Illuminating Super Mario Bros

Quality-Diversity Within Platformer Level Generation

Anonymised for submission

ABSTRACT

In this paper we aim to build on the current understanding of how Quality-Diversity (QD) algorithms can be applied to Procedural Content Generation (PCG) through experimentation with our platform, IlluminatedMario. The platform is capable of generating populations of playable and diverse Super Mario Bros levels and allows us to evaluate the performance impact of using alternative QD algorithms in the level search. It also incorporates the first implementation of the state-of-the-art SHINE algorithm within a PCG domain. This paper has two core aims: to evaluate the performance of SHINE within this domain and to explore the viability of IlluminatedMario's approach to PCG. To achieve these aims we run a series of experiments comparing the performance of two novel variants of SHINE with our chosen benchmark algorithm, MAP Elites. Our results show that the approach to PCG used by IlluminatedMario is a viable foundation for future PCG systems as it is consistently able to produce playable and diverse level populations. The SHINE variants implemented do not significantly outperform MAP-Elites within the current implementation. However, as the algorithm is highly configurable and under-evaluated we argue it performed well enough to warrant further investigation.

KEYWORDS

PCG, Quality-Diversity Algorithms, AI Assisted Design, Super Mario

ACM Reference Format:
Anonymised for submission. 2020. Illuminating Super Mario Bros: Quality-Diversity Within Platformer Level Generation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2020 (GECCO '20).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this paper we explore the promising intersection between Procedural Content Generation (PCG) for video-games and Quality-Diversity algorithms. PCG refers to the creation of user consumable content by software with limited or no input from a human designer[19]. QD algorithms are a recently introduced[13] subset of genetic algorithm which are designed to produce populations of high quality solutions to problems while also maximising diversity between the solutions, hence the name 'Quality-Diversity'. They have received substantial attention in recent years as they have been shown to outperform more traditional genetic search approaches in a variety of domains[14]. They have the potential to be especially relevant to PCG for video-games for several reasons[3], principally because this is a domain where diversity of output is often itself a goal, and because of the difficulty in defining objective functions (I.e defining what constitutes a 'good' video-game level). Theoretically they provide the potential power to ensure that generated content is both of a high quality, and diverse in terms of the parameters selected by the designer without any definition of a specific goal.

Researchers have experimented with using QD algorithms to generate game levels for a variety of different domains, including generating levels for "bullet-hell" games[7] and generating platformer game levels which test certain mechanics[6]. Researchers have also successfully experimented with using QD algorithms to power mixed initiative level design tools[1]. These works and others have confirmed that QD algorithms can perform well within PCG domains. There are however novel QD algorithms including SHINE[15], the focal algorithm of this paper, which have not yet been implemented and evaluated within a PCG setting.

In this paper we introduce IlluminatedMario, a platform for generating populations of diverse and playable levels for Super Mario Bros (Nintendo, 1985). The platform evolves populations of new levels from randomly initialised starting populations, using an AI game playing agent to confirm the playability of levels. The platform is capable of driving the search for new levels with several QD algorithm variants and allows useful comparison between them. Through the experiments presented here, we aim to further demonstrate the power and utility of QD based approaches to PCG. We also aim to explore the potential of the SHINE algorithm within this domain by comparing its performance across multiple level feature defined search spaces to the more tried and tested algorithm, MAP Elites[13].

This paper is presented as follows. In Section 2 we discuss the work that directly led to and inspired the work presented in this paper. In Section 3 we explain the design of IlluminatedMario, including the software used of and the motivations behind key design decisions. In Section 4 we explain the experimental protocol used to evaluate the platform and the QD algorithms implemented and in Section 5 we present the results attained. In Section 6 we discuss the gathered results and in Section 7 we conclude that the platform we present is a viable foundation for future experimental and PCG tools. We also argue that though SHINE appears to be potentially competitive in comparison to MAP Elites, further experimentation and evaluation is required to explore whether its power justifies the extra configuration it requires.

2 RELATED WORK

2.1 Quality Diversity Algorithms

QD algorithms have their roots in the work of Lehman and Stanley developing Novelty Search (NS), originally presented in their paper "Exploiting Open-Endedness to Solve Problems Through the Search for Novelty" in 2008 [9]. They found that genetic algorithms

GECCO '20, July 13–17, 2020, Prague, Czech Republic
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

designed to search for novel solutions without using an objective function ('objective free') could substantially outperform conventional objective-based search approaches. This finding led to the development of several algorithms based on NS, notably including NS with Local Competition (NS-LC)[10] which adds the capacity to promote fitness within clustered solutions, and Minimal Criteria NS (MCNS)[8] which introduces the concept of infeasibility for solutions which do not meet the minimum criteria to be considered viable. These developments give rise to the introduction of QD algorithms, which are distinct from these previous objective-free approaches in that finding novel solutions is not a means to the end of finding fit solutions, but an objective in of itself.

The power and specific benefits of QD algorithms has led to them receiving a lot of attention in recent years. They have outperformed state of the art alternatives in multiple different domains, including controlling robotic swarms[2], maze navigation [15] and controlling real-world robotic limbs [13]. The QD algorithm which has received the most attention thus far is MAP Elites, first presented in Mouret and Clune's 2015 paper "Illuminating Search Spaces by Mapping Elites"[13]. This algorithm works by building an n-dimensional grid where each dimension represents a feature that the search aims to find diversity in. As the MAP Elites search proceeds this grid is populated with the fittest solutions it finds for each location within the grid. MAP Elites is appealing, both for its performance as compared to state-of-the-art alternatives and for its comparative simplicity. To implement it a researcher only needs to decide on which features to find diversity in, what the value ranges of these features should be, and on the resolution of the map. This is power and simplicity is why it was selected as the benchmark algorithm to contrast with the other QD algorithm of interest for this paper, SHINE.

SHINE is an alternative QD algorithm, originally proposed in 2016 by Smith et al in their paper "Rapid Phenotypic Landscape Exploration Through Hierarchical Spatial Partitioning"[15]. SHINE operates similarly to MAP-Elites in that it builds a persistent archive of past solutions, except it builds a tree-structured archive rather than a map. During the search, SHINE actively changes the structure of the archive as it proceeds, building the tree deeper where it has found large numbers of solutions and prioritising search within shallower sections of the tree which are less well explored. In contrast to MAP Elites, SHINE has received little attention, having only appeared in a single paper at time of writing. It also requires significantly more tuning and configuration before being deployed and initialised, as both the maximum depth and maximum node representatives have to be decided on, as well as how levels should be prioritised when they lie in the same node. However initial results obtained have been promising. In Smith et al's original paper SHINE found solutions faster than both Novelty Search and MAP Elites when searching for correct solutions to virtual mazes.

2.2 Procedural Content Generation and Quality Diversity Algorithms

PCG has been commercially popular for video-games since the early days of the medium as it can both reduce development costs and increase the 'replayability' of games through the generation of new content. Good PCG systems have been a key selling point of large numbers of games, ranging from early classics such as 1984's Elite (Acornsoft, 1984) to modern games market dominators like Minecraft (Mojang, 2011). PCG has more recently become the focus of increasing academic interest[18], with its variety of complex challenges to tackle and strong commercial incentives to do so.

The most common use of PCG both commercially and academically in games is level generation[12]. Levels in gaming are virtual locations that have to be traversed in order to progress, and the requirements for commercial level generators are typically complex. Outputted levels need to not only satisfy the minimum requirements of the game for the levels to be viable. They also can have numerous other aesthetic or experiential requirements dependent on the genre of game and the designers intention for the content. A key requirement of automatically generated game content is often that it is sufficiently diverse[12]. If the levels outputted by a PCG system are too consistent then the game risks being tedious for the player as they will be able to quickly predict the output. This is where QD algorithms, with their focus on producing diverse output, have the potential to be very useful.

There has been a substantial amount of research into tackling PCG challenges using QD algorithms, work which this paper intends to build on. Key examples include: Liapis et al's use of the FINS algorithm to handle constraints and produce diversity in generated strategy game maps[12], and Khalifa et al who used MAP-Elites to generate Mario levels that forced the player to use certain game mechanics such as jumping to progress[6].

2.3 The Mario AI Benchmark

The Mario AI Benchmark is a popular benchmark for AI research, based on Super Mario Bros (Nintendo 1985). The benchmark is based on a public domain clone of the original game called Infinite Mario Bros (IMB) which procedurally generated its levels, developed by Markus Persson. The Mario AI Benchmark builds on IMB by adding a suite of different level generation options. The benchmark itself was originally developed in order to run an AI competition to create the highest performing agent for navigating Mario levels[16]. The benchmark has since been used for a wide variety of different research activities, including further AI competitions[5], for novel research into agent design [4] and PCG [17][6]. This makes it an appealing domain for further research due to the amount of work and software that can be redeployed.

3 SYSTEM DESIGN

In this section we discuss the design and functionality of IlluminatingMario. It is a Java-based application designed to interface with the Mario AI Benchmark. The benchmark processes encoded Mario levels and allows them to be evaluated with an AI game-playing agent. This frees the platform to focus on the storage, structure evaluation and search for new levels, as well as the evaluation of the performance of the run.

3.1 Level Encoding

In this platform we use a similar direct level encoding system to the one used in the base Mario AI benchmark. Levels are stored as nested arrays of integers, where each position in the arrays represents a single cell in the level, and the value stored represents the type of block found at that position. For example, a value of 0 represents empty space where as 1 represents a solid block in the level.

3.2 Level Structure

In the current platform implementation the only block types used are either unbreakable solid or empty space, rather than the full range of Super Mario block types that the benchmark supports. This both simplifies the experimental searches and ensures they are focused on level geometry rather than other concerns typical concerns in designing Mario levels such as enemy or power-up placement. For this paper's experiments we also enforced a fixed level width of 100 to make the outputted levels more directly comparable.

To improve the performance and output quality of our search strategies the platform enforces two constraints on the structure of outputted levels:

(1) Each level starts with a solid platform 5 blocks wide and half the height of the level. This limits the number of completely unplayable levels generated by ensuring a clear entrance to the level geometry.
(2) Every level has a solid roof which extends for the entire length of the level. This is to prevent the evaluation agent from jumping along the top of the level. The intention is that by forcing the agent through the level more interesting level structures will be evolved, especially when the system is searching for levels with a high block count.

(Please see Figure 1 for an illustration of these features)

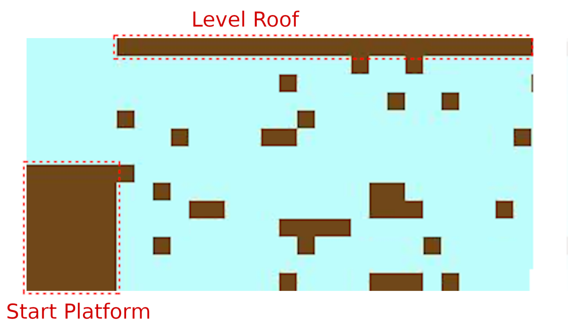


Figure 1: Typical randomly initialised level structure with start platform and roof highlighted

3.3 Level Evaluation

Levels discovered during the search are evaluated both by an AI agent and the system directly. Features that can be extracted from the level representation, such as number of blocks used, are extracted by the system. Features that can only be extracted by playing the level such as number of jumps required are extracted by the agent.

For this papers experiments we use the A* based agent developed by Robin Baumgarten. This game-playing agent was the winning entrant in the original 2009 Mario AI competition that the Mario AI Benchmark was designed for[16]. This agent often used used in research using the benchmark [6][17], as it is simple to run while also performing at a super-human level, so long as the level does not require backtracking. It looks for action sequences that will maximum its horizontal progress towards the goal over the next several frames. Using this agent lets the system confirm what portion of a generated level was navigable, and what actions were required to navigate it.

To support the experimentation presented in this paper the following level features were extracted for each generated level, though several others are supported by the platform.

Agent extracted characteristics:

- **Playability**, defined as the portion of the level that the agent was able to complete on a scale from 0.0 to 1.0, where a score of 1.0 indicates the agent reached the end of the level. This is used as our fitness heuristic.
- **Jump Entropy**, defined as the number of jump actions used by the agent as a proportion of total actions taken by the agent.
- **Speed**, defined as the average tiles per second that the agent was able to traverse during its run

Characteristics from level representation:

- **Block Count**, defined as the number of solid blocks used in the level creation
- **Contiguity Score**, defined as the number of blocks which appear directly adjacent to other blocks within the level representation

3.4 Level Initialisation and Evolution

The goal of any given run using the IlluminatedMario platform is to generate fully playable levels from a randomly initialised population. To generate this initial population the system uses a seed based random level generator. Each level is generated by selecting a pseudo-random percentage between 0% and 30%. The generator then loops through every cell with this probability of setting it to a solid block from being empty space. This generates levels that resemble random static (See Figure 2 for examples of the initial population members). This generator takes a seed as an input so that we can reproduce previous populations, letting the system use a fixed population across multiple runs and making them more directly comparable.

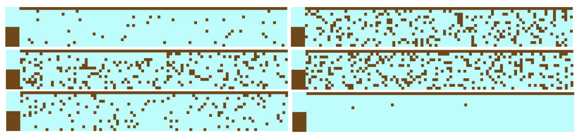


Figure 2: Sample of six levels out of the 20 used as the initial population for each experimental run

To generate offspring, individual levels are selected as parents for offspring generation by the algorithm being used for the run. The mutated offspring are then created using the following operators:

(1) **Tile-Flip Mutator**: This function loops through every cell in the level representation with a chance of flipping from

being a solid block to an empty block or visa versa. Similar tile-flip mutations are widely used in PCG level generation systems which use a tile-based level representations, including the Sentient Sketchbook [11] and the Evolutionary Dungeon Designer [1].

(2) **Column Duplication Mutator**: This function loops through every column in the level, with a chance of duplicating each column and adding it adjacent to the duplicated column. Any time this operator fires the system also removes a random column to keep the output levels at the same fixed width of 100. This operator is designed to promote the evolution of more interesting level structures by pushing offspring away from random noise patterns towards more contiguous and linear patterns which would otherwise take longer to evolve.

(3) **Two-point Crossover Operator**: This function takes in two mutated parent levels and outputs two offspring. Two points along the length of the level are randomly selected. The offspring are created by taking the portion of the level between those two points from one parent and the portion from outside of them from the other parent (Please see figure 3 for a visualisation).

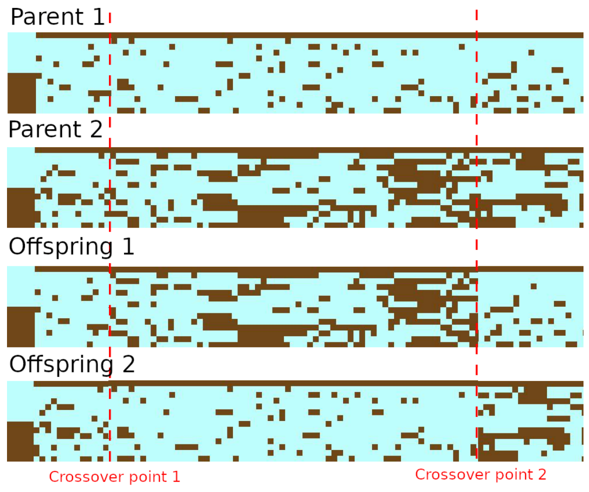


Figure 3: Visualisation of Crossover Operator

3.5 Run Initialisation

For each run using the IlluminatedMario platform the researcher needs to select the following:

(1) Which algorithm is going to drive the search. This algorithm selected drives the logic of which parent solutions are selected to create offspring from, and how previously generated offspring are stored.
(2) How many offspring are going to be generated before the run terminates
(3) Which two level features will define the search space that the run will aim to illuminate

3.6 Run Evaluation

To evaluate the outputs of the individual runs the system calculates three performance metrics as they develop over each iteration. For MAP Elites run these can be calculated directly from its archive map. For SHINE runs, the levels is extracted from the SHINE tree archive and added to a MAP Elites map to support the calculations:

- **Coverage**, defined as the proportion of cells within the search space that the run could find any level within, regardless of playability. We would expect QD algorithms like SHINE, which actively promote exploration of under explored space, would out perform algorithms like MAP Elites which explore more passively.
- **Reliability**, defined as the proportion of cells that the run populated with fully playable levels.
- **Average Fitness**, defined as the average fitness across the entire map, where an unpopulated cell has a fitness of 0. This metric is designed to provide a balance between Coverage and Reliability, in that good performance in both is required for good average fitness scores.

Additionally, the system creates heat maps at checkpoints throughout each run and at its completion so that we can visualise how the outputted levels are distributed within the specified search space. Generating these heat maps for SHINE runes also requires adding their solution archives to a MAP Elites style grid.

4 EXPERIMENTAL SET-UP

In this section we describe the system configuration which was used in this papers experiments, including the configuration of the algorithms used.

4.1 MAP-Elites Implementation

In this paper we closely follow the standard form of MAP-Elites as originally proposed by Mouret and Clune [13]. Each run incrementally populates a 2D dimensional grid representing the feature defined search space. Each cell within the grid contains the most playable level discovered so far at that location. Parent levels are selected randomly from this grid archive. The only significant alteration made in this implementation is that rather than selecting a single random member of the map for each iteration, we instead select a pair of random levels for each iteration. This is to facilitate use of the crossover operator.

Across all experiments we use a fixed grid size of 32 by 32 to match the theoretical maximum resolution reachable by the SHINE implementation. By matching the resolutions in this way we make the results out of MAP-Elites and SHINE more comparable.

4.2 SHINE Implementations

With SHINE we broadly follow the form described by Smith at al in the original paper[15]. However, in the original formulation of SHINE there was no concept of fitness as it was not relevant to the problem domain. Competition within individual nodes was instead based on the solutions distance from the centre of the parent node, referred to as Corner Distance in the original paper (See Figure 4 for a visualisation). While this competition system is effective at driving the search towards new areas of the search space, it has no way of privileging playable levels. In light of this IlluminatedMario implements two alternate variants of SHINE with different node competition systems:

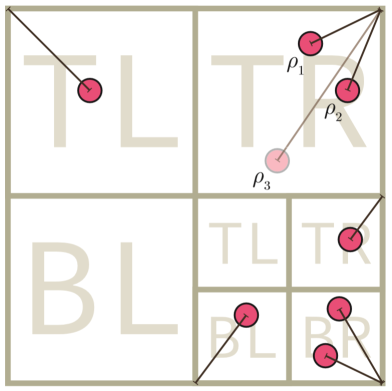


Figure 4: Visualisation of the Corner Distance metric, reproduced from [15]

(1) **SHINE-Fit**, in which parent selection within a node is based purely on fitness, similarly to MAP-Elites. This option was suggested as an alternative by Smith to make SHINE more directly comparable to other quality diversity algorithms[15].
(2) **SHINE-Hybrid**, which uses a combination of playability and corner distance with each given an equal weighting. The intention with this variation was to implement a SHINE variant capable of prioritising fitness while sacrificing less of SHINE's power to drive the search into under explored areas of the search space.

All experimental runs of SHINE use an α (the maximum depth of the tree search) value of 5 and a β (the maximum number of levels that are stored in a node) value of 3 . This α value of 5 makes the performance calculations directly comparable to the MAP-Elites runs using a 32 by 32 map. If the SHINE run finds enough solutions in an area to reach the maximum tree depth of 5 it is effectively populating a grid with the same resolution (2^α where $\alpha = 5$).

For setting the β value we ran a set of initial experiments testing values between 2 and 10, and found that a value of 3 appeared to give the best results in terms of both coverage and reliability. Setting the β higher causes the search to take increasingly long to reach greater depths within the tree, and therefore longer for the competition within nodes to be influential. Setting the value lower diminished the size of the stored archive and therefore the amount of genetic diversity within the stored population.

4.3 Experimental Search Spaces

To evaluate the system and contrast the performance of its alternative algorithms, experiments were run across all three algorithms using different pairs of level characteristics. Each pair contains one characteristic extracted directly from the level representation (Block Count, Contiguity Score) and one extracted from the agent's attempt to complete the level (Jump Entropy, Speed). This is intended to push the level search to find levels that are diverse both in their structures and in the play patterns required to complete them.

The characteristic pairs selected for experimental runs are:

- Jump Entropy vs Contiguity Score
- Jump Entropy vs Block Count
- Speed vs Contiguity Score
- Speed vs Block Count

Each characteristic has also been limited to a range of values so that the results produced by all algorithms are closely comparable. Any levels which are generated whose parameter values fall outside of these specified ranges are discarded. We recognise that this will lead to the loss of some genetic information in the form of discarded offspring. However the parameter ranges have been set include the vast majority of possible outputs from the system with the aim of mitigating against this information loss.

The ranges are as follows:

- Jump Entropy – 0.0 to 0.08
- Contiguity – 0 to 1000
- Block Count – 200 to 550
- Speed – 0.07 to 0.18

4.4 Experimental Length

For each experimental setup MAP-Elites, SHINE-Fit and SHINE-Hybrid were evaluated with 5 runs each. Each run generates 20000 offspring levels from the fixed starting population of 20. For MAP-Elites this means running for 10000 iterations, with each iteration producing two offspring. For both SHINE variants we have configured them to run in 1000 generations of size 20.

5 RESULTS

5.1 Example levels

Presented are selected examples of fully playable levels which have been generated at the extreme ends of the ranges for the behavioural characteristics. These have been selected to demonstrate the expressive range of the system.

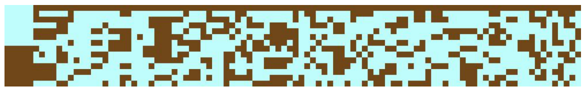


Figure 5: High Jump Entropy, High Contiguity (Jump Entropy = 0.076. Contiguity Score = 795)

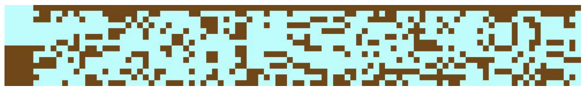


Figure 6: Low Jump Entropy, High Block Count (Jump Entropy = 0.0175. Block Count = 491)

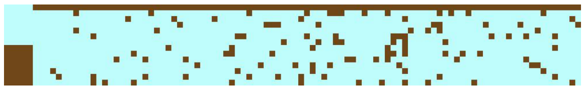


Figure 7: High Jump Entropy, Low Contiguity (Jump Entropy = 0.0788. Contiguity = 90)

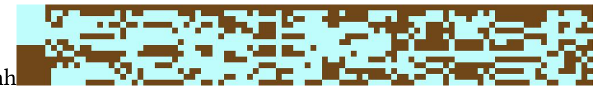
hh


Figure 8: High Speed, High Contiguity (Speed = 0.192. Contiguity Score = 938)

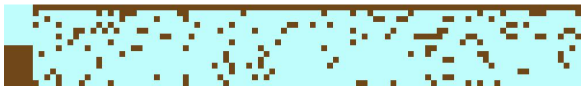


Figure 9: Low Speed, Low Block Count (Speed = 0.100. Block Count = 247)

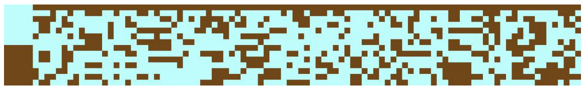


Figure 10: Low Speed, High Block Count (Speed = 0.101. Block Count = 543)

5.2 Overall Performance

The following overall coverage, reliability and average fitness results were attained, averaging the results of 5 runs for each algorithm. Each result is presented with its standard deviation across the 5 runs.

The key observations in these figures are:

- Within each feature defined space, run performance was fairly consistent across all three algorithms.
- MAP-Elites and SHINE-Fit performed very similarly across all runs.
- SHINE-Hybrid was not able to significantly outperform SHINE-Fit in terms of coverage, despite being designed to further promote it.

	MAP-Elites	SHINE-Fit	SHINE-Hybrid
Coverage	0.83±0.01	0.83±0.02	0.84±0.02
Reliability	0.63±0.03	0.63±0.01	0.55±0.04
Average Fitness	0.73±0.01	0.73±0.01	0.71±0.01

Table 1: Average results over 5 runs for Jump Entropy vs Block Count (Result ±standard deviation)

	MAP-Elites	SHINE-Fit	SHINE-Hybrid
Coverage	0.87±0.02	0.86±0.03	0.88±0.01
Reliability	0.61±0.05	0.6±0.05	0.51±0.05
Average Fitness	0.75±0.02	0.74±0.01	0.71±0.02

Table 2: Average results over 5 runs for Jump Entropy vs Contiguity Score (Result ±standard deviation)

	MAP-Elites	SHINE-Fit	SHINE-Hybrid
Coverage	0.9±0.03	0.9±0.01	0.9±0.01
Reliability	0.82±0.02	0.85±0.02	0.8±0.02
Average Fitness	0.87±0.02	0.88±0.01	0.82±0.03

Table 3: Average results over 5 runs for Speed vs Block Count (Result ±standard deviation)

	MAP-Elites	SHINE-Fit	SHINE-Hybrid
Coverage	0.88±0.04	0.9±0.05	0.92±0.04
Reliability	0.76±0.05	0.81±0.05	0.66±0.04
Average Fitness	0.83±0.04	0.87±0.05	0.81±0.04

Table 4: Average results over 5 runs for Speed vs Contiguity Score (Result ±standard deviation)

5.3 Heat Maps

In this section we show the heat maps for the run within each search space which attained the highest average fitness. Fully playable cells are coloured in green, and cells in which no level was discovered are coloured red.

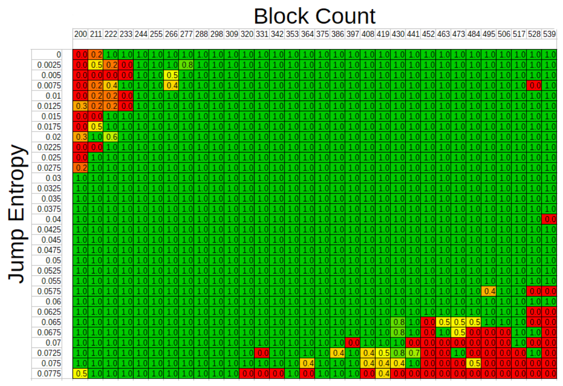


Figure 11: Jump Entropy vs Block Count best heat map (Coverage = 0.85. Reliability = 0.63. Average Fitness = 0.74)

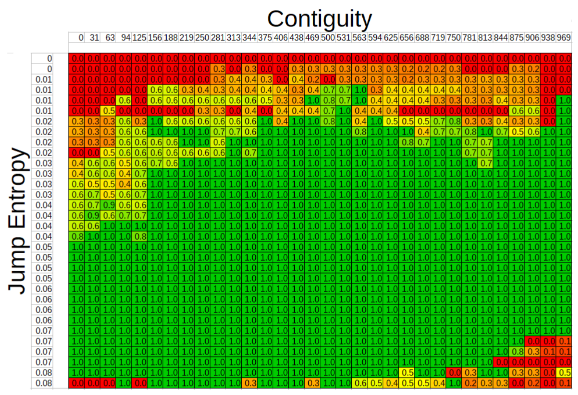


Figure 12: Jump Entropy vs Contiguity best heat map (Coverage = 0.896. Reliability = 0.669. Average Fitness = 0.780)

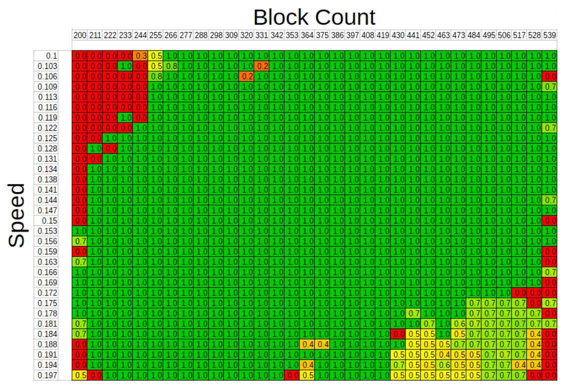


Figure 13: Speed vs Block Count best heat map (Coverage = 0.931. Reliability = 0.842. Average Fitness = 0.896)

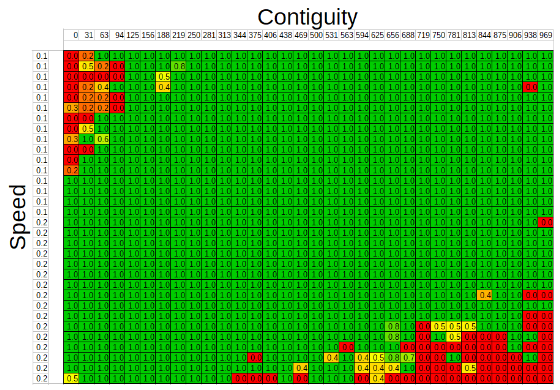


Figure 14: Speed vs Contiguity best heat map (Coverage = 0.929. Reliability = 0.891. Average Fitness = 0.909)

6 DISCUSSION

Through the experimentation explained above we have aimed to answer two broad questions:

- A) Is IlluminatingMario's PCG approach viable for generating diverse and playable levels for 2-D platforming games?
- B) How do the SHINE variants compare to MAP Elites in terms of their performance within this search space?

This section will focus on answering these two questions, and also on the further work that could most usefully be done utilising the platform we have presented in this project.

6.1 Overview of Platform Performance

From the perspective of both search space illumination and the discovery of fully playable levels, the experimental data indicates that this project's approach is viable. Regardless of which algorithm was used, every experiment run was able to populate the majority of the search space with levels, and every run was able to find substantial populations of fully playable levels. We can also see relative consistency in terms of final coverage and reliability between search spaces. These results encourage us to think that the approach could be effective in alternatively defined Mario search spaces, as well as in content search for alternative games.

From the perspective of discovering both quality and diversity the system performed well in all search spaces. Within all selected feature search spaces all three QD algorithms achieved a minimum of 0.83 coverage, 0.65 reliability and 0.71 average fitness. These results are promising as they were achieved within comparatively short run times compared to other work within search-based PCG, while also evolving from small starting populations of only 20. Each run was able to discover a minimum of 665 fully playable and feature distinctive levels ((32*32) * minimum reliability of 0.65). However, more experimentation within this domain with alternative QD algorithms and PCG approaches is required before we can know whether the quality and diversity scores attained are truly competitive.

It is also important to note that the features that defined the explored search spaces are contradictory to an extent. For example, it is much harder to design a level that does not require many jumps using low numbers of blocks than when using a large number. The contradictions are visible in the poorly explored segments of the presented heat maps. This also highlights the utility of QD algorithms in revealing how potential levels are distributed within a search space, one of the key benefits of applying QD algorithms to PCG[3].

In terms of discovering structural and experiential variety within levels outputted, the power of this project's system can also be seen within by examining the output directly (Section 5.1). The most visually interesting levels where the features are most obvious were typically found at the extremes of the feature ranges where the goals were antagonistic but not mutually exclusive. A good example of this is high contiguity and high speed (See Figure 8), which pushes the search to find levels with lots of contiguous blocks and a fast to traverse linear path through.

There were issues with discovering interesting visual variety which could be improved upon in future versions of this platform. Block Count and Contiguity proved to be too similar in their outputs, as the correlation between block count and contiguity is too strong. Additionally, the ranges for feature values (Explained in section 4.3) were set too conservatively. As we can see from the heat maps presented in section 5.3 the search was able to consistently find fully playable levels at the extreme edges of the search space. Given that the most visually distinctive levels are found at the extremes we would recommend extending these ranges in any future use of the IlluminatedMario platform. It is also important to note here that we cannot assume that visual variety or feature variety equates to meaningful diversity of experience for a player of the levels. User studies would be required to make any statements about how well the platform is able to produce levels which are experienced as being diverse and interesting by players.

6.2 Algorithm Performance

As shown in the overall performance section (Section 5.2), the SHINE-Fit variant broadly performed very similarly to MAP-Elites in terms of the quality and diversity of levels discovered. The only domain in which SHINE-Fit performed better was Speed vs Contiguity Score, but these results were still within the standard deviation of MAP-Elites results. SHINE-Hybrid performed worse than both in every domain, achieving worse reliability and average fitness scores while making no significant gains in terms of coverage.

In terms of coverage, our hypothesis before experimentation was that SHINE variants would be able to attain better scores than MAP Elites, as SHINE actively promotes the search of under-explored areas. Though SHINE under performed in this regard, with the final coverage scores being almost identical in every domain, this should not necessarily be read as a failure of SHINE. The consistency found in the final coverage scores instead suggests that the experimental runs were constantly getting close to the maximum possible coverage in the domain. For future experimentation in this field we would suggest experimenting within larger and more deceptive search spaces, as this could help to differentiate the coverage scores between MAP Elites and SHINE variants.

In terms of reliability and average fitness scores, MAP Elites and SHINE-Fit performed comparably whereas SHINE-Hybrid underperformed compared to both. The robust performance of MAP-Elites helps to vindicate the use of it as our benchmark QD algorithm, as it requires such minimal set up and is still able to discover a diversity of playable levels. Based on its performance in this domain, we suggest that MAP Elites could be powerful in a variety of PCG domains.

In the current implementation SHINE-Fit performs very similarly to MAP Elites, while still requiring more computing power and substantially more configuration before running. However, as SHINE and its variants actively prioritise searching in under-explored areas we expect them to perform better in larger and more deceptive search spaces. Based on the consistent coverage scores attained across all algorithms it is possible that the search spaces were too easy to passively explore for the active exploration by SHINE to be relevant.

7 CONCLUSION

In this paper we have presented our system for generating large and diverse Mario levels using two state of the art illumination algorithms, MAP-Elites and SHINE. Through our experimentation with the system we aimed to evaluate both whether the approach to level generation was viable, and the relative performance of these two algorithms within this domain.

Based on the diverse and high performance level populations that the system was able to generate, we are confident concluding that the approach is viable. We suggest that a system based on the approach presented here could be optimised to produce diverse level populations for any game in which level encoding permits the creation of offspring, and where there is an AI agent available to evaluate the playability and features of generated levels.

In the comparison between the algorithms, MAP-Elites performed similarly to SHINE-Fit in every domain in terms of output reliability, highlighting MAP-Elites strengths as a benchmark within the area of QD algorithm research. While the SHINE variants underperformed against our expectations, SHINE-Fit still performed robustly when compared to MAP-Elites. This combined with how under evaluated SHINE is compared to MAP Elites, and how highly configurable it is should prompt further investigation. We suggest that future research into SHINE should further explore both how SHINE's performance compares to alternative QD algorithms in large and deceptive PCG search spaces, and whether an alternative inter-node competition system could improve its performance.

REFERENCES

[1] Alberto Alvarez, Steve Dahlskog, Jose Font, and Julian Togelius. [n.d.]. Empowering Quality Diversity in Dungeon Design with Interactive Constrained MAP-Elites. ([n. d.]). arXiv:cs.AI/http://arxiv.org/abs/1906.05175v1
[2] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. [n.d.]. Evolution of Swarm Robotics Systems with Novelty Search. ([n. d.]). https://doi.org/10.1007/s11721-013-0081-z arXiv:cs.NE/http://arxiv.org/abs/1304.3362v1
[3] Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. [n.d.]. Procedural Content Generation through Quality Diversity. ([n. d.]). https://doi.org/10.1109/MCI.2018.2871703 arXiv:cs.NE/http://arxiv.org/abs/1907.04053v1
[4] Handa Hisashi. 2014. Deep Boltzmann Machine for evolutionary agents of Mario AI. In *2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. https://doi.org/10.1109/cec.2014.6900625
[5] Sergey Karakovskiy and Julian Togelius. 2012. The Mario AI Benchmark and Competitions. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (mar 2012), 55–67. https://doi.org/10.1109/tciaig.2012.2188528
[6] Ahmed Khalifa, Michael Cerny Green, Gabriella Barros, and Julian Togelius. 2019. Intentional computational level design. In *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO 19*. ACM Press. https://doi.org/10.1145/3321707.3321849
[7] Ahmed Khalifa, Scott Lee, Andy Nealen, and Julian Togelius. [n.d.]. Talakat: Bullet Hell Generation through Constrained Map-Elites. ([n. d.]). arXiv:cs.AI/http://arxiv.org/abs/1806.04718v2
[8] Joel Lehman and K.O. Stanley. 2010. Revising the evolutionary computation abstraction: Minimal criteria novelty search. (01 2010), 103–110.
[9] Joel Lehman and Kenneth O. Stanley. 2008. Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. In *ALIFE*.
[10] Joel Lehman and Kenneth O. Stanley. 2011. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM Press. https://doi.org/10.1145/2001576.2001606
[11] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Sentient sketchbook: Computer-aided game level authoring. In *In Proceedings of ACM Conference on Foundations of Digital Games, 2013. In Print.*
[12] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2015. Constrained Novelty Search: A Study on Game Content Generation. *Evolutionary Computation* 23, 1 (mar 2015), 101–129. https://doi.org/10.1162/evco_a_00123
[13] Jean-Baptiste Mouret and Jeff Clune. [n.d.]. Illuminating search spaces by mapping elites. ([n. d.]). arXiv:cs.AI/http://arxiv.org/abs/1504.04909v1
[14] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. 2016. Quality Diversity: A New Frontier for Evolutionary Computation. *Frontiers in Robotics and AI* 3 (jul 2016). https://doi.org/10.3389/frobt.2016.00040
[15] Davy Smith, Laurissa Tokarchuk, and Geraint Wiggins. 2016. Rapid Phenotypic Landscape Exploration Through Hierarchical Spatial Partitioning. In *Parallel Problem Solving from Nature – PPSN XIV*. Springer International Publishing, 911–920. https://doi.org/10.1007/978-3-319-45823-6_85

[16] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. 2010. The 2009 Mario AI Competition. In *IEEE Congress on Evolutionary Computation*. IEEE. https://doi.org/10.1109/cec.2010.5586133

[17] 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 *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM Press. https://doi.org/10.1145/3205455.3205517

[18] Geogios N. Yannakakis. 2012. Game AI revisited. In *Proceedings of the 9th conference on Computing Frontiers - CF '12*. ACM Press. https://doi.org/10.1145/2212908.2212954

[19] Georgios N. Yannakakis and Julian Togelius. 2018. *Artificial Intelligence and Games*. Springer. https://www.amazon.com/Artificial-Intelligence-Games-Georgios-Yannakakis-ebook/dp/B079WGWJB9?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B079WGWJB9