Evolutionary Level Repair

Debosmita Bhaumik
Institute of Digital Games
Msida, Malta
debosmita.bhaumik.22@um.edu.mt

Julian Togelius
Game Innovation Lab
New York, New York, USA
julian@togelius.com

Georgios N. Yannakakis
Institute of Digital Games
Msida, Malta
georgios.yannakakis@um.edu.mt

Ahmed Khalifa
Institute of Digital Games
Msida, Malta
ahmed@akhalifa.com

Abstract

We address the problem of game level repair, which consists of taking a designed but non-functional game level and making it functional. This might consist of ensuring the completeness of the level, reachability of objects, or other performance characteristics. The repair problem may also be constrained in that it can only make a small number of changes to the level. We investigate search-based solutions to the level repair problem, particularly using evolutionary and quality-diversity algorithms, with good results. This level repair method is applied to levels generated using a machine learning-based procedural content generation (PCGML) method that generates stylistically appropriate but frequently broken levels. This combination of PCGML for generation and search-based methods for repair shows great promise as a hybrid procedural content generation (PCG) method.

CCS Concepts

• **Applied computing** → **Computer games**; • **Theory of computation** → **Evolutionary algorithms**.

Keywords

Procedural Level Generation, Generative AI, Evolutionary Algorithms, Quality Diversity Search, Lode Runner

ACM Reference Format:
Debosmita Bhaumik, Julian Togelius, Georgios N. Yannakakis, and Ahmed Khalifa. 2025. Evolutionary Level Repair. In *Genetic and Evolutionary Computation Conference (GECCO '25 Companion), July 14–18, 2025, Malaga, Spain.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3712255.3726692

1 Introduction

Many types of game content, such as levels, exhibit a dual nature where both visual aesthetics and functionality are important. The way the content looks is important, not only for the visual appeal but also for what it communicates about how it can be played. But functionality is, in many games, even more crucial; it determines what is possible to do in the game and what leads to success and failure. For example, a Super Mario Bros level gives a sense of thematic unity, foreshadows challenges, and gently tells the player where to go. Still, it also defines the challenges that need to be overcome. The fusion of visual aesthetics and functionality is central to what a game level is, but functionality is more important. While a playable level that looks like a mess is still passable game content, a beautiful but non-completable level is game-breaking.

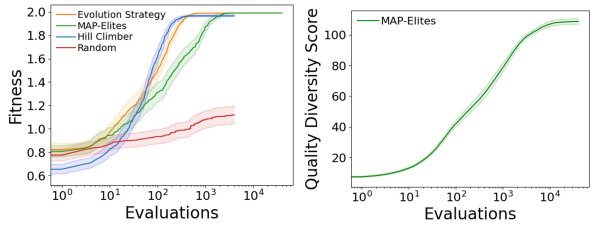
Training (or fine-tuning) a self-supervised learning model of the type usually used for visual data can therefore yield a model that outputs pretty but broken levels. For example, training an autoencoder on a set of levels from the classic puzzle platformer Lode Runner[3] results in a neural network that generates nice-looking levels that mostly cannot be solved. After training GANs to generate Mario levels, most of the naively sampled content is not functional [16]. Many machine learning models can not, on their own, ensure functionality.

So, how can you generate content that both looks good and plays well? One approach is to train a model on existing levels using self-supervised methods, and then use some kind of search algorithm to search the learned space for good content. This is the general approach of latent variable evolution [4, 16] and latent variable illumination [8]. Still, this approach introduces certain constraints on the machine learning method, notably that it has a latent space, which might exclude some ML methods. Even after using these methods and restricting the latent search around the broken level, there is still a chance that the space might not contain a repaired version of it. This reduces the learned latent space to a smaller subset and discards a huge part of the search space because of small mistakes in it, which causes the method to have less creative freedom.

Another approach is to generate and then repair. The problem of level repair then becomes one of making sure the level functions correctly or follows some defined constraints (e.g., that it is playable and contains the appropriate challenges) while maintaining the overall visual aesthetics of the level. Preserving the visual aesthetics can intuitively be achieved by limiting the number of changes made. Different work has combined constraint solving [1, 6], linear programming [18], machine learning models [5, 9], and search-based methods [7, 12] with PCGML to battle the functionality issue. Inspired by the "generate then repair" we are trying to combine the advantage of machine learning methods with evolutionary methods to repair unplayable levels.

GECCO '25 Companion, July 14–18, 2025, Malaga, Spain

ACM ISBN 979-8-4007-1464-1/2025/07
https://doi.org/10.1145/3712255.3726692



(a) Maximum Fitness during optimization. **(b) Quality diversity score of the MAP-Elites algorithm.**

Figure 1: The average performance for all four algorithms with 95% confidence interval over 200 runs.

2 Methods

Repairing broken game levels is not a straightforward task, especially when the goal is to make the level functional. A good repair function will try to make a small number of changes to the existing content to make it playable. If the repaired level differs significantly from the visual aesthetic of the given level, it is generally not a good repair. At the same time, a good evaluation metric is required to measure the changes made by the repair function and guide the process in the right direction. For repairs, we use two different algorithms: $\mu + \lambda$ Evolution Strategy (ES) [2] for normal optimization and MAP-Elites (ME) [13] for Quality Diversity.

We have selected Lode Runner for this project. Lode Runner levels are of size $32x22$ tiles and made of eight different tile types: empty, brick, solid-brick, rope, ladder, gold, enemy, and player. The objective of the game is to collect all the gold in the level without being caught by the enemies. The player can move horizontally on brick tiles and ropes and navigate vertically using ladders. Lode Runner was selected as our domain problem due to its relatively large level, which makes it easy to see when a level looks "noisy".

2.1 Representation and Operators

We use direct representation, where the levels are represented as a 1D array of tiles (flattened representation of the $32x22$ level). For modifying the chromosomes, we only use mutation. The mutation operator picks 1 to m (m is the maximum number of modifications at each step) random locations and changes them. The tile values are changed either randomly by selecting any other tile or copying the corresponding tile from the starting level for that location. The probability of which mutation will be applied is determined based on the fitness value. Lower fitness favors random mutation to explore more towards the beginning of the evolution, while higher fitness encourages copy mutation to ensure that the repaired level still looks similar to the starting level. One important note on the mutation is that we removed the ability to erase or add gold tiles or the player tile to force evolution to repair the level by making it connected instead of removing unreachable gold. Finally, the initial population is created by applying random mutations on the starting level.

2.2 Fitness Function and Characteristics

In our problem, the fitness function needs to deal with playability and similarity. Both are equally important for a good repair function, also difficult to balance. To improve playability, tiles need to be modified, which affects the similarity score in the opposite direction. To solve this, we are using a *cascading elitism* [15] fitness function. This encourages evolution to prioritize playability first, then it tries to improve the similarity as shown in equation 1.

Starting Playability	30% - 50%	50% - 70%	70% - 90%
Random Search	8.8 ± 1.34	10.72 ± 1.72	2.51 ± 1.13
Hill Climber	10.35 ± 1.02	21.43 ± 11.37	35.05 ± 11.39
Evolution Strategy	9.21 ± 1.02	5.4 ± 0.94	4.66 ± 0.65
MAP-Elites	**7.91 ± 0.68**	**4.63 ± 0.67**	**4.22 ± 0.53**

Table 1: Average number of changes applied by each algorithm with 95% confidence interval to repair.

$$f_{total} = \begin{cases} f_{playability} & \text{if } gold_{collect} < gold_{total} \\ 1 + f_{similarity} & \text{otherwise} \end{cases} \quad (1)$$

where $gold_{total}$ is the total number of golds and $gold_{collect}$ is the number of reachable golds.

The playability score consists of the ratio of reachable golds in the level and the exploration score by the flood-fill algorithm, as shown in equation 2. The exploration score is added to help smooth the fitness function. The number of reachable golds is calculated using a flood-fill algorithm from the player's starting location. This algorithm expands only using the possible player's actions at each tile except for digging (for simplicity).

$$f_{playability} = min(1, \frac{gold_{collect} + \frac{tiles_{explored}}{lvl_{size}}}{gold_{total}}) \quad (2)$$

where $gold_{total}$ is the total number of golds, $gold_{collect}$ is the number of reachable golds, $tiles_{explored}$ is the number of tiles explored by the flood-fill agent, and lvl_{size} is the size of the level.

For similarity, we use the hamming distance between the current level and the starting level as shown in equation 3.

$$f_{similarity} = \frac{lvl_{size} - dist(lvl, lvl_{start})}{lvl_{size}} \quad (3)$$

where lvl is the current level, lvl_{start} is the starting level, lvl_{size} is Lode Runner level size (22x32), and $dist(lvl, lvl_{start})$ is the hamming distance between the starting level and the current level.

Behavior characteristics are features that can be calculated for each chromosome. Behavior characteristics help the quality diversity algorithm to find solutions that are different from each other. Since, the connectivity is what we care about towards playability, the different ways to reach this connectivity are important to differentiate between solutions. We used 2 simple behavior characteristics that correlate with connectivity: the number of added Ropes and the Number of added Ladders. The ladder tile is the only tile that allows the player to walk in all 4 directions, and the rope tile allows them to walk in 3 different directions (no up).

3 Experiments

For our experiments, we used 20 unplayable levels, 10 were generated using the Wave Function Collapse algorithm (WFC) [10] and the other 10 levels using a trained autoencoder Bhaumik et al. [3]. We place the player on a randomly selected empty tile, as a good Lode Runner level should be fully connected, meaning that a player can reach all gold tiles from any location.

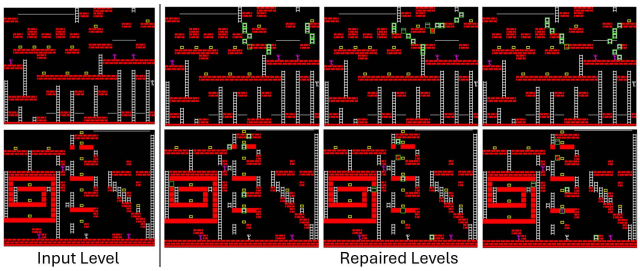
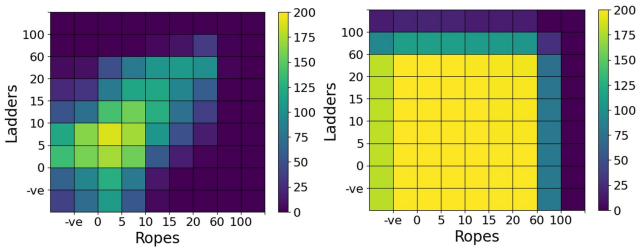


Figure 2: Example of $\mu + \lambda$ evolution strategy repaired levels.



(a) Evolution Strategy **(b) MAP-Elites**

Figure 3: Heatmap of all the discovered solutions over all the runs by $\mu + \lambda$ Evolution Strategy and MAP-Elites.

While generating unplayable levels from WFC and the autoencoder, we discarded any level that had less than 30% playability (the player cannot collect more than 30% of the gold) as repairing these levels might need too many modifications. We wanted to explore the performance of the proposed repair methods on different types of broken levels. We divided the starting levels into three groups based on their starting playability score ($30-50\%$, $50-70\%$, and $70-90\%$).

We have applied the ES and the ME algorithm on each level, and repeated them 10 times to showcase the difference and stability between runs. We ran both evolutions until fitness stopped improving (figure 1a). For the ES, this was 4,000 generations with $\mu = \lambda = 50$, which is 200k evaluations. For ME, this was 2 million evaluations with 100 chromosomes as a start. The number of mutations (m) is limited to a maximum of 10 to allow small changes in the levels. The probability of random mutation and copying tile mutation is linearly sampled between 80/20% at fitness of 0 to 20/80% at fitness of 2. Finally, each behavior characteristic is divided into 9 bins [<0, 0, 5, 10, 15, 20, 60, 100, 100+]. This division is based on some initial experiments where we find that this is the best balance between the number of bins and finding a good solution fast.

We applied a random search algorithm and a hill climber (HC) [14] as baseline algorithms. HC starts with a broken level and at each step selects a random location from the level, then picks the best possible modification at that location. On the other hand, the random search (RS) generates multiple new levels by applying random changes to the starting level (0-20 tiles) and keeps the best-found solution. Both HC and RS ran for 200k evaluations, which is the same number of evaluations as ES.

4 Results

HC, ES, and ME managed to repair all input levels in different time scales, while the RS succeeded only 26% of the time. Figure 1a shows the average of max fitness between all the runs with the 95% confidence interval for different algorithms. It shows the ES and ME reached a little higher fitness compared to the HC in 200k evaluations. We believe that the local nature of HC prevents it from exploring more optimal repairs (that require fewer changes). We see that ME took a little longer to reach the highest fitness and then stabilized. To understand why, we show the quality diversity (QD) score (sum of the fitness of all solutions in the archive) of the ME archive with the 95% confidence interval in figure 1b. We can see that the QD score starts stabilizing around 2 million evaluations. This is reasonable as the MAP-Elites tried to improve the entire archive instead of finding a single solution compared to ES or HC (as shown in figure 4).

To understand more about the repairs, we compared all the algorithms based on the number of changes in each starting level group. Table 1 shows the average number of changes applied on levels with 95% confidence interval. The HC applies higher changes for all groups compared to the others, with the highest values for $50-70\%$ and $70-90\%$ groups. We believe that some of these levels might have more than one way to repair, where the longer repair path has a better fitness landscape increase compared to others. When comparing the changes applied by ME and the ES, for levels with the lowest playability($30-50\%$), the number of changes is maximum, as these levels have a higher number of unreachable golds and to connect them, big changes are required. For the other two groups they are quite close to each other, with ME performing a bit better. We think the behavior characteristics helped the ME to explore different areas of the search space and discover better repairs overall. This can also be seen in figure 3 where the ME algorithm covers the low change cells more compared to the ES.

Figure 2 shows the levels repaired from different runs of the ES. It is visible that the evolution strategy made very few changes to make the level playable. One noticeable attribute is that most repairs involve adding extra ladders, as ladders allow movement in all 4 directions compared to other tiles, but this limits the diversity of the repaired levels as they show similar ways of traversing the levels. Another observation is that the added ladders are placed diagonally instead of a straight line. This is due to the fitness that tries to minimize the amount of tiles modified; diagonal connections allow for more area with fewer changes.

Figure 4 displays the repairs done by the ME algorithm. We show only the bottom corner of the archive (without <0 cells) as we are only interested in solutions that involve a small number of changes. Out of 10 different runs, we display the one with the highest quality diversity score. One interesting feature in these repairs can be seen in the 0 bins of the behavior characteristic. As the algorithm discovers that they can't be solved without adding ladders so the evolution decided to reuse ladders from less needed areas. This is done by removing the ladder from a specific area and adding it to another. Finally, sometimes when the repair does not require extra ladders or ropes, the algorithm just adds random ones to cover that cell in the archive. This problem can be solved by changing the behavior characteristics to only show used ladders and ropes.

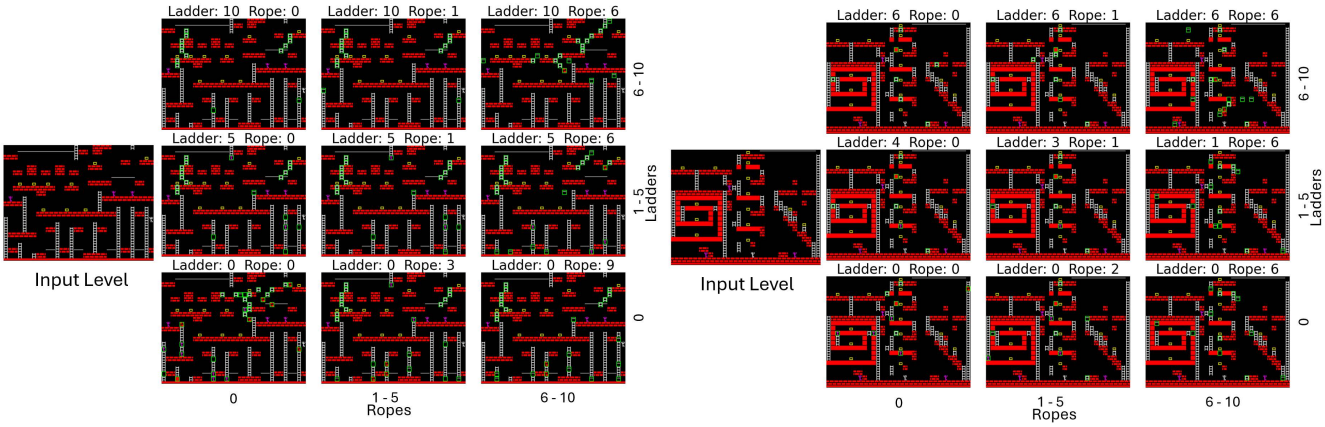


Figure 4: The repaired levels from the bottom corner of the MAP-Elites archive (Ropes and Ladders as behavior characteristic).

To understand the diversity of solutions coming from ES compared to ME, we save playable levels from each generation of ES in an archive with a similar structure to the ME archive (figure 3). We compared the number of archive cells covered by both algorithms over all the runs. On average, the ES managed to fill 25% of the archive cells (with 5% confidence interval) whereas MAP-Elites covered 68% (with 8% confidence interval) of cells in the archive. This was expected as the ES tried to repair levels using minimum changes without caring about the tiles used. Another observation is that the ES focused on the diagonal parts of the archive with most cells in the lower left area, except for the 0 and negative bins. This was expected as the ES tries to reduce the number of changes from the original level.

5 Discussion & Conclusion

We showed that evolutionary algorithms can be used to efficiently repair broken levels. The proposed methods were compared with two baseline methods, random search and hill climber. The hill climber managed to achieve a close fitness to our proposed methods, but it tends to use a large number of changes (due to their local search nature). On the other hand, the random baseline failed to repair levels 74% of the time. Evolution Strategy was able to find solutions in 10 times fewer evaluations than MAP-Elites, but with very little control over how the repairs are happening. Having control over what type of repairs is considered better for mixed-initiative tools [17]. Finally, the MAP-Elites algorithm overall manages to do better repairs with fewer changes than the Evolution Strategy; it showcases the ability of diversity in helping to find better repairs. Perhaps the biggest concern for the method proposed in this paper is the time complexity. Evolutionary PCG is not the fastest method, particularly if a non-trivial evaluation function is used. This raises the question of whether we could train a machine learning method that imitates the repair trajectories discovered here, similar to the work done by Khalifa et al. [11]. That would make level repairing more viable for real-time use cases such as design assistance.

References

[1] Mahsa Bazzaz and Seth Cooper. 2024. Guided game level repair via explainable AI *(AIIDE '24)*. AAAI Press. https://doi.org/10.1609/aiide.v20i1.31874

[2] Hans-Georg Beyer. 2007. Evolution strategies. *Scholarpedia* 2, 8 (2007), 1965.

[3] Debosmita Bhaumik, Ahmed Khalifa, and Julian Togelius. 2021. Lode Encoder: AI-constrained co-creativity. In *Conference on Games*. IEEE.

[4] Philip Bontrager, Aditi Roy, Julian Togelius, Nasir Memon, and Arun Ross. 2018. Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution. In *Conference on Biometrics Theory, Applications and Systems*. IEEE.

[5] Eugene Chen, Christoph Sydora, Brad Burega, Anmol Mahajan, Abdullah Abdullah, Matthew Gallivan, and Matthew Guzdial. 2020. Image-to-level: Generation and repair. In *Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI.

[6] Seth Cooper. 2022. Sturgeon: tile-based procedural level generation via learned and designed constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

[7] Seth Cooper and Anurag Sarkar. 2020. Pathfinding Agents for Platformer Level Repair.. In *AIIDE Workshops on Experimental AI in Games*. AAAI.

[8] Matthew C Fontaine, Ruilin Liu, Ahmed Khalifa, Jignesh Modi, Julian Togelius, Amy K Hoover, and Stefanos Nikolaidis. 2021. Illuminating mario scenes in the latent space of a generative adversarial network. In *Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI.

[9] Rishabh Jain, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. 2016. Autoencoders for level generation, repair, and recognition. In *ICCC workshop on computational creativity and games*. IEEE.

[10] Isaac Karth and Adam M Smith. 2017. WaveFunctionCollapse is constraint solving in the wild. In *Foundations of Digital Games Conference*. ACM.

[11] Ahmed Khalifa, Julian Togelius, and Michael Cerny Green. 2022. Mutation models: Learning to generate levels by imitating evolution. In *Foundations of Digital Games Conference*. ACM.

[12] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2013. Sentient sketchbook: computer-assisted game level authoring. In *Foundation of Digital Games Conference*. ACM.

[13] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).

[14] Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[15] Julian Togelius, Renzo De Nardi, and Simon M Lucas. 2007. Towards automatic personalised content creation for racing games. In *Symposium on Computational Intelligence and Games*. IEEE.

[16] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Genetic and Evolutionary Computation Conference*. IEEE.

[17] Georgios N Yannakakis, Antonios Liapis, and Constantine Alexopoulos. 2014. Mixed-initiative co-creativity. In *Foundations of Digital Games Conference*. ACM.

[18] Hejia Zhang, Matthew Fontaine, Amy Hoover, Julian Togelius, Bistra Dilkina, and Stefanos Nikolaidis. 2020. Video game level repair via mixed integer linear programming. In *Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009