Mario Level Generation From Mechanics Using Scene Stitching

Michael Cerny Green[*]
Game Innovation Lab
New York University
Brooklyn, USA
mike.green@nyu.edu

Luvneesh Mugrai[*]
Game Innovation Lab
New York University
Brooklyn, USA
lm3300@nyu.edu

Ahmed Khalifa
Game Innovation Lab
New York University
Brooklyn, USA
ahmed@akhalifa.com

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

***Abstract*—Video game tutorials allow players to gain mastery over game skills and mechanics. To hone players' skills, it is beneficial from practicing in environments that promote individual player skill sets. However, automatically generating environments which are mechanically similar to one-another is a non-trivial problem. This paper presents a level generation method for Super Mario by stitching together pre-generated "scenes" that contain specific mechanics, using mechanic-sequences from agent playthroughs as input specifications. Given a sequence of mechanics, the proposed system uses an FI-2Pop algorithm and a corpus of scenes to perform automated level authoring. The proposed system outputs levels that can be beaten using a similar mechanical sequence to the target mechanic sequence but with a different playthrough experience. We compare the proposed system to a greedy method that selects scenes that maximize the number of matched mechanics. Unlike the greedy approach, the proposed system is able to maximize the number of matched mechanics while reducing emergent mechanics using the stitching process.**

***Index Terms*—Super Mario Bros, Feasible-Infeasible 2-Population, Evolutionary Algorithms, Stitching, Design Patterns, PCG, Experience Driven PCG**

I. Introduction

Procedural Content Generation (PCG) methods are used to create content for games and simulations algorithmically. A large variety of PCG techniques exist, which have been applied to a wide variety of types of game content and styles. Sometimes developers write down the rules that a program must abide to when generating, like a constraint-satisfaction generative system [1]. Other methods let users define objective functions to be optimized by for example evolutionary algorithms [2], and yet others rely on supervised learning [3], or reinforcement learning [4]. Experience Driven PCG [5] is an approach to PCG that generates content so as to attempt to enable specific player experiences. This includes attempts to produce stylized generated content such as levels of specific difficulty [6], for different playstyles [7], or even for a specific player's playing ability [5]. In the vein of Experience Driven PCG, this paper presents a method of level generation which attempts to create a level in which a player triggers the same game mechanics as the target level, while being structurally different.

Evolving entire levels to be individually personalized is not trivial. In this paper we propose a method of multi-step evolution. Mini-level "scenes" that showcase varying subsets of game mechanics were evolved in previous research [8] using a Constrained Map-Elites algorithm [7] and stored within a publicly available library[*]. The system uses these scenes with an FI-2Pop [9] optimizer to evolve the best "sequence of scenes," with the target being to exactly match a specific player's mechanical playtrace.

By breaking this problem down into smaller parts, the system is to focus more on the bigger picture of full level creation. Our work goes a step farther than Reis et al's [10] generator which stitches together (human-evaluated/ human-annotated) scenes. Khalifa et al. [8] developed a generator which automatically creates the scene corpus with simplicity and minimalism in mind. The levels that the system generates reflect a type of minimalist design that can be used as a tutorial to learn a target mechanic sequence.

Generating levels using this method would allow a game system to provide a player with unlimited training for specific mechanic sequences that they struggle on, to help the players to hone on these mechanics instead of memorizing a single level by replaying it over and over again. Furthermore, game designers could use such a system as inspiration for level design, perhaps modifying the level after generation without having to start from scratch.

II. Background

A. Search-Based Procedural Content Generation

Search-based PCG is a generative technique that uses search methods to find good content [2]. In practice, evolutionary algorithms are often used, as they can be applied to many domains, including content and level generation in video games. Search-based PCG has been applied in many different game frameworks and games, such as the General Video Game AI framework [11], PuzzleScript [12], Cut the Rope (ZeptoLab, 2010) [13], and Mazes [6]. The research in this paper drew inspiration from several previous works [14, 15, 16] where the authors developed evolutionary processes which divided

[*] Both authors contributed equally to this research.

978-1-7281-4533-4/20/$31.00 ©2020 IEEE

[*]https://github.com/LuvneeshM/MarioExperiments/tree/master/scenelibrary

level generation into separate micro- and macro-evolutionary problems in order to obtain better results for large game levels.

B. Mario AI Framework

Infinite Mario Bros. [17] is a public domain clone of Super Mario Bros (Nintendo 1985). Just like in the original game, the player controls Mario by moving horizontally on two-dimensional maps towards a flag on the right hand side. Players can walk, run, and jump to traverse the level. Depending on Mario's mushroom consumption, the player may withstand up to 2 collisions with an enemy before losing and even shoot fireballs. Players may also lose if Mario falls down a gap.

The Mario AI framework [18] is an artificial intelligence research environment built from Infinite Mario Bros. The framework supports AI agents, level generators, and a huge set of levels (including the original mario levels). This framework has been used for AI competitions in the past [18, 19], as well as research into AI gameplay [18] and level generation [20, 21].

C. Level Generation in Mario AI

Several level generators exist in the Mario AI Framework as either artifacts of past competitions or from research projects done independently of them. Competitions for level generation were hosted in 2010 and 2012 [19]. A detailed list of these generators was created by Horne et al. [22], and includes probability generators [23], adaptive generators [20], rhythmic grammars [24], pattern-based generators [25], and grammatical evolution [20]. Similar to the generator in this paper, the *Occupancy-Regulated Extension* generator stitches small hand-authored level templates together to create complete levels [20]. A core difference between the Occupancy-Regulated Extension generator and the work in this paper is that the level templates here are generated rather than handmade.

Outside of competition generators, Summerville et al.[26] generates levels using Markov Chain Monte Carlo Tree Search, which builds levels using a level-pattern representation similar to the "scenes" in our work. Snodgrass et al. [27, 28, 29] uses Markov Chains to procedurally generated IMB levels in a controlled manner. A Long-Short Term Memory network has also been shown to be a successful Mario level generator, either by training on the actual mario levels [30] or trained on video traces [31]. Volz et al. [32] generate playable levels by searching the latent space of a trained Generative Adversarial Network (trained using the original Mario levels) using Covariance Matrix Adaption Evolutionary Strategy. Sarkar et al. [33] use Variational Autoencoders to identify latent representations of both Mario and *Kid Icarus* (Nintendo 1986), with the purpose of generating blended levels containing characteristics of both.

D. Personalized Generation

Several research projects have attempted to generate game levels personalized for a specific player or playstyle. The educational game *Refraction* generates levels using answer set programming to target particular level features [34]. Khalifa et al. [7] evolved levels for bullet hell games via constrained Map-Elites, a hybrid evolutionary search, using automated playing agents that mimicked different human playstyles. Within the Mario AI Framework, Khalifa et al. [8] generated mini-levels called "scenes" using constrained Map-Elites. These scenes focus on requiring the player to trigger a specific mechanic in order to win. The corpus of "scenes" that were publicly available from Khalifa et al.'s project is the same library used in this paper (see Section IV).

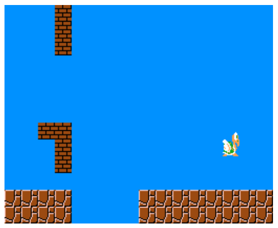


Fig. 1: A example a scene from the system's scene library

Name	Description	Frequency
Low Jump	Mario performs a small hop	25.9%
High Jump	Mario jumps very high	39.44%
Short Jump	Mario jumps and hardly moves forward	28.33%
Long Jump	Mario jumps and moves forward a large amount	19.75%
Stomp Kill	Mario kills an enemy by jumping on it	78.77%
Shell Kill	An enemy is killed by a koopa shell	37.85%
Fall Kill	An enemy falls off the game screen	50%
Mode	Mario changes his mode (small, big, and fire)	22.77%
Coin	Mario collects a coin	50.5%
Brick Block	Mario bumps into a brick block	41.1%
? Block	Mario bumps into a ? mark block	59.79%

TABLE I: A list of the Mario game mechanics and the percentage of evolved scenes that contain them.

III. Methods

Using a target mechanic playtrace, our system is able to generate levels that are mechanically analogous to the input. The method proposed in this paper uses a set of pre-evolved "scenes", which are stitched together to create a full level. A scene is a mini-level that encapsulates a certain idea [35]. Figure 1 shows a scene in which Mario must jump the gap and may also encounter a flying koopa.

Every scene is labelled with the mechanics that an agent triggered while playing it. Table I shows all the game mechanics in the Mario AI Framework that scenes may be labelled with. The scene library used in this system, originally evolved using a MAP-Elites algorithm to heavily promote the use of sub-sets of game mechanics in a previous project [8], encapsulates many feasible mechanic combinations (see Table I). The table also shows the percentage of scenes from the corpus that contains specific mechanics. Jump is the most common mechanic which is not surprising since Mario is a jumping platform game (jumping over gaps, jumping on enemies, jumping to headbutt blocks, etc). In the following subsections, we will explain each part of the FI-2Pop algorithm [9] that our system uses to stitch pre-generated scenes into playable Mario levels that an agent plays by triggering a specific game mechanic sequence.

A. Chromosome Representation

A level consists of a number of scenes stitched together. A chromosome is synonymous with its level representation, and each scene within this chromosome is not limited to only having one mechanic label. Using scenes that contain multiple mechanics enables the generator to generate levels that are more condensed.

B. Genetic Operators

The system uses mutation and crossover as operators. Two-point crossover allows the system to increase and decrease level length as well as swap-out any number of scenes, from a single scene to the entire level. The generator uses five mutation types:

- **Delete:** delete a scene.
- **Add:** add a random scene adjacent to a scene. The random scene is selected with probability inversely proportional to number of mechanics using rank selection.
- **Split:** split a scene in half and replace it with a left and new right scene, randomly selecting half the mechanics to go in a new left scene and the rest to go in the right.
- **Merge:** add the mechanics of a scene to the left or right scene, then replace both scenes with one from the corpus that has the combined list of mechanics.
- **Change:** changes a scene with another random scene. The random scene is selected with probability inversely proportional to number of mechanics using rank selection.

The system selects a scene to apply one of the operators, with a higher likelihood to select scenes with higher numbers of mechanics.

C. Constraints and Fitness Calculation

FI-2Pop uses both a feasible and an infeasible population. The infeasible population tries to make the chromosomes satisfy constraints (like making sure the level is playable), while the feasible population tries to make sure that the mechanics in the new levels are similar to the input mechanic sequence.

To calculate the constraints, an agent plays each chromosome N times. The system calculates the constraint value using the following equation:

$$C = \begin{cases} \frac{1}{N}\sum_{i=1}^{N}\frac{d_i}{d_{level}} & \text{if } \frac{1}{N}\sum_{i=1}^{N} w_i < p \\ 1 & \text{if } \frac{1}{N}\sum_{i=1}^{N} w_i \geq p \end{cases} \tag{1}$$

where d_i is the distance traveled by the A* agent on the level on the i^{th} iteration, d_{level} is the maximum length of the level, w_i is equal to 1 if the agent reached the end of the level on the i^{th} run, and p is the threshold percentage.

To calculate the fitness, the system uses the playtrace in which the agent not only won the level but also triggered the fewest mechanics. The level is assigned an initial score S, which is decremented based on the number of "faults". A fault is a mechanic sequence mismatch between the input sequence, which is the target mechanic playtrace, and the newly generated agent playtrace, defined either as an extra mechanic placed between a correct subsequence of mechanics, or else as a missing mechanic that would create an otherwise correct subsequence. Figure 2 displays an example of both. The system uses a sequence matching algorithm to calculate fault counts, as shown by Algorithm 1.

Algorithm 1: Calculating fault count in a chromosome

```
GIVEN: generatedSeq, targetSeq
pTarget = 0; pGenerated = 0;
extraMechs = 0; missedMechs = 0;
for pTarget < len(targetSeq), pTarget += 1 do
    targetMechanic = targetSeq[pTarget];
    genSubList = generated-
      Seq.subList(pGenerated,len(generatedSeq));

    mechanicIndex =
      genSubList.indexOf(targetMechanic);
    if mechanicIndex == -1 then
        missedMechs += 1;
    else
        pGenerated += mechanicIndex + 1;
        extraMechs += mechanicIndex;
    end
    if pGenerated >= len(generatedSeq) then
        pTarget += 1;
        break;
    end
end
return extraMechs, missedMechs
```

Algorithm 1 loops over the target sequence and searches for the first occurrence of each target mechanic in a sub-array of the generated mechanics list, from the beginning to the end of the list. If the mechanic is not found, it increments a counter tracking the number of missed mechanics. When the mechanic is found, the index of the mechanic is the number of extra mechanics between it and the previous matching mechanic. The pointer of the generated mechanic sequence is moved to point at this new position to continue the loop.

Based on how faults are calculated, it is possible for a level to have a negative fitness score. The fitness function is calculated based on the following equation:

$$\begin{aligned} P_{missed} &= W_{missed} \cdot M_{missed} \\ P_{extra} &= W_{extra} \cdot \tanh(b \cdot M_{extra}) \\ F &= S - (P_{missed} + P_{extra} \cdot (S - P_{missed})) \end{aligned} \tag{2}$$

where P_{missed} is the penalty of missed mechanics, P_{extra} is the penalty of extra mechanics, M_{missed} is the number of missed mechanics, M_{extra} is the number of extra mechanics, S is the initial starting value, and b, W_{missed}, W_{extra} are predefined weights.

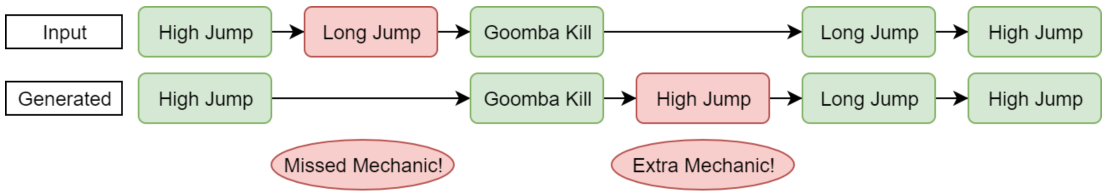


Fig. 2: An example of missing and extra mechanic faults. The Long jump in the input playthrough is missing in the newly generated map's playthrough. The generated playthrough also contains an extra high jump not included in the input.

IV. Experiments

To test the algorithms, we generate levels using three original Super Mario levels as targets (1-1, 4-2, 6-1 in Figures 3a, 3d, and 3g). The Robin Baumgarten A* algorithm, which was developed for the first Mario AI competition [36], is run once on each of the three levels. The resulting mechanic sequences are collected and used as targets.

Our system uses population of 250 chromosomes each generation, with 70% crossover and 20% mutation rates, and 1 elite. Chromosomes are initialized using a random scene picker, which selects anywhere between 5 to 25 scenes with which to populate the level. Each scene is randomly selected from the corpus based on the assigned number of mechanics to it. The number of mechanics for each scene is sampled from a Gaussian distribution with a mean as the average number of mechanics in the target and standard deviation of 1. For the constraints calculation, we used p equal to 1, while for the fitness calculation, we used W_{missed} equal to 5.0, W_{extra} equal to 1.0, b equal to 0.065, and S initially equal to 100. These values were picked based on some preliminary experiments that proves that they lead to the better performance.

The scene corpus is taken from the results of *Intentional Computational Level Design* [8]. The corpus [†] contains a total of 1691 Mario scenes, with an average of 5.45 mechanics in a scene and a standard deviation of 1.74. The library is generated using the Constrained MAP-Elites generator created in the aforementioned project, with the dimensions corresponding to the mechanics shown in Table I. The game playing agent in the MAP-Elites generator is the same agent we use: the winning A* agent from the 2013 Competition[19].

We compare the results from the our system against two baselines. The random baseline generates levels with 5 to 25 scenes which are picked randomly from the corpus. The greedy baseline generates levels with 5 to 25 scenes, selected such that the resulting level maximizes the number of matched mechanics based on each scenes labeled mechanics in the corpus. Each generator generates levels until it creates a total of 20 levels considering each world. Table II displays information about the mechanic makeup of the input playtraces. We can notice that all the playtraces have low frequency of killing enemies, hitting blocks, or collecting coins. This was not surprising as the used A* agent is designed to reach the furthest to the right in the least amount of time without caring about its score.

[†]https://github.com/LuvneeshM/MarioExperiments/tree/master/scenelibrary

Mechanic	Level 1-1	Level 4-2	Level 6-1
Low Jump	14	20	18
High Jump	4	9	4
Short Jump	6	16	14
Long Jump	11	12	7
Stomp Kill	1	2	0
Shell Kill	0	0	0
Fall Kill	0	0	0
Mode	0	0	0
Coin	1	6	1
Brick Block	0	0	0
? Block	2	2	0
Total	39	67	44

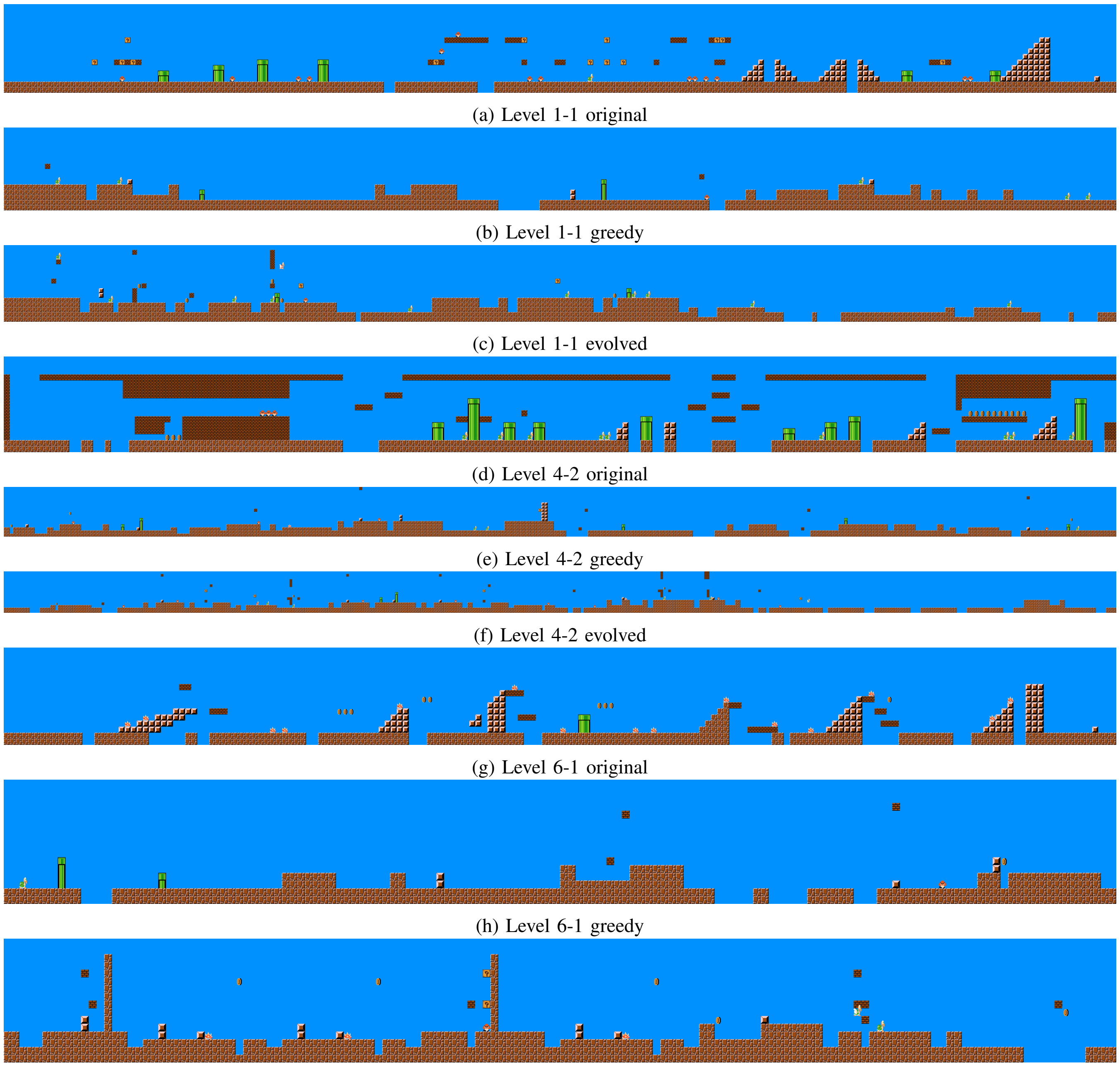
TABLE II: The frequency of each mechanic in the input playtrace.

Experiment	Playability	Inter-TPKLDiv	Intra-TPKLDiv
Original Levels	52%	0.715 ± 0.410	-
Random Levels 1-1	10.75%	0.697 ± 0.265	2.941 ± 1.005
Greedy World 1-1	28.5%	0.675 ± 0.228	2.636 ± 0.795
Evolution World 1-1	100%	0.269 ± 0.127	1.601 ± 0.573
Random Levels 4-2	10.75%	0.697 ± 0.265	2.941 ± 1.005
Greedy World 4-2	26.25%	0.648 ± 0.181	3.329 ± 0.647
Evolution World 4-2	99.5%	0.264 ± 0.094	1.997 ± 0.466
Random Levels 6-1	10.75%	0.697 ± 0.265	2.941 ± 1.005
Greedy World 6-1	25%	0.648 ± 0.172	2.601 ± 0.577
Evolution World 6-1	87.25%	0.348 ± 0.117	1.505 ± 0.404

TABLE III: Different level statistics calculated over 20 generated levels using different techniques and compared to the original levels as a reference point.

V. Results

Figure 3 shows the original levels from Super Mario Bros (Nintendo, 1985) and its greedy and evolved counterparts. Both greedy and evolved levels have less graphical variance, a result of the lack of diversity in the scenes they stitched together. However, scene containing a 3-tile-high-pipe requires the same jump that 3 breakable brick tiles stacked on top of each other. The fitness function used to evolve these scenes [8] most likely negatively impacts level diversity, as it aims to create simple and uniform spaces. Most of the generated levels (greedy or evolution) seem flatter than their original counterparts. The fitness function used for evolving the scenes [8] implies a pressure for lower tile variance, and therefore impacts levels created with the scenes in a similar way.



(a) Level 1-1 original

(b) Level 1-1 greedy

(c) Level 1-1 evolved

(d) Level 4-2 original

(e) Level 4-2 greedy

(f) Level 4-2 evolved

(g) Level 6-1 original

(h) Level 6-1 greedy

(i) Level 6-1 evolved

Fig. 3: A random sampling of greedy-generated and system-evolved levels, compared to their original equivalents

A. *Level Playability*

To calculate playability for each group of levels, we run Robin Baumgarten's A* agent [36] 20 times per level and average the results over the whole group of levels. Table III shows the playability percentages of each of these groups. We find it notable that the A* can only win 52% of the original levels, which includes all the levels from the original Super Mario Bros except for the underwater levels and castle levels. This demonstrates that the A* is not a perfect algorithm and not able to beat every level every time. The evolved 1-1, 4-2, and 6-1 levels all have close to 100% playability. In contrast, greedy stitching seems to make poor quality levels in terms of playability ($25-28.5\%$). Random stitching predictably creates barely playable levels (10%).

B. *Mechanic Similarity*

Figure 4 displays the mechanic evaluation across all three target levels for all generators. "Matches" are the number of matched mechanics while "Extras" are the extra generated mechanic between the input sequence and generated agent playtrace. "Matches" and "Extras" are normalized using the *total* value from Table I for each level. As fitness is impacted

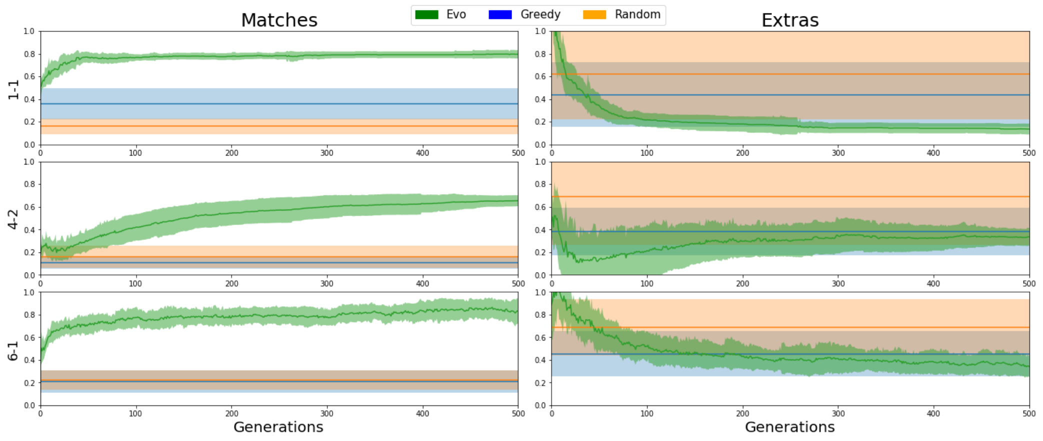


Fig. 4: Tracking mechanic statistics throughout generations across all three target levels for all generators.

by matching mechanics it makes sense that the evolutionary generated levels go up over time, just as fitness does. Across all three levels, the evolution agent far outperforms both the greedy and random generators. This is also true for the extra mechanics (which are minimized) on 1-1 and 6-1. However, on 4-2 the evolutionary generator seems to add more extra mechanics after a brief drop ending around generation 50 which might be inevitable to increase the matched mechanics. This stabilizes around generation 200, which is also when the match mechanic count stabilizes.

C. Structural Diversity

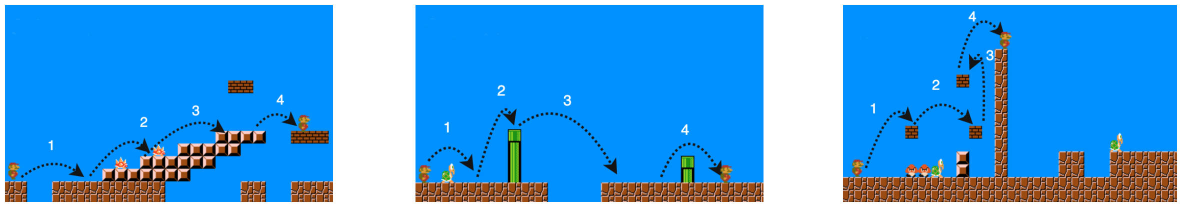
We use Tile Pattern KL-Divergence (TPKLDiv) calculations [37] to measure the structural similarity between the 20 generated levels which we call Inter-TPKLDiv and between the generated levels and the corresponding original level which we call Intra-TPKLDiv. We use a 3x3 tile pattern window to calculate the TPKLDiv. For the Inter-TPKLDiv, this value reflects how different these 20 levels are to each other. In order to calculate it, we calculate the TPKLDiv between each level and the remaining levels and take the average of the minimum 20 values such that all the levels are present. For the Intra-TPKlDiv, this value reflects how different the generated levels to the original level. We compute this value by calculating the TPKLDiv between each generated level and the original level and then we compute the average over all these values.

Table III shows both Inter-TPKLDiv and Intra-TPKLDiv for the 20 generated levels. From observing the values of the Inter-TPKLDiv, it is obvious that every technique has a lower diversity score than the original levels of Super Mario Bros. We noticed that the random generated levels have a lower TPKLDiv value which indicates that the evolved levels have less diversity overall. Also, the evolutionary levels generated for World 1-1 have low diversity relative to the others. In the context of it being the very first level of the game with the most basic and simple mechanics for new players, this diversity score makes sense. This makes it harder to find more diverse structural levels. It is surprising that World 4-2 has a less diversity than World 1-1. The World 4-2 playtrace has so many fired mechanics, it might be more difficult to find a playable level with such a high amount even with 25 scenes, especially when scenes with small numbers of mechanics are selected more often. One last note, the greedy algorithm has a higher diversity than all the evolution levels, but at the same time they less likely to be playable.

By observing the Intra-TPKLDiv values, the random generator's levels have the largest TPKLDiv except for in World 4-2 where we were surprised to find that greedy generator has a higher value. This might be due to the greedy generator creating longer levels in reaction to the longer length of the input mechanic sequence for that level. The evolved levels have nearly half the TPKLDiv value of the random generator levels except for World 4-2, also probably in response to that playtrace's mechanic count. To reference the findings presented in Table III, 4-2 levels seem to contain only small Inter-TPKLDiv (all the 20 generated levels are structurally similar to each other) but show an increase in the Intra-TPKLDiv (all the 20 generated levels are structurally different from the original level).

VI. Discussion

Looking to results as shown by Figure 4, it is clear the FI-2Pop generator outperformed the baselines in the defined terms of matching the mechanic sequence of the input. As we allow for the length of the levels to vary within a defined range, it is important to observe the convergence of said level lengths. A typical level from Super Mario is 14 scenes in length and the levels generated for level 1-1 and level 6-1 converge toward and hover around that length. However, the generator for level 4-2 converges to a length of roughly 23 scenes, nearly 1.64 times that of the other 2 observed levels. We presume the reason behind this influx in scene length is due to sheer number of mechanics present in the original level 4-2. Level 4-2 has the most number of mechanics triggered, close to 1.7 times the amount of level 1-1 and to 1.5 times the amount for level 6-1. We believe, the generator could not guarantee levels where the input mechanic sequence occurred in the given length and thus favored levels with more scenes. Spreading the



(a) Original Sequence (b) Greedy Sequence (c) Evolved Sequence

Fig. 5: An example of a 4 jump sequence in the original 6-1 level, and how the two generators try to copy it.

mechanics allows the generator to better guarantee generating levels with a higher likelihood of being aligned to the input from a mechanics sequence standpoint. This is evident as the number of matches increases as the overall length of the level increases for level 4-2. Since the overall length of level 4-2 increases, the likelihood of additional mechanics occurring between the wanted mechanics also increases, explaining the rise in extra mechanics for the generated levels for 4-2.

The evolutionary generator is influenced to ensure mechanics from the input sequence are forced to happen in their particular order in the generated levels. Evolution is driven by the matching pressure in the fitness function to guarantee the agent had no other choice but to perform certain mechanics before progressing forward. Figure 5 shows an example of this with a zoom into a subsection of 6-1 from the original level, greedy generated level and evolved level. The original level requires the agent to perform a 4-jump sequence in order to progress forward in the level. A similar manner of triggered mechanics can be seen in both the greedy and evolved levels. The greedy generator simply places scenes next to each other in which jumps occur. However, it cannot guarantee or force the agent to perform the jumps outside of having a strong likelihood of the jumps occurring. If the agent was delayed to react it might stomp on the Koopa instead of jumping over it, leading to a different mechanic sequence. Looking to the evolved example, we see there is a long wall that acts as a hard gate, blocking the agent from progressing forward, without first performing the 4 jumps.

The corpus from Khalifa et al. [8] was built using a fitness function that encouraged simplicity, creating a minimalized pressure that is reflected on the levels built using the corpus. In Figure 5, the "original" two-scene section that requires 4 jumps to overcome translates to another two scene section with a Koopa turtle, a pipe, a gap, and a little wall in the "greedy" level section. The evolutionary generator, driven by matching pressure, shrinks this down into a single scene to force the agent into having no other choice but this mechanic sequence. In a way, the evolution method is performing a type of minimalist level generation, creating the simplest levels which are mechanically analogous to the original.

Based on the results of Section V-C and Table III, we hypothesize that a mechanical sequence populated with small amounts of relatively simple mechanics (Evolution World 1-1) has only a small range of mechanically similar cousins. It seems that having a large population of mechanics also makes it difficult to curate diversity (World 4-2). It is possible that World 6-2 represents a sweet spot in terms of diversity. In future work, we'd like to test this theory using a more diverse array of scenes and levels.

VII. Conclusion

In this paper, we explore a means to automatically generate personalized content by stitching pre-generated scenes to construct levels for Super Mario. We compare the sequence in which mechanics occur in the playthrough of the Robin Baumgarten A* agent on three unique levels from the original Super Mario (1-1, 4-2 and 6-1) to the sequences the same A* agent would take in the generated levels from the various methods to judge the success of the experiments. We use the FI-2Pop evolution algorithm, with the focus of developing winnable levels that are mechanically analogous to the original level, and we find that it is able to match the sequence much better than either baseline. Although both baselines and the new method have lower diversity among themselves than the original Mario levels, this is most likely a result of the entropy pressure during the scene generation process.

While these results are promising, this work can be further expanded to examine playthroughs from personified agents or human testers to generate levels unique for their playstyles. We would like to observe how the generated levels for various groups of users differ in what the focus of the level becomes based off how a user plays the initial Super Mario level. For example, a generated level could focus on trying get coins in hard to reach places, like areas in which the player needs to reach both a certain height and perform a long jump. With a user study, it would be possible to gain understanding of player preferences between the original level and personalized generated levels.

Acknowledgments

Michael Cerny Green acknowledges the financial support of the SOE Fellowship from NYU Tandon School of Engineering. Ahmed Khalifa acknowledges the financial support from NSF grant (Award number 1717324 - "RI: Small: General Intelligence through Algorithm Invention and Selection.").

References

[1] A. M. Smith and M. Mateas, "Answer set programming for procedural content generation: A design space approach," *T-CIAIG*, vol. 3, no. 3, 2011.

[2] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *T-CIAIG*, vol. 3, no. 3, 2011.

[3] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *ToG*, vol. 10, no. 3, 2018.

[4] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "Pcgrl: Procedural content generation via reinforcement learning," *arXiv preprint arXiv:2001.09212*, 2020.

[5] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *Transactions on Affective Computing*, vol. 2, no. 3, 2011.

[6] D. Ashlock, C. Lee, and C. McGuinness, "Search-based procedural generation of maze-like levels," *T-CIAIG*, vol. 3, no. 3, 2011.

[7] A. Khalifa, S. Lee, A. Nealen, and J. Togelius, "Talakat: Bullet hell generation through constrained map-elites," in *GECCO*. ACM, 2018.

[8] A. Khalifa, M. C. Green, G. Barros, and J. Togelius, "Intentional computational level design," in *GECCO*. ACM, 2019.

[9] S. Kimbrough, G. Koehler, M. Lu, and D. Wood, "Introducing a feasible-infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch," *European Journal of Operational Research*, vol. 190, 10 2008.

[10] W. M. Reis, L. H. Lelis *et al.*, "Human computation for procedural content generation in platform games," in *CIG*. IEEE, 2015.

[11] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General video game ai: Competition, challenges and opportunities," in *AAAI Conference on Artificial Intelligence*, 2016.

[12] A. Khalifa and M. Fayek, "Automatic puzzle level generation: A general approach using a description language," in *CCG Workshop*, 2015.

[13] N. Shaker, M. Shaker, and J. Togelius, "Evolving playable content for cut the rope through a simulation-based approach." in *AIIDE*, 2013.

[14] C. McGuinness and D. Ashlock, "Decomposing the level generation problem with tiles," in *CEC*. IEEE, 2011.

[15] S. Dahlskog and J. Togelius, "A multi-level level generator," in *CIG*. IEEE, 2014.

[16] J. Togelius and S. Dahlskog, "Patterns as objectives for level generation," in *Proceedings of the Second Workshop on Design Patterns in Games;*. ACM, 2013.

[17] M. Persson, "Infinite mario bros," *Online Game). Last Accessed: December*, vol. 11, 2008.

[18] S. Karakovskiy and J. Togelius, "The mario ai benchmark and competitions," *T-CIAIG*, vol. 4, no. 1, 2012.

[19] J. Togelius, N. Shaker, S. Karakovskiy, and G. N. Yannakakis, "The mario ai championship 2009-2012," *AI Magazine*, vol. 34, no. 3, 2013.

[20] N. Shaker, J. Togelius, G. N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi *et al.*, "The 2010 mario ai championship: Level generation track," *T-CIAIG*, vol. 3, no. 4, 2011.

[21] N. Sorenson and P. Pasquier, "Towards a generic framework for automated video game level creation," in *EvoStar*. Springer, 2010.

[22] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of procedural level generators in the mario ai framework." Society for the Advancement of the Science of Digital Games, 2014.

[23] N. Shaker, G. N. Yannakakis, and J. Togelius, "Feature analysis for modeling game content quality," in *CIG*. IEEE, 2011.

[24] G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha, "Launchpad: A rhythm-based level generator for 2-d platformers," vol. 3, no. 1. IEEE, 2011.

[25] S. Dahlskog and J. Togelius, "Patterns as objectives for level generation," 2013.

[26] A. J. Summerville, S. Philip, and M. Mateas, "Mcmcts pcg 4 smb: Monte carlo tree search to guide platformer level generation," in *AIIDE*, 2015.

[27] S. Snodgrass and S. Ontanon, "A hierarchical approach to generating maps using markov chains," in *AIIDE*, 2014.

[28] ——, "A hierarchical mdmc approach to 2d video game map generation," in *AIIDE*, 2015.

[29] S. Snodgrass and S. Ontanón, "Controllable procedural content generation via constrained multi-dimensional markov chain sampling." in *IJCAI*, 2016, pp. 780–786.

[30] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," *arXiv preprint arXiv:1603.00930*, 2016.

[31] A. Summerville, M. Guzdial, M. Mateas, and M. O. Riedl, "Learning player tailored content from observation: Platformer level generation from video traces using lstms," in *AIIDE*, 2016.

[32] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *GECCO*, 2018.

[33] A. Sarkar, Z. Yang, and S. Cooper, "Controllable level blending between games using variational autoencoders," *arXiv preprint arXiv:2002.11869*, 2020.

[34] A. M. Smith, E. Andersen, M. Mateas, and Z. Popović, "A case study of expressively constrainable level design automation tools for a puzzle game," in *FDG*. ACM, 2012.

[35] A. Anthropy and N. Clark, *A Game Design Vocabulary: Exploring the Foundational Principles Behind Good Game Design*, ser. Game Design Series. Pearson Education, 2014.

[36] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 mario ai competition," in *CEC*. IEEE, 2010.

[37] S. M. Lucas and V. Volz, "Tile pattern kl-divergence for analysing and evolving game levels," in *GECCO*, 2019.