
Illuminating Zelda Rooms in the Latent Space of a Deep Convolutional Generative Adversarial Network

Reilly Goddard

Abstract

Generative Adversarial Networks (GANs) are now powerful and popular tools for the task of content generation, but are not free from downsides. Generated content may be missing certain important qualities and can be lacking in diversity when training data is scarce. When paired with quality diversity techniques such as Covariance Matrix Adaptation MAP-Elites (CMA-ME), generative adversarial networks capably generate synthetic content that is not only high-quality but also diverse. Here, we apply this technique to the generation of playable levels for *The Legend of Zelda* by exploring the latent space of the generative network in terms of the level characteristics produced. We compare the resulting rooms generated to those generated via other methods in terms of diversity of environment. We also use a Zelda-playing agent to optimise for completability of rooms generated and compare the generation methods further. We find that the MAP-Elites algorithm represents a significant improvement in terms of both completability and diversity of rooms generated over a random search of the GAN latent space. We also find that CMA-ME in turn represents a significant improvement over MAP-Elites in terms of diversity without compromising on completability.

1. Introduction

Generative adversarial networks (GANs) (Goodfellow et al., 2014) represent a versatile tool for computer generation of content, and have been applied to many areas, such as the ones mentioned in (Yinka-Banjo & Ugot, 2020; Antipov et al., 2017; Paganini et al., 2018; Lan et al., 2020). However, while GANs can often learn to produce high-standard imitations of given content, they don't necessarily generalise to a diverse distribution of these imitations when given limited datasets. Nor do they necessarily produce content with particular desirable qualities which may be no trouble for alternative, more heavily scripted content generation algorithms. Previous attempts have been made to 'steer' GANs in a particular direction with respect to certain qualities of the generated output - such as rotation and scale of objects in generated pictures (Jahanian et al., 2019) - by manipulating the GAN's latent space. While impressive, these approaches are limited and require training of an additional

algorithm.

Quality diversity algorithms (QD) provide a promising path around the diversity problem. They seek to ensure, for a given search space, that not only are returned solutions high quality with respect to a given objective function, but that a diverse range of such solutions is returned. One such approach to QD is illumination: choosing meaningful dimensions along which to stratify the search space, performing a genetic-algorithm-style search, and recording the individuals from each strata with the highest score on the objective (Mouret & Clune, 2015). This can be applied to the latent space of a GAN in order to identify latent vectors that produce certain qualities in the generated output (Fontaine et al., 2020).

Video games provide a fantastic context for testing algorithms in general due to their often predictable dynamics, low-noise outputs and intuitive digital representations. One particular area of interest in video game AI is procedural content generation (PCG) - many games involve partially or fully computer-generated environments/characters. However, PCG for video game levels can be a balancing act between ensuring playability and giving the algorithm creative freedom.

Here, we present an illuminated GAN as a method for generating *The Legend of Zelda* (1986) levels. Using a GAN rather than an algorithm with a hard-coded completability constraint, as well as applying illumination should guarantee a diverse range of generated levels that make use of a larger share of the search space - usually a desirable quality in video game levels and much other content. Our method follows previous work which took a similar approach to *Super Mario Bros* (1985) (Fontaine et al., 2020). We seek in particular to generate levels which pose diverse challenges to the player, for example varying the enemies fought and the length of time required to progress during each level. We will also evaluate these generated levels in terms of their completability using an A* agent.

This paper is structured as follows: Section 2 describes the dataset we use and how we evaluate it; Section 3 concerns the models and algorithms used; Section 4 details the experiments we carried out; Section 5 discussed related research; and Section 6 evaluates our work and discusses future work on this topic.











Tiles	Encoded Tiles (Characters)	Encoded Tiles (Numbers)	Graphic
Void	-	0	
Wall	W	1	
Floor	F	2	
Block	B	3	
Monster	M	4	
Element (Lava, Water)	P	5	
Element + Floor	O	6	
Element + Block	I	7	
Door	D	8	
Stair	S	9	

Table 1. Encoding details of the tiles in Zelda excluding the 4 unknown tiles

2. Data Set and Task

2.1. Data

The Legend of Zelda as a game is made up of an overworld and levels. The levels are subdivided into rooms, each taking the space of a single screen, with only one room shown to the player at a time. A room may contain anything from Table 1, but must contain at least a single entry or exit point. The game contains 9 individual levels with a total of 237 screens.

To access the individual rooms, data was pulled from the The Video Game Level Corpus (VGLC) (Schrum et al., 2020). In this repository, each room in game's levels has been saved as an txt file with tiles encoded into different characters in a 11 x 16 grid. Before the data is input into the GAN, the txt file is converted into a 2D array and characters are encoded once again into numbers. It is then padded with void tiles to be square and turned into a 3D array of one-hot encodings of shape 14 x 16 x 16. Some rooms contain tiles not described in the VGLC documentation and which are not present in the full levels. In practice these were rare enough to be ignored and we theorise that they correspond to the starting locations of Link and bosses.

2.2. Evaluating Rooms

With the rooms used as training data, our goal is to train a generator that is able to generate Zelda room that are indistinguishable from the rooms given as training data. After we have a generator that is capable of the above task, we will apply the illumination methods MAP-Elites and CMA-ME to the generator, in order to obtain diverse sets of rooms.

To evaluate the rooms generated, two metrics were used.

First is to see if the room is completable. To do this, we use an run basic checks such as ensuring there are at least two doors in the room. Then an A* agent is used to find a path between the doors if possible, possibly defeating monsters on the way. The second metric is the average L2 similarity between the rooms generated and all the training rooms in their 3D one-hot array form.

$$L_2(X, Y) = \sqrt{\sum_{i=1}^n (X_i - Y_i)^2}$$

X_i and Y_i are components of vector X and Y respectively

2.3. Evaluating Illumination Methods

An illumination method is successful if it produces diverse and high quality outputs. Thus we evaluate MAP-Elites and CMA-ME by: the number of rooms with unique sets of characteristics they generate; the percentage of these rooms that are completed by our agent; and the percentage of the possible space of certain characteristics that they utilise. These metrics will be discussed in further detail in Section 4.2, as it is beneficial to have read section 3 beforehand.

3. Methodology

3.1. Generative Adversarial Network

Generative Adversarial Networks were introduced by (Goodfellow et al., 2014). A generator network G, and a discriminator D will be trained at the same time. Generator G takes in a random vector and will generate a room from that vector. Discriminator D will take rooms from the original game and from G, and will try to distinguish which are real and which are fake. These two models will essentially compete against one another, the discriminator D aims to minimize the probability of miss judgement, while generator G aims to maximize that probability. The generator G's training is complete when it can steadily generates rooms that causes D to have the same accuracy as a random classifier.

3.2. GAN Models

Deep Convolutional Neural Networks are used to train both the generator and discriminator, the architecture of the network is shown in figure 1. For the generator it receives a latent vector of size 16 and will pass it through the convolution layers until a 10 x 16 x 16 matrix is generated. The discriminator receives a 10 x 16 x 16 matrix and will pass it through the convolution layers producing a vector of size 1, which indicates whether the room is from the original game or not. The model uses strided convolutions in the discriminator and fractional-strided convolutions in the generator. Batchnorm are used in both the models after every layer. ReLU activation are used in all layers of the generator where LeakyReLU activation are used in the discriminator.

After training the GAN, the generator would be able to

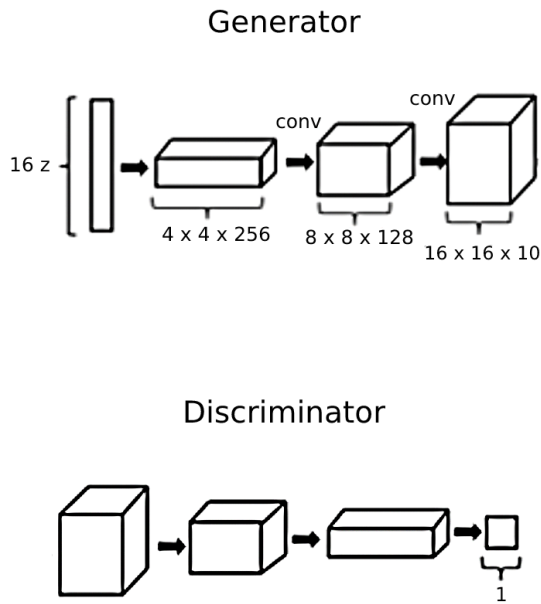


Figure 1. DCGAN architecture

generate a $10 \times 16 \times 16$ matrices, and we would combine the 10 channels back into one channel resulting with a 16×16 matrix, then we would crop away the bottom 5 rows of the room resulting with a 11×16 room.

3.3. Quality Diversity Algorithms

Quality diversity algorithms - presented by (Gravina et al., 2019) - are a set of algorithms which seek to generate a diverse set of high quality solutions. In terms of Zelda room generation, the goal of a QD algorithm is to search for rooms that the GAN generates.

The generator of the GAN generates each room from a latent vector it takes as input in a deterministic but not human-predictable manner. Using a QD algorithm allows us to search the latent vector space for specific features in the output space.

3.4. Illumination

The Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) algorithm (Mouret & Clune, 2015) is an example of QD. First, the user chooses several important characteristics which they seek variety within - these are called behavioural characteristics (BCs). Illumination methods are QDs which use BCs, such as MAP-Elites. Together the BCs form a behavior space and this space is tessellated into hypervolumes. An evolutionary algorithm is used to generate a population of individual samples via mutation operations from an initial random population. We keep track

of which individual has the highest fitness or performance within individuals that fall into the same BC hypervolume and call this individual the elite. Once a sufficient number of these hypervolumes are occupied by an elite, or once another termination condition is met, MAP-Elites returns the elite individuals.

Algorithm 1 MAP-Elites

```

X = {}, P = {}
for i = 1 to max_iter do
  if i < population_size then
    x = get_random_solution()
  else
    x' = random_choice(X)
    x = variation(x')
  end if
  b = feature_descriptor(x)
  p = performance(x)
  if b not in X or p > P[b] then
    X[b] = x
    P[b] = p
  end if
end for
return X, P

```

Covariance Matrix Adaptation MAP-Elites (CMA-ME) is an improvement upon MAP-Elites, which uses Covariance Matrix Adaptation Evolution Strategy (CMA-ES), an evolutionary algorithm for optimising a search. CMA-ME maintains different so-called emitters, which are analogous to CMA-ES instances. The emitters will rank solutions higher if they fill previously unfilled cells, other solutions are ranked on their improvement on the fitness of the current cells. Thus CMA-ME is driven towards engaging in greater exploration of the behaviour space than MAP-Elites.

3.5. The Agent

Originally we planned on using an agent trained using deep reinforcement learning (Torrado et al., 2018) on the General Video Game AI (GVGAI) framework. Unfortunately we ran in to compatibility issues with various required packages and so instead we built an A*.

We will briefly explain how we apply the A* search algorithm to our problem. This search algorithm treats every tile in the room as a node, with adjacent tiles treated as child nodes. Only reachable tiles will be considered - i.e. floor tiles and the monster tiles (as you can defeat the monster and walk through it). Next, the agent will pick the child node with the lowest cost, which is estimated by $f(n) = g(n) + h(n)$, where $f(n)$ is the cost, $g(n)$ is the cost from current node to this child node and $h(n)$ is a heuristic function that estimates the cost of the path from the child node to the goal. The above process is repeated until the goal is reached.

A major advantage of using the A* agent is that it is always guaranteed to find a path where possible and will return the shortest path available. The downside is that the agent

we implemented is only a path finding agent instead of an agent that actually understand and plays the game like the GVGA agent mentioned above. Because of this our agent is limited to collecting simple data for evaluations such as the number of enemies beaten and the number of steps it took to clear the room.

4. Experiments

This project is available on Github. ¹

4.1. Training

As discussed previously, our neural net design followed that of (Fontaine et al., 2020) and (Volz et al., 2018) and as such we used similar hyperparameters for training. In particular we used a learning rate of 5×10^{-5} for both the generator and discriminator networks; a batch size of 32; RMSProp as our optimisation method; and trained for 5000 epochs. Our latent space was only 16-dimensional, compared to (Fontaine et al., 2020)'s 32 dimensions. This is because Zelda rooms contain fewer distinct tile types and each room is smaller than a Mario level section generated by their network.

We trained using these setting with three different seeds and found the results to be easily satisfactory enough - in terms of completability and L2 distance of rooms - to apply illumination.

4.2. Behavioral Characteristics and Coverage

We chose the room properties as detailed in 2 as our behavioural characteristics for illumination. 'Steps' and 'Kills' are obvious metrics for determining how a room feels to play and ones we would like to see significant variation in; if these metrics are low, the room will likely be too simple to be entertaining; too high and the room may be too difficult, especially for new players. These metrics are measured by our agent and do not cover every possible solution to each room, only a relatively short one. Through experimentation on generated rooms it was determined that the absolute number of steps to complete a room never exceeded 40. The same was true of the original training rooms. We discretised this range into 9 buckets: if a room was uncompletable its 'Steps' characteristic was 0, otherwise its 'Steps' characteristic was given by $\text{ceiling}(\text{number of steps to complete room}/5)$. This approach makes more sense than using the exact number of steps to complete the room because the exact number is not very important to the room design; it is, however, typical for games to divide their levels into difficulty brackets

'Enemies' and 'Blocks' and 'Water' are simple characteristics which describe the makeup of a room. We observed that the natural range of number of enemies in generated rooms did not exceed 8. For the other two characteristics we used a similar discretisation approach as for 'Steps'.

We calculated the mean L2 distance between the training rooms to be 6.81. Our 'L2' characteristic therefore tells us whether the generated room is a greater or lesser L2 distance from the training rooms' average than they were.

To measure the exploration of the algorithms examined, we use a metric which we call 'W/B/E Coverage'. This metric attempts to estimate the percentage of the possible rooms - described in terms of Water, Blocks and Enemies - which the algorithm generated. Due to our discretisation of the BCs, the number of possible rooms is extremely difficult to accurately calculate given how the number of blocks, water tiles and enemies are related to each other. Therefore we estimate the number of possible BC combinations by multiplying the number of values explored across all generated and training rooms for each BC, yielding 2106. We note that this number includes impossible rooms - i.e. ones containing more than 84 tiles - and excludes possible rooms (although most of these would be uncompletable). For reference, 24 different W/B/C combinations are present in the training rooms from the original game which would give a W/B/C coverage of 1.14%. There are only 237 such rooms however, and as such this percentage should not be compared directly to those obtained during our experiments which generated far more rooms. Steps and Kills are unused for this metric as, by design, our agent finds the simplest path through each room and as such provides little information about the range of total possibilities of room architecture.

4.3. Random Generation

As a baseline comparison we performed three runs of generating 10000 samples without any kind of illumination method. The latent vectors for these samples were generated from a Gaussian distribution with mean 0 and variance 1. These samples were then classified by their behavioural characteristics so as to have a set with unique characteristics comparable to the set of elites generated by the other two methods.

4.4. Map-Elites Performance

We ran the standard MAP-Elites algorithm using the behavioural characteristics and model described above. The algorithm was run until 10000 individuals had been generated and evaluated. This experiment was repeated three times with different initial populations to reduce the risk of anomalous behaviour. The results are shown in table 3.

Compared to the random baseline, MAP-Elites generated significantly more rooms with unique combinations of the BCs (705 vs. 988) and a higher percentage of possible rooms were explored (7.68% vs 8.51%). Furthermore, a higher percentage of the rooms generated were completable (71.12% vs. 77.69%). This higher completion rate is to be expected as the latent vectors for MAP-Elites are created from previously generated elites which are themselves more likely to be completable than the average randomly generated room as completability is used as the fitness met-

¹<https://github.com/1645474/ZeldaGAN-LSI>

Name	Type	Range	Description
Steps	Discretised	0-8	a measure of how many tiles the agent had to traverse to complete the room, or 0 if it couldn't
Kills	Literal	0-8	how many enemies the agent killed while completing the room
Enemies	Literal	0-8	total number of enemies present in the room
Blocks	Discretised	0-8	a measure of how many block tiles were present in the room
Water	Discretised	0-8	a measure of how many water tiles were present in the room
L2	Binary	0-1	whether the mean L2 distance from the training rooms is greater than 8.61

Table 2. Behavioural Characteristics for the MAP-Elite Algorithm

ric.

4.5. CMA-ME Performance

We ran CMA-ME using the behavioural characteristics and model described above. similarly to MAP-Elites, the algorithm was run until 10000 individuals had been generated and evaluated with this process being repeated three times. The results are shown in table 3.

CMA-ME achieved better results across the board than both the baseline and MAP-Elites: all of the best values obtained occurred when on CMA-ME runs and the CMA-ME mean values were superior to the mean values for the other methods. The gains made by CMA-ME over MAP-Elites for the number of elites generated and W/B/E coverage are comparable to those of MAP-Elites over the random baseline - i.e. CMA-ME is a very significant improvement. This indicates that the measures employed by CMA-ME based on CMA-ES to boost exploration are working effectively and the fact we also saw an increase in completability suggests this exploration didn't come at the cost of another desirable trait.

4.6. Generated Rooms

As well as any effects on exploration and completability, another advantage of using an elites-based method is that we are provided with a list of elite rooms broken down by behaviour characteristics; thus we can chose any type of room that takes the player's fancy by its properties. For example, we could chose a room with the highest Water characteristic of any generated room but no blocks or enemies 2. Rooms with more extreme BC values tend to have higher L2 values and look less like human-designed rooms. However, high L2 rooms, while less conventional, are are often still completable - 79.92% of the time for CMA-ME for instance. For example, the room shown in 3, which had the largest Steps characteristic and even contains a void tile in the middle of the room.

Full levels in the original game often contain more complex objectives, such as obtaining a key in one room to unlock a locked door in another. Some generated rooms appear to be far too easy, but may be based on patterns present in rooms which would normally contain such additional objectives. For example, the addition of a key to the top-right area of 4 would make for a room which forces the player to walk past enemies which then corner them between blocks and walls.

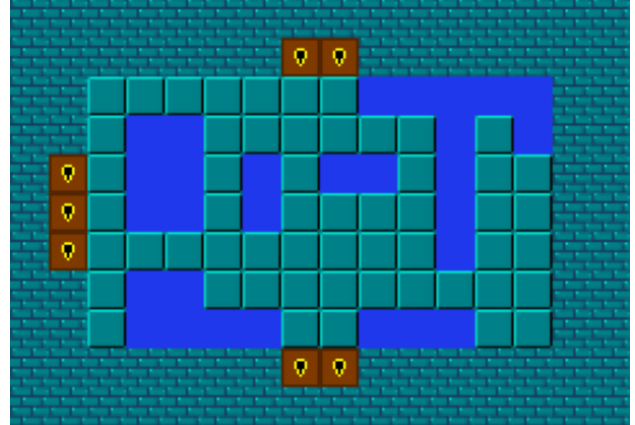


Figure 2. (MAP-Elites generated) A completable elite with zero enemies or blocks and a water level of seven.

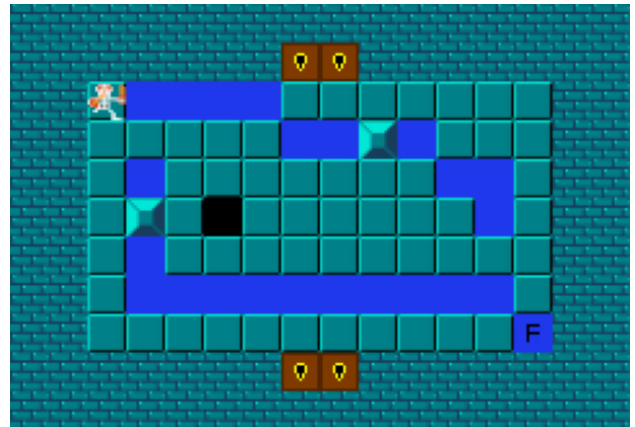


Figure 3. (MAP-Elites generated) A high L2 room with Step = 8.

Where generated rooms fail to be completable they are often not far off. A common problem is the presence of only one door for example. Many such problems could be easily fixed by a hard-coded program after generation if these level-generation techniques where to be used in a practical application. For example a script which detected when a block tile was obstructing a door tile and removed it would turn 5 into a challenging but legitimate room.

5. Related Work

The key concept for our experiment, illumination, was first proposed by (Mouret & Clune, 2015). They used illumination to to perform searches in three different spaces to design: neural network, simulated soft robot morpho-

Method	Random				MAP-Elites				CMA-ME			
	1	2	3	Mean	1	2	3	Mean	1	2	3	Mean
Elites Generated	714	712	691	705.67	951	960	1053	988	1213	1044	1108	1121.67
Completable Elites	512	508	486	502	736	736	832	768	946	808	832	862
Completeness (%)	71.67	71.35	70.33	71.12	77.39	76.67	79.01	77.69	77.99	77.39	79.51	78.30
W/B/E Coverage (%)	7.69	7.74	7.60	7.68	8.45	8.31	8.78	8.51	9.97	9.35	9.16	9.50

Table 3. Results of the three elite generation methods. All figures rounded (if applicable) to two decimal places.

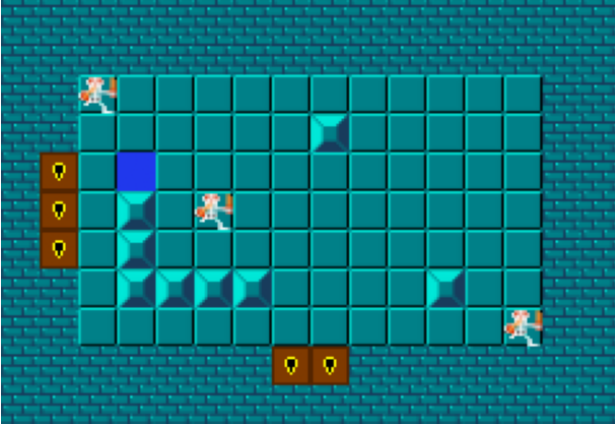


Figure 4. (CMA-ME generated).

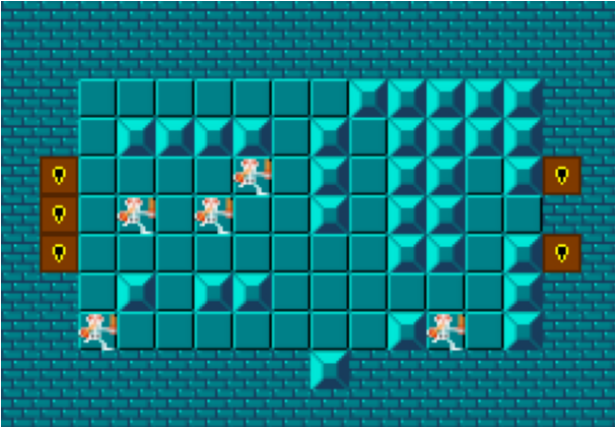


Figure 5. (CMA-ME generated) A slightly invalid room.

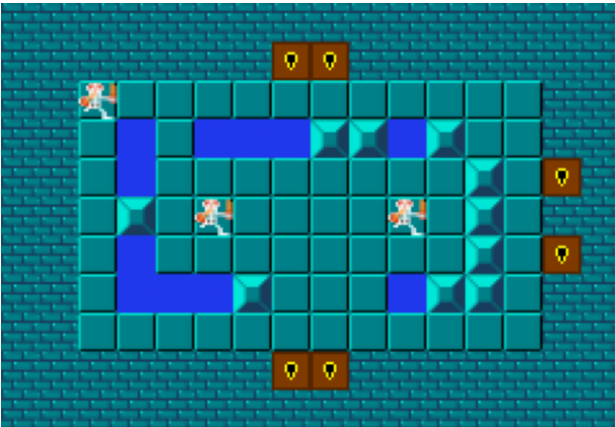


Figure 6. (CMA-ME generated) A room which uses several of each B/W/E tile type.

gies, and a real, soft robotic arm. In their experiments, they proved that the MAP-Elites illumination search outperforms other search algorithms by a huge margin in different aspects such as reliability, precision and coverage.

GAN were first used by (Goodfellow et al., 2014). In their experiments, they trained their GAN with some common training datasets such as MNIST, CIFAR-10 and TDF. Comparing the results of their GAN with some other generator like DMN, Stacked CAE, Deep GSN, they show that GANs are able to generate better images. GANs quickly became popular for generating contents and thus different mutations of GAN were also developed including DCGAN (Radford et al., 2016) which we used for our network.

Our experiment is based on (Fontaine et al., 2020), who successfully applied illumination methods to a Mario DCGAN. Applying illumination to Mario achieved a great success in generating a diversity of levels each with their own features. In Fontaine's work, not only did they use illumination to generate levels, they also used it to filter out levels that were not optimised, leaving a generator that only generated quality levels.

6. Conclusions

6.1. Evaluation

In this paper we have shown that the generation of Zelda rooms to be a fruitful application of generative adversarial networks and illumination methods to work effectively in increasing the quality and diversity of the content generated by such GANs. In particular, we evaluated the performance of the MAP-Elites and CMA-ME algorithms in terms of their ability to generate high-quality rooms - measured by completeness - and diverse rooms - measured by the number of uniquely characterised elites generated and W/B/E coverage. We found MAP-Elites to be a significant improvement over random room generation in terms of completeness and diversity and the CMA-ME algorithm to be a significant improvement upon that in terms of diversity. This demonstrates that the problem of a small training dataset can be somewhat overcome by the additional degree of exploration of the behaviour space forced by illumination. Specifically with respect to procedural content generation for video games, we have reinforced the idea put forward in similar papers: that illuminated GANs can provide new levels on-demand for players without being too predictable and while giving them control over level characteristics.

6.2. Future Research

The results of this paper, while promising, still leave some areas and applications unexplored. It demonstrates that an illuminated GAN can successfully take a small initial data set and expand on it. Within the context of Zelda, only 237 individual screens were available for training, with each screen representing a single room. However, each generated room was treated as an individual piece, independent from any larger level. This raises a few queries, namely how the generated rooms would act together within the context of a level, and if this method of generation would be applicable to data sets outside of video games.

The Legend of Zelda functions differently from the Mario game originally used with the method implemented in (Fontaine et al., 2020). It contains a world map that links smaller dungeons, or rooms together. While this may appear similar to the Mario game at first glance, it is functionally far from the same. Mario's levels involve continuous, left-to-right scrolling and are functionally independent from one another with a clear start and finish for each level. Contrasting this, the levels in Zelda are made up of individual screens transitioned between via doorways, creating a sprawling grid of interconnected screens. In addition to the more complicated connectivity of the levels, the player has a persistent inventory including items such as bombs and keys that may be needed to progress.

Future iterations of this project could aim to create multi-room levels with a properly connected grid. There are two methods that come to mind when trying to accomplish this. First, the current rooms could be grouped based on the location of the doors, algorithmically turning a set of rooms into a grid with a start room and end door. Then any remaining doors on the bordering rooms without a connection could be replaced with walls to limit the size of the level. Techniques such as horizontal or vertical mirroring could also be used to match rooms for level generation.

The second method would involve generating a graph grammar which describes the architecture of the grid and relations of the items within rooms, such as that used in (Gutierrez & Schrum, 2020). This method would fit well with an elites-based rooms generating method, as the graph grammar specifies not only the locations but the key properties of each room which could then be searched for in a set of elites whose behavioural characteristics are chosen to match such potential key properties.

Both methods still greatly benefit from the use of illumination, due to the limited data set and diversity, but either of these methods would provide a more usable output for the end user. It would allow for the creation of fully playable levels instead of a single room but present new challenges in terms of agent AI.

Our results, as well as those such as (Fontaine et al., 2020), (Steckel & Schrum, 2021), (Sarkar & Cooper, 2021), show the techniques we used and similar techniques to be effective at level generation for 2D, grid-based video games in

general. These methods could obviously be applied to more games, hopefully including more recent and complicated ones.

However, these methods can in theory be applied to any generation task which produces readily classifiable features over which variation is desirable and there exists a clear objective to optimise for other than just similarity to the distribution of the training data.

References

- Antipov, G., Baccouche, M., and Dugelay, J. Face aging with conditional generative adversarial networks. In *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 2089–2093, 2017. doi: 10.1109/ICIP.2017.8296650.
- Fontaine, Matthew C., Liu, Ruilin, Khalifa, Ahmed, Torgelius, Julian, Hoover, Amy K., and Nikolaidis, Stefanos. Illuminating mario scenes in the latent space of a generative adversarial network. 7 2020. URL <http://arxiv.org/abs/2007.05674>.
- Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial networks. 6 2014. URL <http://arxiv.org/abs/1406.2661>.
- Gravina, Daniele, Khalifa, Ahmed, Liapis, Antonios, Torgelius, Julian, and Yannakakis, Georgios N. Procedural content generation through quality diversity. *CoRR*, abs/1907.04053, 2019. URL <http://arxiv.org/abs/1907.04053>.
- Gutierrez, Jake and Schrum, Jacob. Generative adversarial network rooms in generative graph grammar dungeons for the legend of zelda, 2020.
- Jahanian, Ali, Chai, Lucy, and Isola, Phillip. On the "steerability" of generative adversarial networks. *CoRR*, abs/1907.07171, 2019. URL <http://arxiv.org/abs/1907.07171>.
- Lan, Lan, You, Lei, Zhang, Zeyang, Fan, Zhiwei, Zhao, Weiling, Zeng, Nianyin, Chen, Yidong, and Zhou, Xiaobo. Generative adversarial networks and its applications in biomedical informatics. *Frontiers in Public Health*, 8:164, 2020. ISSN 2296-2565. doi: 10.3389/fpubh.2020.00164. URL <https://www.frontiersin.org/article/10.3389/fpubh.2020.00164>.
- Mouret, Jean-Baptiste and Clune, Jeff. Illuminating search spaces by mapping elites. *CoRR*, abs/1504.04909, 2015. URL <http://arxiv.org/abs/1504.04909>.
- Paganini, Michela, de Oliveira, Luke, and Nachman, Benjamin. Accelerating science with generative adversarial networks: An application to 3d particle showers in multilayer calorimeters. *Phys. Rev. Lett.*, 120:042003, Jan 2018. doi: 10.1103/PhysRevLett.120.042003. URL <https://link.aps.org/doi/10.1103/PhysRevLett.120.042003>.

- Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.
- Sarkar, Anurag and Cooper, Seth. Generating and blending game levels via quality-diversity in the latent space of a variational autoencoder, 2021.
- Schrum, Jacob, Gutierrez, Jake, Volz, Vanessa, Liu, Jialin, Lucas, Simon, and Risi, Sebastian. Interactive evolution and exploration within latent level-design space of generative adversarial networks, 2020.
- Steckel, Kirby and Schrum, Jacob. Illuminating the space of beatable lode runner levels produced by various generative adversarial networks, 2021.
- Torrado, Ruben Rodriguez, Bontrager, Philip, Togelius, Julian, Liu, Jialin, and Perez-Liebana, Diego. Deep reinforcement learning for general video game ai. In *Computational Intelligence and Games (CIG), 2018 IEEE Conference on*. IEEE, 2018.
- Volz, Vanessa, Schrum, Jacob, Liu, Jialin, Lucas, Simon M., Smith, Adam M., and Risi, Sebastian. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. *CoRR*, abs/1805.00728, 2018. URL <http://arxiv.org/abs/1805.00728>.
- Yinka-Banjo, Chika and Ugot, Ogban Asuquo. A review of generative adversarial networks and its application in cybersecurity. *Artificial Intelligence Review*, 53:1721–1736, 3 2020. ISSN 15737462. doi: 10.1007/s10462-019-09717-4.