing the initially proposed maps compared to the other users, user 4 selected maps based on emptiness in order to start off each room design with a clean slate. To him, the proposed maps held little to no value. User 3 spent a lot of time evolving maps, albeit not always selecting any of the evolved maps. Out of the five users, user 3’s workflow most closely matched the kind
Figure 21: (a) User 2’s first room and (b) the pattern overlay for the room. All of this user’s rooms were dominated by corridors and connectors and, when asked, he explained that his intention was to create a claustrophobic and maze-like environment for first-person-shooter style game. This is reflected in the pattern overlay by the high number of corridors and connectors and the large dead end areas (and might also explain the unusually high number of enemies). Since the majority of our meso-patterns involve chambers, most of the patterns in this room are limited to the micro-level, pointing to a need for a broader range of meso-pattern definitions to suit games of varied styles and genres.

Figure 22: (a) User 3’s third room and (b) the pattern overlay for the room. As a non-designer, user three appeared to rely more on the generated suggestions and less on manual editing and, as such, produced maps with a high-density of identified design patterns.
Figure 23: (a) User 4’s third room and (b) the pattern overlay for the room. Designed for symmetry, the user expressed disappointment that room suggestions based on this room were not similarly symmetrical (which is as expected, since our method does not optimise to preserve aesthetic qualities of rooms).

Figure 24: (a) User 5’s third room and (b) the pattern overlay for the room. The user kept large parts of the suggested rooms unchanged, focusing more on editing the position and distribution of enemies and treasure and modifying the number of paths through the room. Structures he considered design patterns are not identified as such. For example, he described the enemy in the bottom right as a “jump scare”, since it is hiding behind a corner. As with user 1 (see fig. 20), user 5’s open areas are too small to be considered chambers by our generator.
of workflow EDD was originally intended to support, which suggests that changes to this workflow are necessary.

We noticed that the two level designers tended to use the initially selected maps as templates for further development. Neither of them spent much time generating new maps. While they both used the pattern overlay to some degree, user 1 expressed a lack of usefulness of pattern detection and display in rooms of this size. User 1 noted that what he felt would constitute a chamber failed to be recognised as such, due to our definition of a chamber containing a core of 3 by 3 tiles (see fig. 20). Compared to the other designers, they tended to design rooms with fewer dead ends and more alternative paths between the doors (see fig. 21). The alternative paths were described by user 5 as a means to present a potential player with different challenges, by e.g. utilising risk and reward [2]. The designers’ low usage of the map generation button might be an indication of them having concrete ideas for the rooms, thus making the use of further map suggestions redundant for them. Two of the users considered the generated rooms to be too random, making the evolutionary path from the parent room hard to see or understand.

Most of the users made little use of the pattern overlay. When it was used, it was mainly used as a “check” in between shorter edit iterations. In no case did the users manually edit the map with the pattern overlay active, even though one of the authors specifically demonstrated this functionality. Some of the users used the overlay to analyse the different paths in each room. User 4 expressed a desire for the different paths to be more aware of their situation in terms of e.g. difficulty and reward. User 3 spent more time with the pattern overlay than most other users, but seemed to be more interested in the technical aspects of the pattern detection and composition (see fig. 22) than in the patterns’ effect on gameplay. As previously mentioned, some of the users expressed a perceived lack of usefulness for pattern visualisation in the relatively small rooms they were expected to work with. Nonetheless, the same users expressed that the pattern display would be useful in larger rooms in order to easier grasp the layout and nature of those rooms.

With regards to the usefulness of pattern detection and visualisation, users 1, 4, and 5 noted some perceived problems. In some cases the tool didn’t agree with their notion of what should match a game design pattern instance, e.g. a chamber or a corridor. User 4 at one point actively tried to construct rooms containing patterns not previously described, which the tool failed to recognise. Two of the users expressed a wish to be able to define new patterns or redefine existing ones, e.g. the dimensions of a corridor or the definition of a guarded treasure, which most users interpreted as being a treasure with an adjacent enemy. All in all, four out of five users considered the pattern detecting capabilities important, with the dead end pattern being deemed particularly important.

Somewhat related to the pattern visualisation is the feasibility display mechanism, which marks infeasible rooms (see section 4.1) by adding a red border to the map. This feature wasn’t described during user 1’s demonstration phase. He stated during the interview that he failed to see a connection between the red border and the feasibility/infeasibility of the room. Once the connection was pointed out, he deemed it useful. The remaining users were made aware of the feature during the demonstration phase. While one of the users lamented the lack of more specific information regarding what made a room infeasible, the
overall notion was that the feature is an important one.

Several users noted that the generalised approach taken by EDD was problematic. They lacked information regarding the game type and game mechanics, e.g., the nature and behaviour of the enemies or the meaning of the treasures. They also lacked an overview of the entire level (of which each room is just a small part), which is important for understanding the room's role in the player's progression through the level. This suggests that a tool such as EDD could benefit from being part of an existing tool such as the Evolutionary World Designer [9].

While all users claimed to design their rooms with regards to gameplay to some degree, some also designed rooms with more obvious attention given to their aesthetic qualities. User 4 described aesthetics and gameplay as equally important and also expressed disappointment when the tool failed to recognise and preserve his highly symmetrical designs (see fig. 23). Since room symmetry isn’t encoded into our current patterns, there is no way for the tool to detect symmetry and optimise for it. In total, three users expressed a desire for the tool to preserve visual qualities of a room during evolutionary runs.

Some of the users suggested new features that could be added to the tool. User 2 used the tool to design rooms with 3D FPS games (see fig. 21) in mind and suggested that a 3D level export function would be useful. He also lacked the ability to edit several maps at once, using tabs or some similar device. User 5 requested an undo function and the ability to copy and paste parts of a room.

Four out of five users considered the tool interesting. User 2 noted that the tool could be useful for prototyping new levels. User 1 found the possibility of using generated rooms as a starting point for large scale rooms appealing, but stressed that he wouldn’t trust a computer to generate levels without human input. The fact that EDD lets the designer edit the generated maps makes it more useful in his opinion. User 3 agreed with this sentiment, and stated that manual editing capabilities are a must for a design tool. The group of users in total couldn’t agree on whether the MI aspect of EDD would save design time. However, user 5, who is a designer, argued that time could be saved by using edited room designs as seeds for new rooms. In general, the users agreed that the tool did help to enhance creativity. In particular, user 1 noted that while designing vast dungeons, mental fatigue will reduce the diversity in room designs. Having a tool that provides novel ideas could by his reasoning help mitigate problems with decreased creativity stemming from mental fatigue.

5.3 Conclusions

An important takeaway from the user study is that the usefulness of a mixed-initiative tool needs to be properly examined and assessed. More research needs to be done to fully understand the typical level designer’s workflow and needs.

It is interesting to note that the pattern overlay was used to get an overview of the rooms, hinting that game design patterns visualised in form of a graph, pattern icons, and tile tinting can be used to convey a room’s various qualities in a comprehensible and reasonably efficient way. This seems to especially hold true for larger rooms. Still, it is important to note that this might not be true if more patterns are added to the tool,
cluttering the maps with more symbols.

While game design patterns have been demonstrated to have a perceived value to the users, the patterns currently detected by EDD were deemed too limited, inflexible, and incomplete. For a tool of this kind to be more useful, those issues need to be addressed. The pattern concept may have to be expanded to include aesthetics-related patterns, in order to accommodate the desire of some users to retain visual aspects of manually edited rooms.

Another consideration is how useful it is for designers to be able to iteratively improve room suggestions by a process of editing and generating new suggestions. In this user study, the room generation function was not heavily used. While this might imply that room generation after the initial generation run is of limited value, it is important to consider that the small sample size of the users, as well as a lack of familiarity with this kind of tool, may skew the results. However, it does imply that our current way of generating or presenting new room suggestions lacks appeal to at least our user group.

The study suggests that a mixed-initiative PCG tool can be a valuable asset to level designers. It is, however, important to note that EDD was not used in a way that the authors had anticipated. This suggests that while the concept of a mixed-initiative tool is sound, a more useful instantiation will probably work differently from the instantiation presented here. Several users expressed a desire to see how the individual rooms fit into the larger picture, i.e. within a dungeon level. This points towards the need to include the fine control over individual rooms that EDD offers in a higher-level dungeon generation tool that creates entire level layouts, such as Evolutionary World Designer.
6 Experiments

To determine the usefulness of the generator, we must be able to show that it is possible to control the pattern-related properties of the generated rooms by adjusting the pattern-related input parameters, for a wide variety of different values. As such, we conducted a series of experiments to a) show that the genetic algorithm optimises the population of rooms to a fitness close to the maximum fitness – 1.0 – and b) measure the amount of each type of pattern detected in the optimal solution for a range of inputs. These experiments were conducted in two iterations: the first considering only micro-patterns in the generator’s fitness function and the second considering both micro- and meso-patterns in order to study whether the generator is able to optimise for the presence of meso-patterns in generated rooms.

6.1 Micro-Pattern Generation

In this set of experiments we study the extent to which we can control the proportion of spacial micro-patterns (chamber, corridor and connector) in generated rooms. That is: does there exist a correlation between the value of the generator’s input parameters governing pattern proportions and the proportion of those patterns detected in generated rooms? Additionally, we examine whether the generator optimises the fitness of these design patterns in order to create pattern instances that conform to the input quality targets. Finally, we show that the generator is able to optimise the overall room fitness.

6.1.1 Experimental Setup

Each experiment used a total population of 150 individuals, split evenly into feasible and infeasible populations. Rooms were 12x12 tiles (comparable to, for example, The Binding of Isaac [27], where rooms are 7x13 tiles). Using this configuration, one run takes approximately five seconds to complete, which indicates that the algorithm is suitable for real-time use. For each set of input parameters, 100 runs were executed, terminating after a fixed 150 generations and the following data were recorded for the feasible population at each generation:

- Maximum feasible fitness, \( \max(f_{\text{feasible}}) \)
- Average feasible fitness, \( \bar{f}_{\text{feasible}} \)
- Chamber fitness, \( f_{\text{chamber}} \)
- Corridor fitness, \( f_{\text{corridor}} \)
- Ratio of chamber tiles to passable tiles (not weighted by quality)
- Ratio of corridor (and connector) tiles to passable tiles (not weighted by quality)
- Enemy and treasure fitness, \( f_{\text{enemytreasure}} \)

For each entire run, the following were also recorded:
• The optimal room’s chromosome
• A graphical rendering of the optimal room in .png format

In total, 28 combinations of the following input parameters were tested (see section 4 for a discussion of their significance), chosen to represent a broad enough range of configurations to give us confidence in the internal validity of our results:

• Target chamber ratio, $T_{chamber}$
• Target corridor ratio, $T_{corridor}$
• Target chamber area, $T_{chamberarea}$
• Chamber squareness weight, $w_{squareness}$
• Chamber size weight, $w_{size}$
• Target corridor length $T_{corridor length}$
• Turn quality, $Q_T$
• Intersection Quality, $Q_I$

### 6.1.2 Limitations & Threats to Validity

At this stage of the evaluation of our generator, we are primarily interested in the ability to detect spacial design patterns, so the focus of the experiments and the presentation of our results will be on those patterns: chambers, corridors and connectors. In the version of the tool studied in this set of experiments, inventorial patterns were yet to be defined, so they are not addressed here. In the second set of experiments, presented in section 6.2, we do not deem it meaningful to study inventorial patterns in isolation, since their occurrence in rooms is closely linked to the occurrence of the meso-patterns that contain them.

Additionally, we are only interested in the variables that make up the genetic algorithm’s fitness function, not other parameters such as those controlling mutation, crossover and population size. All these parameters will remain constant, with values chosen for their ability to generate high quality solutions in preliminary testing.

When considering the diversity of generated rooms, we currently rely on visual inspection to judge the ability of an input configuration to create diverse rooms. Since we are not currently primarily examining the diversity, this kind of subjective evaluation may be sufficient, but in future work it may be wise to adopt quantitative measures of diversity, such as the distance metric discussed by Liapis et al. [23], in addition to considering the expressive range of the generator.
6.1.3 Optimisation of Feasible Fitness

Figure 25 shows the progression of the maximum and average fitnesses for the parameters \( T_{\text{chamber}} = 0.5, T_{\text{corridor}} = 0.5, T_{\text{chamberarea}} = 25, w_{\text{squareness}} = 0.5, w_{\text{size}} = 0.5, T_{\text{corridorlength}} = 4, Q_T = 0.5, Q_I = 0.5 \). While the maximum fitnesses at generations 1 and 150 vary between sets of input parameters, the remaining 27 configurations show the same trend of increasing maximum fitness and the average fitness approaching the same value as the maximum fitness. We observe that, in general, the algorithm achieves higher fitness when targeting a high ratio of chamber tiles and lower when targeting corridor tiles, due to a higher weight being assigned to the corridor fitness in the feasible fitness function.

6.1.4 Optimisation of Micro-Pattern Fitness

Here, we present the results of a selection of parameters, primarily focussing on configurations with different corridor and chamber ratios in order to discuss how well the input parameters correspond to the detected patterns. In each case corridor fitness, chamber fitness, the ratio of corridor tiles to all passable tiles and the ratio of chamber tiles to all passable tiles are graphed for each generation, averaged over 100 runs. Each graph is presented alongside an example of the optimal solution of one of those 100 runs.

Figure 26 shows the results for the configuration: \( T_{\text{chamber}} = 0, T_{\text{corridor}} = 1, T_{\text{chamberarea}} = 25, w_{\text{squareness}} = 0.5, w_{\text{size}} = 0.5, T_{\text{corridorlength}} = 4, Q_T = 0.05, Q_I = 0.95 \). In this case, the generator is able to produce solutions with the desired ratio of chamber tiles.
Figure 26: (a) shows an optimal solution for the configuration: $T_{\text{chamber}} = 0$, $T_{\text{corridor}} = 1$, $T_{\text{chamberarea}} = 25$, $w_{\text{squareness}} = 0.5$, $w_{\text{size}} = 0.5$, $T_{\text{corridorlength}} = 4$, $Q_T = 0.05$, $Q_I = 0.95$. 

(b) shows the progression of average corridor and chamber fitnesses and the average ratio of corridor/chamber tiles to passable tiles for all runs in this configuration.

Figure 26: (a) shows an optimal solution for the configuration: $T_{\text{chamber}} = 0$, $T_{\text{corridor}} = 1$, $T_{\text{chamberarea}} = 25$, $w_{\text{squareness}} = 0.5$, $w_{\text{size}} = 0.5$, $T_{\text{corridorlength}} = 4$, $Q_T = 0.05$, $Q_I = 0.95$. 

(b) shows the progression of average corridor and chamber fitnesses and the average ratio of corridor/chamber tiles to passable tiles for all runs in this configuration.

Figure 27 shows the results for the configuration: $T_{\text{chamber}} = 0.2$, $T_{\text{corridor}} = 0.8$, $T_{\text{chamberarea}} = 25$, $w_{\text{squareness}} = 0.5$, $w_{\text{size}} = 0.5$, $T_{\text{corridorlength}} = 4$, $Q_T = 0.5$, $Q_I = 0.5$. 

In this case, the corridor ratio in the optimal solution matches almost exactly the input parameter. The chamber ratio, on the other hand falls below the target of 0.2 to end just under 0.1, indicating that corridors have a tendency to dominate in the generated rooms. This can be explained by the fact that corridors must be flanked by wall tiles, which reduces the available floor tiles for chambers.

Figure 28 shows the results for the configuration: $T_{\text{chamber}} = 0.5$, $T_{\text{corridor}} = 0.5$, $T_{\text{chamberarea}} = 25$, $w_{\text{squareness}} = 0.5$, $w_{\text{size}} = 0.5$, $T_{\text{corridorlength}} = 4$, $Q_T = 0.5$, $Q_I = 0.5$. 

In this case the final ratios of both corridors and chambers are close to the target ratios, with corridors slightly closer at 0.55 compared to chambers at 0.41. Again, corridors are slightly overrepresented, while rooms are underrepresented. The figure also shows that the chamber fitness decreases even as the chamber ratio draws closer to the target value, indicating that the increase in corridor tiles causes a decrease in the quality of the existing chambers. The rate of increase of corridor fitness is, on the other hand, significantly higher than the rate of decrease of the chamber fitness, leading to an overall high fitness for the room.
Figure 27: (a) shows an optimal solution for the configuration: $T_{\text{chamber}} = 0.2$, $T_{\text{corridor}} = 0.8$, $T_{\text{chamberarea}} = 25$, $w_{\text{squareness}} = 0.5$, $w_{\text{size}} = 0.5$, $T_{\text{corridorlength}} = 4$, $Q_T = 0.5$, $Q_I = 0.5$. (b) shows the progression of average corridor and chamber fitnesses and the average ratio of corridor/chamber tiles to passable tiles for all runs in this configuration.

Figure 28: (a) shows an optimal solution for the configuration: $T_{\text{chamber}} = 0.5$, $T_{\text{corridor}} = 0.5$, $T_{\text{chamberarea}} = 25$, $w_{\text{squareness}} = 0.5$, $w_{\text{size}} = 0.5$, $T_{\text{corridorlength}} = 4$, $Q_T = 0.5$, $Q_I = 0.5$. (b) shows the progression of average corridor and chamber fitnesses and the average ratio of corridor/chamber tiles to passable tiles for all runs in this configuration.
Figure 29: (a) shows an optimal solution for the configuration: $T_{chamber} = 0.8$, $T_{corridor} = 0.2$, $T_{chamberarea} = 25$, $w_{squareness} = 0.5$, $w_{size} = 0.5$, $T_{corridorlength} = 4$, $Q_T = 0.5$, $Q_I = 0.5$. (b) shows the progression of average corridor and chamber fitnesses and the average ratio of corridor/chamber tiles to passable tiles for all runs in this configuration.

Figure 30: (a) shows an optimal solution for the configuration: $T_{chamber} = 1$, $T_{corridor} = 0$, $T_{chamberarea} = 25$, $w_{squareness} = 0.5$, $w_{size} = 0.5$, $T_{corridorlength} = 4$, $Q_T = 0.5$, $Q_I = 0.5$. (b) shows the progression of average corridor and chamber fitnesses and the average ratio of corridor/chamber tiles to passable tiles for all runs in this configuration.
Figure 29 shows the results for the configuration: $T_{\text{chamber}} = 0.8$, $T_{\text{corridor}} = 0.2$, $T_{\text{chamberarea}} = 25$, $w_{\text{squareness}} = 0.5$, $w_{\text{size}} = 0.5$, $T_{\text{corridorlength}} = 4$, $Q_T = 0.5$, $Q_I = 0.5$. Results are similar to those in fig. 28, with corridors being slightly overrepresented at 0.25, while chambers are slightly under-represented at 0.72. Corridor fitness is again significantly higher than room fitness.

Figure 30 shows the results for the configuration: $T_{\text{chamber}} = 1$, $T_{\text{corridor}} = 0$, $T_{\text{chamberarea}} = 25$, $w_{\text{squareness}} = 0.5$, $w_{\text{size}} = 0.5$, $T_{\text{corridorlength}} = 4$, $Q_T = 0.5$, $Q_I = 0.5$. In this case, both chambers and corridors are able to arrive at high fitness values: 0.88 and 1.0 respectively and almost all of the available passable tiles are chamber tiles, while none are corridor tiles. Without having to compete with corridors, it is clearly easier to optimise the room fitness.

6.1.5 Solution Diversity

While a high fitness and a ratio of pattern tiles closely matching that specified in the input parameters are important, the generator should also produce diverse solutions within these constraints. If the same inputs always result in the same output, the generator becomes useless as a tool for providing novel and diverse room suggestions. Figure 31 shows eight rooms generated using the same set of inputs, demonstrating that the generator is capable of producing visually and topologically varied outputs for a given set of inputs.

![Figure 31: Eight solutions resulting from the same input parameters: $T_{\text{chamber}} = 0.5$, $T_{\text{corridor}} = 0.5$, $T_{\text{chamberarea}} = 25$, $w_{\text{squareness}} = 0.8$, $w_{\text{size}} = 0.2$, $T_{\text{corridorlength}} = 4$, $Q_T = 0.5$, $Q_I = 0.5$, showing diverse layouts for a similar distribution of design patterns.](image-url)

In addition to producing diverse solutions for a set of inputs, the ability of the generator to control the extent to which patterns appear in rooms indicates that it is possible to generate solutions with diverse make-ups of patterns, by varying the generator’s inputs.
Figure 32: Some configurations result in a high number of solutions with no impassable tiles, as shown in (a). This is not necessarily bad as long as the user can understand why an empty room has been generated based on their inputs. The 180° rotation mutation operator makes partially symmetrical solutions, like (b), likely to be generated.

6.1.6 Anomalous and Noteworthy Results

One set of input parameters – \( T_{\text{chamber}} = 1, T_{\text{corridor}} = 0, T_{\text{chamber area}} = 9, w_{\text{squareness}} = 0.8, w_{\text{size}} = 0.2, T_{\text{corridor length}} = 4, Q_T = 0.5, Q_I = 0.5 \) – produced 34 rooms containing no wall tiles out of 100 runs. See fig. 32(a) for an example of one such room. Tests with similarly high squareness values, low target areas and a very low target corridor ratio produce similar results. This can be explained by the relative ease with which the generator can satisfy the squareness property compared to the target area property, since removing any wall tile will take the solution closer to the maximal square chamber: an entirely empty room. While not unexpected, these results could be confusing to a user of the generator, so such results should potentially be discarded from those presented to the user. There exist other “hard to generate” rooms configurations, such as simultaneously targeting a high ratio of corridor tiles and very large chambers. There is no obvious solution to this, since it should not be possible to simultaneously have, for example, 90% corridor tiles and chambers with an area of 100 in a 12x12 room. It may be desirable to prevent the user from selecting contradictory sets of input parameters. In the final version of EDD we avoid this by limiting direct control over input parameters.

Across most of the input configurations, we have observed a tendency for a significant number of the generated rooms to display partial 180° rotational symmetry for wall and floor tiles. See fig. 32(b), fig. 31(e), fig. 26(a) and fig. 3(a) for some examples. Since the likelihood of these symmetrical solutions arising by chance is slim, we can assume this is a result of the 180° rotation mutation operator applied in the genetic algorithm. While symmetrical solutions are often visually pleasing, it could be argued that a high chance of symmetrical maps reduces the potential variety of the maps that can be generated. The
use of the 180° rotation mutation operator could be a user-defined preference rather than a mandatory feature of the generator. In the case of square maps a 90° rotation mutation could also offer interesting symmetrical solutions.

6.1.7 Conclusions

While we have verified that the generator optimises the overall fitness of rooms, our results show that the generator is better at optimising either chambers or corridors than both at the same time, in particular, there are several cases in which chamber fitness actually decreases. In all cases targeting less than 100% of one pattern type, the ratio of chamber tiles to corridor tiles in the optimal solution is slightly lower than the value specified in the input parameters, whereas the value for corridors is either the same or higher than the input. However, despite the generator not producing rooms with the exact ratios of chamber and corridor tiles given as inputs, there is a clear correlation between these values, demonstrating that it is possible for the generator’s user to control the nature of the patterns found in the solution.

Given that the generator is intended to be part of a tool for aiding creativity in level design, there is no requirement that the level of control offered be particularly high, with a combination of control and diversity being more important. Visual inspection of the generated rooms indicates that the diversity of the generated solutions for a given set of input parameters is high, but a more thorough evaluation of diversity should be performed in the future.

Some of the cases showing the most deviation between the desired ratio of pattern tiles and the ratio in the optimal solution, such as that shown in fig. 27 result from the selection of combinations of parameters that are either impossible or difficult to satisfy entirely. For the aforementioned case, the ratio of chamber tiles is much closer to 0.2 when the target chamber area is changed from 25 to 9, which is a more realistic number of remaining passable tiles after 80% have been filled by corridors. A possible solution to this is to limit the combinations of input parameters to prevent inconsistent or incompatible parameters being chosen.

6.2 Meso-Pattern Generation

After the introduction of meso-pattern detection to our fitness function, we conducted a second set of experiments in order answer a number of questions regarding the extent to which the generator is able to optimise rooms to include these patterns:

1. Is it possible to control the proportion of dead ends in a room by modifying the generator’s input parameters?

2. Does the algorithm optimise rooms to increase the average number of treasure chamber, guard chamber, ambush and guarded treasure meso-patterns?

The reason for the distinction between dead ends and the remaining meso-patterns is that we provide direct control over how desirable dead ends are as a feature of rooms,
whereas the other meso patterns only allow control over the number of contained invendorial patterns (e.g. a treasure chamber’s quality is based on the number of treasures it contains). For the non dead-end meso-patterns, their occurrence should be largely governed by the target proportions of enemies and treasures in the room. For example, in a room where the target number of treasures in a treasure chamber is three and the target number of treasures in the whole room is six, we would expect the room to contain two treasure chambers. Essentially: given that there are a certain number of treasures in the room, we would prefer that these treasures are grouped together as part of a treasure chamber than spread across the room randomly.

6.2.1 Experimental Setup

Each experiment used a total population of 150 individuals, split evenly into feasible and infeasible populations. Rooms were 12x12 tiles. Using this configuration, after the introduction of meso-pattern detection to the fitness function, runs take significantly longer to complete, suggesting that fewer generations or smaller populations are more appropriate for real-time use in a mixed-initiative tool (and during the user study described in section 5 rooms were generated using significantly fewer generations and smaller population sizes). For each set of input parameters, 100 runs were executed, terminating after a fixed 150 generations and the following data were recorded for the feasible population at each generation:

- Maximum feasible fitness, $\max(f_{feasible})$
- Average feasible fitness, $\overline{f_{feasible}}$
- Chamber fitness, $f_{chamber}$
- Corridor fitness, $f_{corridor}$
- Ratio of chamber tiles to passable tiles (not weighted by quality)
- Ratio of corridor (and connector) tiles to passable tiles (not weighted by quality)
- Enemy and treasure fitness, $f_{enemytreasure}$
- Ratio of dead end tiles to passable tiles (not weighted by quality)
- Number of ambushes
- Number of treasure chambers
- Number of guard chambers
- Number of guarded treasures

For each entire run, the following were also recorded:

- The optimal room’s chromosome, in a format that can be opened in EDD for later analysis of contained patterns
A graphical rendering of the optimal room in .png format

31 combinations of the following input parameters were tested (see section 4 for a discussion of their significance), chosen to represent a broad enough range of configurations to give us confidence in the internal validity of our results:

- Target chamber ratio, $T_{\text{chamber}}$
- Target corridor ratio, $T_{\text{corridor}}$
- Target chamber area, $T_{\text{chamber area}}$
- Dead end penalty, $P_{de}$
- Target treasure range, $T_{\text{treasure}}$
- Target enemy range, $T_{\text{enemies}}$

Modified parameters were chosen based on their relevance for our chosen meso-patterns. The dead end penalty $P_{de}$ for example is intended to act as a means of controlling the proportion of dead ends present in a generated room (a value of 0 does not penalise dead ends at all, while a large value $> 10$ should heavily limit the number of dead ends found in the room). Target chamber ratio, corridor ratio and chamber area allow us to observe the generation of meso-patterns in rooms with different proportions of these spatial micro-patterns (we expect, for example, a room that contains no chambers not to contain any treasure chambers!). Target treasure and enemy ranges allow us to observe whether increasing the total number of treasure or enemies in a room results in a corresponding increase in the number of treasure or guard chambers generated.

Note that the target number of guards or treasures in a chamber for all of the chamber-based meso-patterns was set at three for all configurations.

6.2.2 Limitations & Threats to Validity

Note that we are not aiming to show that our generator is always able to generate all types of meso-patterns for every possible set of configurations. What we instead aim to show is that there exist sets of configurations for which we can generate meso-patterns and that, within those, it is possible to control the extent to which these patterns feature in generated rooms. A mixed initiative tool, such as EDD, can then be tuned to generate rooms using these configurations in order to focus on particular styles of gameplay (as defined by micro- and meso-patterns). Because of this, we cannot generalise our conclusions to other generator configurations (though we have selected a broad enough range of configurations that we are able to make reasonable assumptions about the types of meso-patterns that will be generated using similar configurations).
6.2.3 Meso-Pattern Optimisation

Figure 33 shows the results for a series of runs where dead ends were not penalised at all. It was our experience during prototyping that the default behaviour of the generator is to create maps with pattern graphs that do not contain cycles (and, as such, are likely to contain dead ends). The result here supports this: showing that when dead ends are not penalised in this configuration, an average of 38% of passable tiles are situated in dead ends. In practice, it is not possible to generate a map with 100% dead ends, since at least some of the tiles must be on the critical path, which, by definition, cannot be part of a dead end. It is not however clear what the actual upper limit on dead ends is, but it is likely significantly higher than the 38% average recorded here (another configuration – not shown here – optimising for 100% corridors resulted in an average of 58% dead ends, which is likely approaching this upper limit). Since we only penalise dead ends, rather than rewarding them, it is unlikely that we will be able to consistently achieve higher proportions of dead ends.

Figure 33: (a) shows an optimal solution for the configuration: \( P_{de} = 0, T_{chamber} = 0.5, T_{corridor} = 0.5, T_{chamberarea} = 9, T_{treasure} = [0.04, 0.05], T_{enemies} = [0.04, 0.05] \). (b) shows the pattern overlay for this solution, indicating that a large proportion of the room consists of dead ends. (c) shows the progression of the average ratio of dead end tiles to passable tiles for all runs in this configuration.

As illustrated in fig. 34 and fig. 35, we are able to manipulate the dead end penalty to generate rooms that are more or less likely to contain branching paths or “optional” areas. It should however be noted that this does not give fine control over the type of paths through the room: while a room with no (or very few) dead ends may contain multiple branching paths, it may also be made up of just one winding path connecting all the doors, since both are equally free of dead ends. In order to achieve finer control over the type of paths in the room, additional meso-, or possibly macro-patterns may need to be defined.

We can see in fig. 36 that the proportion of dead ends generated is inversely proportional to the value of the dead end penalty (for rooms with 50% chamber and 50% corridor – but a similar trend is shown for other configurations), demonstrating that it is possible
Figure 34: (a) shows an optimal solution for the configuration: $P_{de} = 3$, $T_{chamber} = 0.5$, $T_{corridor} = 0.5$, $T_{chamberarea} = 9$, $T_{treasure} = [0.04, 0.05]$, $T_{enemies} = [0.04, 0.05]$. (b) shows the pattern overlay for this solution, where, while there are a number of minor dead ends, the majority of the room consists of two branching paths. (c) shows the progression of the average ratio of dead end tiles to passable tiles for all runs in this configuration.

Figure 35: (a) shows an optimal solution for the configuration: $P_{de} = 10$, $T_{chamber} = 0.5$, $T_{corridor} = 0.5$, $T_{chamberarea} = 9$, $T_{treasure} = [0.04, 0.05]$, $T_{enemies} = [0.04, 0.05]$. (b) shows the pattern overlay for this solution, indicating that there are no dead ends and there are many alternative routes by which to traverse the room. (c) shows the progression of the average ratio of dead end tiles to passable tiles for all runs using this configuration.

to control the proportion of dead ends in a generated room. However, a weakness of this approach to controlling dead ends is that it is not immediately obvious how different values of the dead end penalty will affect the proportion of dead ends in the room (beyond the fact that larger penalties lead to the generation of fewer dead ends). We showed that we are able to reasonably accurately generate rooms with a given proportion of chambers and corridors by specifying the proportion of those patterns we want in the room. We have no such mapping from dead end penalty to the proportion of dead ends in the room, so it is not possible to specify, for example, that the desired result is a room with 45% dead ends. As the same dead end penalty may result in different proportions of dead ends in rooms.
with different proportions of spacial micro-patterns, it may not be possible to define such a mapping.

For the remaining meso-patterns: guard chamber, treasure chamber, ambush and guarded treasure, we simply count the number of instances of each pattern occurring in the optimal solution for each tested configuration, at each generation. In this case it is not meaningful to consider the proportion of the room that is made up of these patterns, since it will be largely dependent on details of the micro-patterns in the room. For example, in the case of ambush it is not possible for there to be more than one in a room, since the room only has one entrance. Likewise, the number of treasure chambers or guard chambers will depend on the total number of treasures and enemies in the room.

Figure 37 shows the meso-pattern counts for a configuration generating mainly chambers. In this case, the algorithm increases the number of both guard chambers and treasure chambers, with an average of about 1.42 of each in a room of this configuration. Guarded treasures are also generated by this configuration, with an average of 0.1 per room (since it is very unlikely that two guarded treasures will occur in a single room of this size, this essentially means that 10\% of maps generated with this configuration contained one guarded treasure). The average number of ambushes on the other hand actually decreases from its initial value at generation one, arriving at an average of 0.34 (or one in every three rooms). This is likely down to two main reasons: firstly, in terms of fitness, it is just as good for an enemy to be part of a guard chamber as an ambush, so the total number of enemies in the room must (unlike treasures), be divided up among guard chambers and ambushes. Secondly, an ambush can only occur if the entrance is actually in a chamber.
These four macro patterns have little impact on the layout of spacial micro-patterns, but rather control the distribution of inventorial micro-patterns, so if the entrance does not happen to be in a chamber, an ambush will not be present.

Figure 37: (a) shows an optimal solution for the configuration: \( P_{de} = 3, T_{chamber} = 0.5, T_{corridor} = 0.5, T_{chamberarea} = 9, T_{treasure} = [0.04, 0.05], T_{enemies} = [0.04, 0.05]. \) (b) shows the pattern overlay for this solution. Including the meso-patterns guard chamber and treasure chamber has encouraged the formation of chambers containing either treasure or enemies – rather than a combination of both. (c) shows the progression of the average number of each meso-pattern for all runs using this configuration.

When the target number of treasures is tripled, from \( T_{treasure} = [0.04, 0.05] \) to \( T_{treasure} = [0.12, 0.15] \), we see a corresponding three-fold increase in the number of detected treasure rooms in fig. 38 with an average count of 4.34. A similar increase in guard chambers occurs when increasing the target number of enemies. Note that increasing the number of treasures also leads to an increase in guarded treasures (from 0.1 to 0.25 in the cases of fig. 37 and fig. 38).

When generating rooms with lower proportions of chambers, it is expected that there will also be lower numbers of these chamber-centric meso-patterns. This expectation is demonstrated to be the case for the configuration shown in fig. 39 where, after an initial increase in frequency, the frequency of treasure chambers and guard chambers decreases before settling at a value of around 0.57. Ambushes are also less frequent, as it is less likely that the entrance will be part of a chamber. Guarded treasures were not found in any of the optimal solutions (in general this was the case for almost all tested configurations).

Tripling the target treasure range again results in a roughly three-fold increase in the number of treasure chambers, from 0.57 to 1.71, as shown in fig. 40.

### 6.2.4 Conclusions

By integrating meso-patterns into our fitness function, we are able to promote the generation of elements that represent higher-level gameplay concepts than those encapsulated by micro-patterns. While these elements may occur by random chance without a meso-pattern-based fitness function, the use of such a fitness function allows these patterns to
Figure 38: (a) shows an optimal solution for the configuration: $P_{de} = 3$, $T_{chamber} = 1$, $T_{chamberarea} = 9$, $T_{treasure} = [0.12, 0.15]$, $T_{enemies} = [0.04, 0.05]$. (b) shows the pattern overlay for this solution. Increasing the total number of treasures in the room has resulted in an increase in the average number of treasure chambers. (c) shows the progression of the average number of each meso-pattern for all runs using this configuration.

Figure 39: (a) shows an optimal solution for the configuration: $P_{de} = 3$, $T_{chamber} = 0.5$, $T_{corridor} = 0.5$, $T_{chamberarea} = 9$, $T_{treasure} = [0.04, 0.05]$, $T_{enemies} = [0.04, 0.05]$. (b) shows the pattern overlay for this solution. A lower proportion of chambers in the room has made chamber-based meso-patterns less likely to appear. (c) shows the progression of the average number of each meso-pattern for all runs using this configuration.

be specifically optimised for and increases the likelihood that micro-patterns in a room will contribute to a richer gameplay experience by being combined with other patterns.

We discussed the difficulty of generating rooms with conflicting configurations in section 6.1.6 and, in the same way that it is difficult to generate a room that simultaneously contains large chambers and a high proportion of corridors, it is difficult (or impossible) to generate treasure chambers in a room that contains no chambers or no treasure. Likewise, when aiming to generate a room with no dead ends, one of the easiest ways to do this is to generate a completely empty room. Knowing where to strike a balance between
Figure 40: (a) shows an optimal solution for the configuration: $P_{de} = 3$, $T_{chamber} = 0.5$, $T_{corridor} = 0.5$, $T_{chamberarea} = 9$, $T_{treasure} = [0.12, 0.15]$, $T_{enemies} = [0.04, 0.05]$. (b) shows the pattern overlay for this solution. Increasing the total number of treasures in the room has resulted in an increase in the average number of treasure chambers. (c) shows the progression of the average number of each meso-pattern for all runs using this configuration.

Micro- and meso-patterns in the fitness function, in order to achieve the characteristics of both, is a challenge – one that we have so far approached by a largely trial-and-error approach. Other research on multi-objective optimisation for game levels presents many opportunities for improvement here [31, 35].

Another challenge is to provide more control over the generation of meso-patterns. While we have shown that we are able to control the proportion of dead ends in a room, we are not able to influence the nature of these dead ends, so, for example, while a lack of dead ends often leads to a room layout with multiple branching paths, it cannot be relied upon to do so. Introducing new meso-patterns to govern the layout of paths in a room may be desirable, in order to provide more precise control (by, for example, defining a risk vs. reward meso-pattern that involves two paths, one of which is more challenging than another, but also contains more treasure).

On the other hand, it can be argued that it is not necessary to provide additional control over generation if our end goal is to provide designers with suggestions upon which they can iterate and build. In that case, it is more important that, by varying the generator’s input parameters, we are able to produce a wide array of rooms containing diverse micro- and meso-patterns.
7 Discussion and Conclusions

The goal of this research was to explore how game design patterns can be used in a procedural dungeon generator in order to generate levels based on the type of gameplay they contain, as well as to examine how this can be used in a mixed-initiative level generator as a means of facilitating collaboration between human designers and PCG algorithms. In particular, we asked three questions:

**RQ1** How can game design patterns be used to control procedural generation of dungeons?

**RQ2** How can game design patterns in dungeons be detected algorithmically?

**RQ3** How can the use of game design patterns in a mixed-initiative generator help designers realise their design goals?

Our answer to RQ1 comes in the form of the dungeon room generation method we presented in section 4. By using Dahlskog et al.’s approach of separating game design patterns into different abstraction levels based on their scale and complexity [4], we are able to use micro-patterns as a stepping stone towards generating meso-patterns (and, potentially, patterns on higher levels of abstraction). Meso-patterns represent gameplay scenarios that, while not necessarily conceptually complex, consist of multiple smaller constituent elements (such as a treasure chamber that consists of a chamber and multiple treasures). Because of this complexity and the size of the search space (all possible combinations of tiles in a dungeon room), a random process like a genetic algorithm is unlikely to be able to optimise a room to contain these patterns directly.

We showed in section 6 that we are able to control the generation of micro-patterns in dungeon rooms and that, by selecting appropriate micro-patterns, we are also able to generate meso-patterns based on these micro-patterns. Our results show that the values of the input parameters specifying target spacial micro-patterns (chamber and corridor) ratios are correlated with the ratios of these patterns found in the generated rooms, but that they do not currently offer precise control over the occurrence of these patterns. It is clear that the challenge of optimising a room to feature very different design patterns simultaneously is hard. In the case of chambers and corridors, the definitions are almost opposite: one is defined by open space and the other by enclosed space, so it is perhaps inevitable that one will tend to dominate the other – in our case corridors. It is likely that with further tweaking of weights in our fitness functions, a closer relationship between input and output ratios of design patterns is possible to achieve, but the generator may already be sufficiently accurate to be useful for game designers. As Togelius et al. note, “Optimising some aspect of a game for playability is inherently a multi-objective problem,” and this certainly applies to game design patterns (which can be considered to concern playability). As such, multi-objective evolutionary algorithms that search for Pareto fronts in order to optimise the fitness of one objective without reducing the fitness of other objectives are a potential source of improvement to our method, such as the SMS-MOEA algorithm that Togelius et al. use. However, it is unclear how effective existing multi-objective approaches are when there exist more than two or three objectives, which could be problematic when attempting to optimise many different patterns simultaneously.
Our proposed quality metric (see section 4) for patterns allows us to not only optimise rooms for the presence of specific patterns, but also to achieve finer control over the nature of those patterns, such as the length of corridors or the squareness of rooms. Consequently, the method has potential for supporting a wide array of design styles and dungeon themes, such as more regular ‘man-made’ style dungeons with square chambers and straight walls, or more organic cave-like structures with winding corridors and irregularly shaped chambers. It is important that these possibilities are afforded if the tool is to be useful for a designer, since the designer should feel they are able to express their own design style when creating levels.

An additional contribution we make as part of our method is the refinements in the pattern definitions we presented in section 4. While Dahlskog et al.’s dungeon design pattern catalogue [4] serves as a first step in providing the level of detail necessary to build a pattern-based dungeon generator, it rapidly became clear to us that nearly every pattern defined needed to be more precisely stated or redefined in order to use them in an instantiation. As such, we provide pattern definitions that are directly translatable to a real dungeon generation scenario. This is not to say that our pattern definitions are better than or should replace those given by Dahlskog et al. – on the contrary, their definitions acknowledge that even in the relatively narrow domain of dungeon levels, there exist many dramatically different approaches to level design (they identify five distinct dungeon types and within those there are certainly many different possible pattern definitions). Our pattern definitions provide one possible description of a small subset of the possible gameplay in connected-rooms style dungeons.

We also note that, if our method is to be applied in full, it is useful for meso-patterns to be defined in terms that can easily be translated to search-problems in an adjacency graph of micro-patterns. This should be taken into consideration when defining future meso-patterns. We are of the opinion that if there is to be truly robust game design pattern-based dungeon (or general level) generation, there needs to exist a clearer definition of the boundaries between micro-, meso- and macro-patterns, since there currently seems to be a considerable amount of overlap between these three abstraction levels.

The problem we consider in RQ2 is answered by the details we provide of how we detect micro- and meso-patterns in section 4. We describe a process in three stages: the first is the use of grid-search algorithms (breadth first search, flood fill) to directly analyse and categorise the entire room into spacial micro-patterns, the second is the construction of an adjacency graph from these micro-patterns and the third is the use of graph-search algorithms to identify meso-patterns in the graph. More generally, it is interesting to consider how the definitions of micro- and meso-patterns relate to our proposed method and, as such, how these patterns should ideally be defined in order to map well to the method. We assume that micro-patterns concern arrangements of space in dungeon levels - be it a single tile in the case of a treasure or a collection of connected tiles with specific geometric qualities in the case of a corridor. Then, meso-patterns are defined by how micro-patterns relate to each other in space, either in terms of adjacency in a pattern graph or in terms of the patterns they contain (such as a chamber that contains enemies).

To address RQ3, we need to consider how EDD’s workflow helps designers create dungeon rooms. One of the motivations for mixed-initiative PCG is the potential for
enhancing creativity by suggesting options a human designer wouldn’t have thought of or
eventing them to vary their style. One way of doing this is to provide suggestions, as
we do in EDD’s room suggestion view. By attempting to make these suggestions as close
as possible to something a human might actually design, through the use of game design
patterns, we can provide a starting point that is of a higher quality than a suggestion simply
generated at random or a blank canvas (though some users, including one of our user study
participants, might prefer a blank canvas). Beyond simply providing inspiration, this also
saves time, since the designer hopefully does not need to manually alter as much of the
level as they otherwise might.

What is less clear is how valuable it is for users to be able to iteratively work on their
levels with the help of the generator. For this to be useful, the designer must be able to
reasonably expect that rooms generated based on their design either preserve the design’s
essence (in the form of design-patterns present, appearance or both) or provide a better
alternative. For this to be possible, the generator needs to be able to recognise the user’s
conscious design decisions as game design patterns – something that was definitely not
always the case during our user study. Allowing designers to define their own patterns
– the things they consider important in the levels they design – could be an important
improvement in this regard. Another possibility is to automatically detect meaningful
patterns in users’ manually created designs by recognising regularly repeated features and
recording these as design patterns. Such an approach would likely require machine learning
techniques such as neural networks in order to be successful. Better recognition of what
users really consider to be design patterns would likely also enhance the utility of the
tool’s pattern visualisation functions, which were underused in our user study.

Another important consideration is how to balance aesthetics and gameplay when
co-designing levels with a mixed-initiative generator. It was clear to us when observing
our user study participants that some rooms were based around their desired gameplay
properties, while some focussed instead on aesthetics - such as symmetrical layouts and
repeated patterns or shapes. Since games are generally a visual medium, aesthetics will
always be an important concern. If a designer has selected a suggestion based on its
aesthetic qualities or designed a room to have specific aesthetics, it is reasonable for them
to assume that further suggestions will preserve those aesthetic qualities. This indicates
a significant gap in our method and also reveals several new challenges: firstly, how do we
know if a designer is more concerned with aesthetics or gameplay? Secondly, how can we
extend our method to simultaneously optimise rooms based on their aesthetics and the
game design patterns they contain?

Even within the same genre, games can be wildly different and even subtle differences
like the way an enemy behaves (what are its movement patterns? Does it engage in close-
combat, or does it attack from a distance, or both?) are very important to game designers
and play a major role in how they choose to design a level. As such, it might seem
necessary to either tailor a mixed-initiative level design tool very specifically to the game
it is to be used with (a potentially expensive task that may counteract some of the benefits
of using PCG) or to consider a collection of patterns broad and diverse enough to cover
every eventuality (likely leading to an unmanageably large multi-objective optimisation
problem even if it is possible to catalogue a large enough range of game design patterns). A
more practical option is to accept that a generator cannot reflect the subtlety and nuance in game design that a human designer understands and embrace the fact that mixed-initiative generation presents the possibility of combining the advantages of automated PCG in terms of speed and diversity with the intuition and experience of human designers.

While we have limited the scope of this research to the dungeon-generation domain (and more specifically connected-rooms style dungeons), we feel that the core of our method is applicable to level generation in general, given the existence of a comprehensive and well-defined set of game design patterns organised hierarchically. In particular, the use of micro-patterns as a stepping-stone for generating meso-patterns in the fitness function of a genetic algorithm as well as the use of quality-measures to achieve finer control over how patterns are expressed are general techniques that could be applied to the generation of other types of levels.

7.1 Future Work

As this work exists as an extension to the previous Evolutionary World Designer project \[9\], which generates higher level dungeon layouts (above the room level), a next step is to integrate this with EDD so that full dungeon levels can be evolved. Feedback from participants in our user study indicated that there is a need when designing detailed areas, like dungeon rooms, to be able to see these areas in their larger context, so this integration would likely improve the usability of EDD as a mixed-initiative design tool.

It is our view that there is a gap between the PCG research community’s understanding of the needs of the game development industry and its actual needs. Since design research relies upon its relevance for society, these needs should be systematically catalogued in order to serve as a robust foundation for future design research studies of PCG-assisted design tools. An appropriate approach might be to conduct a series of case studies at a range of game development companies (small and large and working on games of different genres and styles) to examine current design practices and tools as well as to identify gaps that could be filled or improvements that could be made by new tools and/or methods.

In this work our instantiation was in the context of a mixed-initiative design tool, but it would also be interesting to examine the potential of pattern-based methods in an online generator (i.e. content is generated while playing the game, without the direct intervention of game designers). Borrowing from Experience-Driven PCG \[40\], player or designer models could be automatically used to steer content generation in a way that suits a particular player or mimics the style of a particular designer, acting as a sort of automated version of the human designer in our mixed-initiative workflow. As suggested by Liapis et al. \[24\], designer modelling could also be used in a mixed-initiative tool in order to better tailor generated content to users (by, for example, emphasising aesthetics when being used by a designer who favours the aesthetic qualities of levels).

One aspect that has been the focus of previous research on mixed-initiative level generation \[23\], but was beyond the scope of this work is the importance of novelty when generating game content (where novelty refers to how diverse the population of candidate solutions is), particularly if the goal is to use generated content as suggestions for designers to build upon. Liapis et al. employ Lehman and Stanley’s Novelty Search \[19\] as well as
their own Feasible Infeasible Novelty Search, which combines the FI-2pop GA and Novelty Search paradigms in Sentient Sketchbook [21]. However, the challenge of optimising both novelty and quality (in our case based on game design patterns) has been less well explored. An approach like Lehman and Stanley’s Novelty Search and Local Competition, that treats optimising novelty (diversity) and quality as a multi-objective optimisation problem would likely serve as a good starting point for extending EDD with novelty-based objectives.

Beyond applying our method to domains outside of dungeon level generation, General Video Game Level Generation [17] – the task of generating levels for arbitrary games – is another potential application of a pattern-based approach to level generation. If games can be defined in terms of the game design patterns they contain, it could well be possible to use pattern-based approaches such as ours to generate levels for them.
References


Appendix A  User Study Questionnaire

The following are the questions participants in the user-study were asked to answer (in writing) before beginning the study.

1. What is your current position and role in the area of game development?

2. Briefly describe your experience of level design. In particular (but not limited to), the design of dungeon-style levels (in RPGs/roguelikes)?

3. Have you played/do you play games with dungeon-style levels (e.g. RPGs/roguelikes)?

4. Are you familiar with the term Game Design Patterns? If so, describe them briefly in your own words.

5. Have you used any Artificial Intelligence-assisted design tools? If so, which tools and (briefly) what was your opinion of them?
Appendix B  User Study Interview Guidelines

The following served as a set of guidelines questions/topics for the semi-structured interview conducted with user-study participants after they had used Evolutionary Dungeon Designer. Since the interviews were semi-structured in nature, questions asked in each interview do not necessarily correspond exactly to those listed below (some questions may not have been asked and additional questions may have been asked). The audio of each interview was recorded and written notes were made by an observer.

B.1 Introductory question

What is your general opinion of the tool? (First impressions)

B.2 Questions on patterns

- How useful do you think the patterns were?
- Did you find useful for the designing process to visualize the information about:
  - the chambers and corridors?
  - the connectors?
  - the dead ends?
  - the different kinds of chambers (ambush, treasure chambers, guarded chambers, and guarded treasures)?
  - the paths between the different patterns?
  - the infeasibility of the whole room?
- Do you think on these design patterns (or what they represent) during the process of designing a room?
- Are there any patterns missing in the tool?
- What else do you consider important during the process of designing a room?
- How important do you consider that aesthetics are while designing a room?
- How important do you consider that the gameplay is while designing a room?

B.3 Questions on the mixed-initiative tool

- Does the tool provide a sufficient level of control over the generated levels?
- In your opinion, what features are missing in the tool?
- How useful do you think that combining automatic suggestions with manual editing is?
- Do you think that this approach enhances the creativity of the level designer? Why?
• Do you think that you can save time while designing levels using a tool like this? Why?

• How useful do you think that the suggestions were while designing the rooms? Would you like to have more suggestions?

• How useful do you think that the manual editing was while designing the rooms? Would you like to have more editing options?

• How useful do you think that noticing the infeasibility of the room is?

B.4 Final question

Any final comments?