Evolving lock-and-key puzzles based on nonlinear player progression and level exploration

André Siqueira Ruela
Postgraduate Program in Information Systems
University of São Paulo
São Paulo, Brazil
andre.siqueira.ruela@gmail.com

Karina Valdivia Delgado
dept. of Information Systems
University of São Paulo
São Paulo, Brazil
kvd@usp.br

***Abstract*—This work proposes a novel approach for evolving puzzles of the lock-and-key style. The algorithm interprets a predefined graph structure as a dungeon level for which the puzzle is created and then an Evolutionary Algorithm is applied to fill the level with obstacles that challenge the player. A particular evaluation function in the Evolutionary Algorithm was conceived, allowing the emergence of diverse solutions that feature a nonlinear progression and a wide spatial exploration. The computational results showed that the algorithm is fast enough, that an online application is seen as a possibility.**

Keywords-Procedural content generation; automated game design; lock-and-key puzzle; evolutionary computation;

I. Introduction

In the last years, the field of Procedural Content Generation (PCG) has been growing fast, gathering attention from both industry and academia. In this scenario, one of the most traditional generation problems is the automatic production of levels and its inherent properties, entities, and artifacts [1]. The *level* is commonly defined as the spatial environment where the player character wanders through, during the gameplay. Although the level is considered a *content* of a game, each of its components constitutes also a class of content. Despite being a well-known problem, the task of producing a consistent level is still difficult and it is usually managed by search-based approaches, such as Evolutionary Algorithms (EA) [2].

Besides the construction of a good spatial level, it is necessary to fill it with obstacles that challenge the player to overcome them. The player, then, must be able to identify the obstacle, understand it, and search for a way to defeat it. The nature of these obstacles can vary considerably, depending on the game style. In general, these obstacles may not be crossed until the player reaches a symbol or fulfill a condition, such as an item or skill.

The purpose of this paper is to provide a tool for generating obstacles for game levels. In this level, the player must arrive at a *goal*, moving through the *rooms* and eventually overcoming the obstacles. In the scope of this paper, we define this obstacle as a lock-and-key puzzle [3]. Here, the lock and the key are metaphors for an obstacle and its solution, respectively, and not necessarily a literal door and a literal key.

According to [3], in a lock-and-key puzzle, the puzzle is finding out what is an obstacle, what and where is a key to overcome it, and finally using the key to master the challenge. This paper presents a novel approach, guiding the algorithm to maximize the use of the level in a nonlinear fashion. Our results show that the proposed method can develop distinct, resolvable, and nontrivial solutions with user-controlled nonlinearity and desired aesthetic features. The developed code is available at our GitHub repository[1] under the GNU General Public License v3.0.

II. Related Work

Lately, some papers have used graph grammars to generate levels for adventure games [4]–[6]. In these mentioned models, the graph structure describes first the puzzle (also called a *quest*, *mission* or a *plot* sometimes) that motivates the spacial level generation. Thus, driven by missions, the algorithm can generate meaningful levels. However, these works do not explicitly focus on the linearity property or the level utilization or exploration.

The level's nonlinearity is a very important feature that gives the player the impression he is following his own path or taking his own decisions to solve his challenges, and not merely following what the designer planned for him. According to [7], Shigeru Miyamoto, creator of the *Zelda* series, once said that he wanted to evoke the feelings associated with exploration in the player. Therefore, our proposed algorithm explicitly attempts to achieve such properties.

In this paper, the spatial level is given as an input for the puzzle generator, diverging from the related literature. This is done mainly because of our adopted goals, that are: achieve a nonlinear solution and a required broad exploration. To ensure such properties, it is important to know the level previously. In this way, the level is initially defined, for which a puzzle is generated. The level generator, however, is described in our previous work [8] and thorough details about its process are out of the scope of this paper.

[1]Source: https://github.com/KuruLab/PuzzleGenerator

III. Proposed Approach

We employ EAs, and more specifically, Genetic Algorithms (GA), due to its inherent ability to explore a diverse space of possible solutions and yet present high-quality results. In this section, we present our approach to procedurally generate lock-and-key puzzles through an EA.

A. Content Representation

To keep the algorithm's efficiency and ensure a fast computational performance, the puzzle content is represented by an integer array, where each element is a room ID and each position in such array have its respective meaning. The Table I shows the genotype-phenotype mapping scheme for the adopted representation.

Table I
Genotype-Phenotype Mapping

Index	Genotype	Phenotype
0	12	The room 12 is the *Start* room.
1	16	The room 16 is the *Boss* room (goal).
2	21	The room 21 contains the *A* key.
3	06	The room 06 contains the *B* key.
4	22	The room 22 contains the *C* key.
5	23	The room 23 contains the *D* key.

However, to decode the individual into a complete puzzle solution, more information is required. For instance, the location of a given key does not define where the doors will be placed. Thus, the decoding process not just distributes the keys along the level, but also computes the layout of the doors and suggest the level of difficulty for each room. This process is very similar to the *dungeon step* proposed in our previous work [8].

B. Evaluation

The fitness assessment takes into consideration a decoded and complete solution. In other words, the evaluation is computed over the input level with all entities, artifacts and conditions settled. The algorithm attempts to minimize the fitness value of the individuals. This value is defined by the fitness function, which is composed of three *design preferences* and a penalty function. Several papers from literature deal with unfeasible solutions by constrained optimization methods [5], [9], [10]. In this paper, we deal with this problem with a penalization method through the evolutionary process. Thus, unfeasible solutions coexist in the evolving population, but their fitness value receives a drastic penalty assessment, reducing its survival and reproduction chances. The evaluation function takes into consideration a route R computed by an A* algorithm that can solve the level by violating the game rules, breaking the doors.

Following, we present the fitness function, i.e. each design preference and the penalty function:

$$\phi = \mathrm{P} + \mathrm{DIN} + \frac{1}{\mathrm{VR}} + \frac{1}{\mathrm{TD}}, \tag{1}$$

$$\mathrm{P} = \sum_{i}^{R} \mathtt{max}\,(0, k_i - k_{A^*}) \times \beta, \tag{2}$$

$$\mathrm{DIN} = |\mathrm{NL}_{\mathrm{ideal}} - \mathrm{NL}_{\mathrm{actual}}|, \tag{3}$$

$$\mathrm{VR} = \sum_{i}^{N} v_i, \tag{4}$$

$$v_i = \begin{cases} 1 & \text{if } r_i \in R \\ 0 & \text{if } r_i \notin R, \end{cases} \tag{5}$$

$$\mathrm{TD} = \sum_{i}^{R} d_{ij}, \tag{6}$$

where P is the penalty, DIN is the *Distance from the Ideal Nonlinearity* (considering $\mathrm{NL}_{\mathrm{ideal}} = 3$), VR is the number of *Visited Rooms*, TD is the *Travel Distance*. The k_i is the key level required to access the room r_i, $r_i \in R$, k_{A^*} is the highest key level acquired by the A* algorithm when entering the room r_i, and β is a static penalty coefficient, defined as $\beta = 10^3$. Therefore, if $k_{A*} \geq k_i$, then $(k_i - k_{A*}) \leq 0$, resulting in no penalization. On the other hand, if $k_{A*} < k_i$, the individual fitness is penalized accordingly. Additionally, v_i marks if the room r_i is visited or not. The TD is the total travel distance of the entire route R, considering each room coordinates. Finally, where d_{ij} is the travel distance from room r_i to room r_j, such that $r_i, r_j \in R$ and $j = i + 1$.

Given the strength of the penalty coefficient, β, in some sense, the primary goal of the algorithm is to find feasible solutions. In this fitness landscape, the three design preferences can be seen as tiebreakers. However, they are the main contribution of this paper, in comparison with the related works, ensuring the good quality of the final solutions.

C. Genetic Operators

In this work, the algorithm has a population of $\mu = 100$ individuals and runs for a maximum of $g = 100$ generations. The individuals undergo selection employing a Binary Tournament. The crossover happens with a probability $\rho_c = 0.9$ and occurs with a random cut-off point, producing a pair of children, consisting of the permutation of their fathers' chromosomes. After the crossover, all children are submitted to the mutation operator, which has a global probability $\rho_m = 0.1$. There are two mutation operators, both with the same 50% probability: (i) swap the position of two randomly chosen rooms in the chromosome; (ii) replace a randomly chosen room by another randomly possible room. At the end of the reproduction steps, each child overtakes its parents' position in the population, without any kind of elitism. After elapsed a total of $g = 100$ generations, the algorithm stops and returns the best individual found so far.

IV. Computational Experiments

In this section, we present the computational results of the conducted experiments. To test the algorithm, there are two classes of levels, based on their sizes. It is challenging to define the ideal number of rooms in a dungeon. That relies on personal feeling and can vary drastically from game to game. While in [10] a procedurally generated dungeon presents 27 rooms, in [5] the authors takes as reference the Gnarled Root Dungeon, from *The Legend Of Zelda: Oracle of Seasons*, with only 20 rooms. In this work, we set the normal value at 25, but we also consider a larger dungeon of 50 rooms to verify the algorithm's capabilities.

The level generator consists of a stochastic search-based method itself [8]. For this reason, a set of 30 predefined levels is considered for each size. For each of these 30 levels, the puzzle generator is executed another 30 times. Consequently, there are a total of 900 puzzle solutions for each dungeon size.

The Tables II and III show the results of the computational experiments. The values are separated into minimal, maximal and mean with standard deviation. For each metric, the minimal and maximal values are evaluated independently. Note that for VR and TD, the best value is actually the "Max", while the best values of DIN, ϕ and the runtime are expressed by the column "Min".

Analyzing the results, it is possible to see that the algorithm is able to find the desired nonlinearity score in most of the times, however, failing in a few executions. Yet, considering the low mean values and their deviations, we can affirm that the results are very satisfactory and the algorithm is notably robust.

On average, the algorithm design puzzles that demand the player to visit 20 rooms to solve the level, for $N = 25$. In other words, the player must explore *at least* 80% (on average) of the dungeon space, but this exploration may be even more widespread during gameplay. For $N = 50$, this value is around 34 rooms, which corresponds to 68%. The best results of 24 (96%) for $N = 25$ and 43 (86%) for $N = 50$, are very exciting. And finally, the worst solutions seem to be disappointing, however, just like the DIN metric, based on the mean values, we can affirm that the algorithm achieves its goals.

Similar to the ideal number of rooms in a dungeon, an in-depth discussion of the traveled distances is complicated because it also depends on *ad-hoc* features of the game. This means that our interpretation of the results may vary considerably, depending on the game style. This is one of the reasons that TD is considered as the least important designer preference, or just a tiebreaker for the VR. This metric was adopted as a complementary way to promote

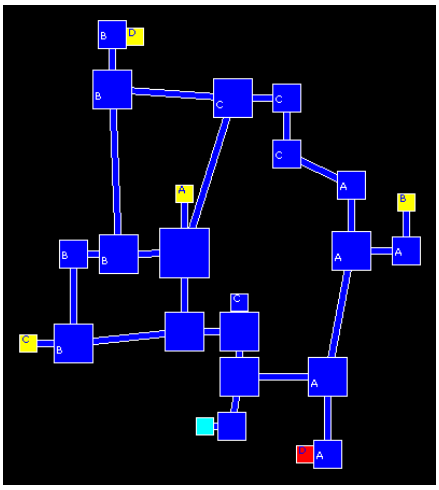

Figure 1. An individual from the $N = 25$ scenario.

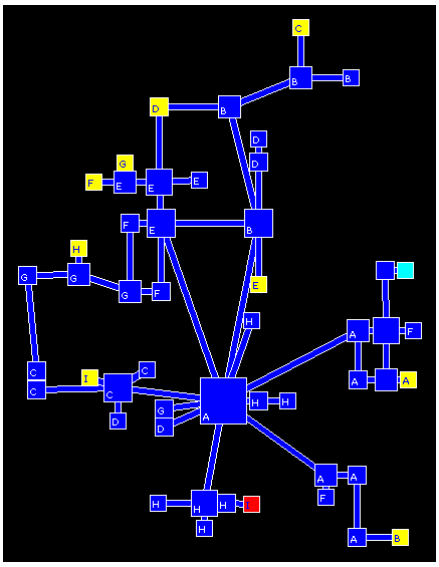

Figure 2. An individual from the $N = 50$ scenario.

Table II
Algorithm performance for $N = 25$

Metric	Min	Mean ± Std	Max
DIN	0	$0.003_{\pm 0.058}$	1
VR	16	$20.083_{\pm 1.823}$	24
TD	1034.086	$1953.182_{\pm 293.573}$	2890.191
ϕ	0.042	$0.054_{\pm 0.058}$	1.059
Time(s)	1.308	$1.366_{\pm 0.029}$	1.415

Table III
Algorithm performance for $N = 50$

Metric	Min	Mean ± Std	Max
DIN	0	$0.048_{\pm 0.224}$	2
VR	19	$34.657_{\pm 3.762}$	43
TD	1920.378	$4016.131_{\pm 689.647}$	7040.998
ϕ	0.023	$0.077_{\pm 0.224}$	2.039
Time(s)	3.096	$3.279_{\pm 0.097}$	3.477

the player exploration. However, without good support for comparison of results, it is challenging to judge the quality of a solution featuring a TD $= 2000$, TD $= 4000$ or even TD $= 7000$, for example.

Finally, one impressive and robust result is the short execution time. The algorithm is written in Java and tests were performed on an AMD FX-8350 octa-core, with 4.0 GHz. The slowest execution for $N = 25$ was completed with just 1.4 seconds and 3.5 seconds for $N = 50$. This fast execution time comprehends every step, from reading the original level from the disk to writing the puzzle layout on it. With this result, it is possible to affirm that this generator allows an online level replayability. This means the puzzle generation can be applied during the gameplay, even considering common computers or consoles.

The Fig. 1 illustrates the output of the proposed approach for $N = 25$. The Fig. 2 does the same for $N = 50$. The blue squares represent the empty rooms, the cyan square is the *Start* room, the red square is the *Boss* room and the yellow ones are the locations of the keys. The capital letter on a blue room refers to the condition (the required key) to access that room, while a letter on a yellow room denotes the key itself.

In general, we can affirm that the algorithm reaches the goals for which it was designed. To observe that, we suggest the reader take another look on Fig. 1 and Fig. 2 and mentally outline a route to solve the puzzle. The reader may easily realize that, at the end of the route, almost the entire level has been traversed. In addition, on a few occasions, he had to decide on the best route to take, moving forward and backward on a nonlinear pace. These features summarize what we desired to achieve with this paper.

V. Conclusions and Future Works

This paper presented a novel approach for evolving puzzles for adventure or dungeon based games. The algorithm takes as input an already generated level for which a lock-and-key puzzle is produced. Diverging from related literature records, this work is focused on a nonlinear player progression and the maximization of the level exploration. For this purpose, a simple content representation was adopted and a unique evaluation function was conceived. We employ an EA to evolve several puzzle layouts in a fast computational time.

The computational results demonstrated that the algorithm is able to achieve its goals with a high robustness, even over larger levels. Due to its fast execution time, it is possible to conclude that this algorithm can run online, during gameplay. However, it is good to remember that it is a stochastic approach and so we can not guarantee it will achieve good results with hundred percent of certainty. Thus, practical applications would require additional work to ensure the output feasibility.

In future works, we plan to extend both the previous level generator and the current proposal in a multi-layered system. In this plan, both generators work together to produce multi-level content or complex worlds formed by not just one but several integrated dungeons. In addition, in the current state of this project, the puzzle generator demands a level as input. It would be very promising if the puzzle generator outputs could be used to feedback the level generator again, thus creating a virtuous loop, where the resulting content is iteratively refined.

References

[1] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*. Springer International Publishing, 2016.

[2] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, Sep 2011.

[3] C. Ashmore and M. Nitsche, "The quest in a generated world," in *Proceedings of the 2007 DiGRA International Conference: Situated Play, DiGRA 2007*, Tokyo, Japan, Sep 2007, pp. 503–509.

[4] J. Dormans and S. Bakkes, "Generating missions and spaces for adaptable play experiences," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 216–228, Sep 2011.

[5] J. M. Font, R. Izquierdo, D. Manrique, and J. Togelius, "Constrained level generation through grammar-based evolutionary algorithms," in *Applications of Evolutionary Computation*, G. Squillero and P. Burelli, Eds. Cham: Springer International Publishing, 2016, pp. 558–573.

[6] R. van der Linden, R. Lopes, and R. Bidarra, "Procedural generation of dungeons," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 78–89, Mar 2014.

[7] M. Stout, "Learning from the masters: Level design in the legend of zelda," Gamasutra, Jan 2012. [Online]. Available: https://www.gamasutra.com/view/feature/134949/learning_from_the_masters_level_.php

[8] A. S. Ruela and K. V. Delgado, "Scale-free evolutionary level generator," in *Proceedings of the 2018 IEEE Conference on Computational Intelligence and Games (CIG'18)*, Maastricht, The Netherlands, Aug 2018, pp. 245–252.

[9] I. D. Horswill and L. Foged, "Fast procedural level population with playability constraints," in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12*, Stanford, California, USA, Oct 2012, pp. 20–25.

[10] D. Karavolos, A. Bouwer, and R. Bidarra, "Mixed-initiative design of game levels: Integrating mission and space into level generation," in *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015*, Pacific Grove, CA, USA, Jun 2015.