Interactive Latent Variable Evolution for the Generation of Minecraft Structures

Tim Merino
tm3477@nyu.edu
New York University Tandon
Brooklyn, New York, USA

M Charity
mlc761@nyu.edu
New York University
Brooklyn, New York, USA

Julian Togelius
julian@togelius.com
New York University Tandon
Brooklyn, New York, USA

ABSTRACT

The open-world sandbox game Minecraft is well-known for applying a wide array of procedural content generation techniques to create unique and expansive game environments. However, procedurally generated buildings are absent in the Minecraft world, thus players must build their own structures to flesh out their worlds. This build process can be extremely time-consuming and appeals to more creatively-inclined players. To aid players in this process, we introduce a tool combining interactive evolution with latent variable evolution to evolve procedurally generated Minecraft structures to a player's aesthetic choices. We employ two separate neural network models to generate structures: a 3D generative model for generating the structure design and an encoding model for applying Minecraft textures to the structure's voxels. We evaluate this tool with a user study incorporating an online interface that allows participants to select, evolve, and guide a population of these generated 3D structures towards a specific design goal.

ACM Reference Format:
Tim Merino, M Charity, and Julian Togelius. 2023. Interactive Latent Variable Evolution for the Generation of Minecraft Structures. In *Foundations of Digital Games 2023 (FDG 2023), April 12–14, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3582437.3587208

Figure 1: Prebuilt structures in the game Minecraft. These buildings are used in procedurally generated NPC villages.

1 INTRODUCTION

Minecraft is an open world, 3D voxel-based survival game that is extremely popular in the gaming, education, and AI research communities. A core feature of the game is the ability to remove and place blocks in a 3D voxel grid. With over 1000 unique block types, players are able to create a variety of structures. While the game has examples of pre-built structures (shown as an example in Figure 1), user-built structures are integral to the Minecraft gameplay experience and can vary in design; from survival homes to serve as a player's base of operations, to defensible military bases for player-vs-player experiences, to immersive fantasy structures to serve a story narrative.

It can be challenging and time consuming to create interesting buildings that suit a player's world or narrative needs. To address this, the Minecraft community has created websites full of downloadable, user-created structures that can be freely shared. These structures are typically encoded in a "schematics" file format that can be used with third party programs to import other player's structures into their game world. This approach is limited by the diversity and number of schematics files other users choose to upload, as well as a player's willingness to search through databases for a particular structure.

The wide availability of user created structures, as well as the need to generate assets (in this case, buildings) for player experiences motivates a procedural content generation via machine learning (PCGML) approach to generating Minecraft structures. Creating a tool to procedurally generate 3D structures can allow players and game developers alike to quickly design and implement structures that fit their building style or gameplay needs. Generated structures can serve as inspiration or a starting point for players who prefer the open-ended building aspect of Minecraft. Alternatively, players who care more about gameplay can use this tool to do the building for them, leaving them to focus on more interesting parts of the gameplay loop. PCGML for 3D structures can also help game developers create fully generated worlds. In Minecraft, nearly every aspect of the game world is procedurally generated, with the exception of village buildings. An approach for building generation could allow for an entirely generated game world.

In this paper, we introduce a system[1] capable of generating novel and aesthetically pleasing structures while reducing the time and in-game effort needed to build a 3D voxel structure from scratch. We use interactive latent variable evolution to evaluate the tool's capability in generating Minecraft structures that resemble real in-game buildings and help users guide the evolved population towards their ideal design.

FDG 2023, April 12–14, 2023, Lisbon, Portugal
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9855-8/23/04…$15.00
https://doi.org/10.1145/3582437.3587208

[1]https://github.com/TimMerino1710/Minecraft-Interactive-Evolution

2 BACKGROUND AND RELATED WORKS

2.1 Generative Models for 3D Content Creation

In the games and AI research community, Minecraft is understandably used as a 3-dimensional generative test domain. Some applications include creating settlements [12], regenerating designs [14], generating worlds from a single sample [1], and creating solvable mazes [10]. However, these studies do not focus on the procedural generation of Minecraft structures and buildings individually. Barthet et. al. specifically focus on procedurally generating novel Minecraft buildings using various autoencoder models [2]. However, these structures are only evaluated on diversity and novelty - and not aesthetic design - resulting in more artificial-looking structures. For our work, we incorporate human design decisions and aesthetic preferences into the evaluation of the generated 3D structures.

2.2 Interactive Evolution

Interactive Evolution (IE) involves humans in the evolutionary search process by having their selections act as the evaluated fitness function - thus guiding the evolved and generated population towards a desired state [15]. For procedural content generation in the games and AI community, IE has been used to evolve and generate race tracks [5], aesthetic game maps using Twitter [6], and 3d game assets [19].

Interactive Evolution can enable evolutionary search in domains where a traditional fitness function is difficult to define. In trying to evolve aesthetic Minecraft buildings, trying to define a measure of aesthetical appeal is very challenging, as each player will have their own subjective preferences and criteria. However, a drawback of IE is user fatigue, which increases with the number of generations.

In the domain of Minecraft, research conducted by Gbric et. al. with the EvoCraft API evaluated interactive evolution of Minecraft creations by tasking users to generate "interesting" structures. This evaluation was completed in the Minecraft engine itself and the generated structures were designed as abstract shapes, rather than habitable structures that a player might build. The authors also note that the generation and evolution time for the experiment was a concern that could lead to user fatigue and therefore decided to stick with the simplest IE strategy [8].

While this research was influential in our system, we sought to be more accessible to users and not require interfacing with the Minecraft engine itself. We also prioritized obtaining results in as few generations as possible to avoid user fatigue during the study. Rather than abstract structures, our goal to generate samples that look like they could be built by a player in Minecraft, and could function as a home or to anchor a game narrative.

2.3 Latent Variable Evolution

Deep Interactive Evolution using Latent Variable Evolution (LVE) involves mutating and evolving a population of vectors that is passed through a deep neural network to create new samples. Combined with IE, Latent Variable Evolution allows controllable evolution of generated samples. Bontrager et al. introduces and demonstrates its effectiveness in generating images [4]. Schrum et al. experiment with using Interactive Latent Variable Evolution (ILVE) to evolve

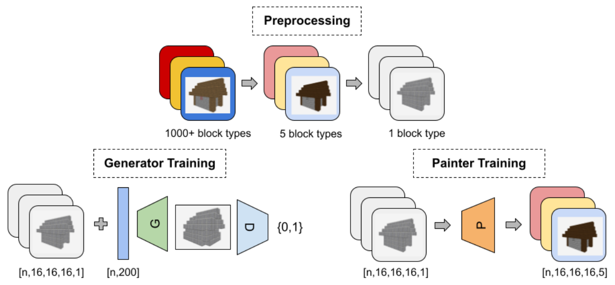

Figure 2: A diagram of the preprocessing and training steps for the generative and painter models.

generated levels for tile based 2D video games, The Legend of Zelda and Super Mario Bros [13]. Thakkar et al. use LVE with an autoencoder generative model to generate levels for the game Loderunner [16]. Volz et al. use LVE with a Generative Adversarial Network to evolve levels for Super Mario Bros [17]. However, there has yet to be an application of ILVE to 3D domains, as LVE for video game content generation is mostly done in the space of 2-dimensional games. Motivated by the success of Latent Variable Evolution for PCG in the level design space, we aim to expand to the domain to 3-dimensional games with Minecraft.

3 METHODS

3.1 System Description

Unlike other 3D generators, the generation process for this system is made up of 2 separate neural network models, which serve different purposes. The first model, a generative model referred to as the "Generator," aims to create structures out of a single block type that resembles the structural design of the training data. For this model, we chose a Generative Adversarial Network (GAN). Previous research has shown that GANs are capable of generating binary 3D voxel models that closely resemble training data [18].

The second model, referred to as the "Painter", aims to turn a binary structure into a structure composed of Minecraft blocks. We trained a 3D Convolutional Neural Network (CNN) to accomplish this task. For each Minecraft structure, a binary structure is generated by converting all non-empty blocks to 1, and empty blocks to 0. We then train the Painter in a supervised setting, using pairs of binary and original structures. The model tries to "paint" the binary structure to be identical to the original structure. Our goal with this model is to learn the aesthetic styles of existing Minecraft structures, and apply them realistically new binary structures created by the Generator.

Both models work together to generate a structure using Minecraft blocks from a random latent vector. First, the Generator transforms the vector into a binary structure. Then, that output is fed through the Painter network to introduce the different block types present in the game. The output of the painter model can then be viewed by the user. This represents the genotype-to-phenotype mapping used in our Evolutionary Strategy.

For the Interactive Evolution system, users evolve structures in an online interface by selecting from a population of structures created from this generative pipeline. New latent vectors for the population are evolved using crossover and mutation operators on selected latent vectors. These vectors are then passed through the pipeline again to create a new population. The following sections go into further detail for each individual step of the generation and evolution process.

3.2 Dataset and Preprocessing

3.2.1 Extraction. We aimed for a dataset of buildings that were relatively simple in style (i.e, not containing overly complicated shapes like spirals, and generally looked like a "house"). We limited our generated structure size to a maximum of 16^3 blocks, and only selected structures contained within that size for our training data.

We used in-game Minecraft structures — structures used for NPC villages (Figure 1) — as our "ideal structure design" to optimize our generative models towards. They are generally simple structures, meant to be homes and workplaces of in-game NPC characters, but also have diverse styles based on terrain biome. Based on our initial size and style constraints, we were only able to extract a total of 128 in-game houses for the system's training dataset.

Finding data on third-party sites that fit our criteria proved too time consuming. The range for structural designs was too complex, and many structures were too large for our 16^3 size limit. To obtain more data, we added the CraftAssist house dataset - a crowd sourced dataset of 2586 Minecraft houses, which were built by users within a 30 minute time window, with no additional constraints [7]. The unrestricted nature of this data allows us to capture more natural human creativity, which we aim to mimic with our research. In contrast, the in-game Minecraft structures were built by game developers to fit in with the base game's aesthetics. We manually assessed these CraftAssist structures to remove any bad data: empty builds, collections of random blocks, overly simple or 'cubic' structures, or "joke" builds such as humanoid figures. Removing these structures and applying our previous constraints left us with 219 suitable structures from the CraftAssist dataset.

3.2.2 Preprocessing. We represent each Minecraft structure using a 3D array of one-hot encoded vectors, where each index represents a block texture. To reduce the dimensionality of our data from the 1000+ possible Minecraft textures, we compressed the structures to use a smaller number of block types. We aimed to have enough block types to achieve aesthetic diversity in our builds while significantly decreasing our data size.

For our final models, we use 5 different block types: "Air", "Stone", "Dirt", "Wood", and "Slab". We chose to ignore special condition blocks such as "half" size blocks and blocks with varying orientations such as "stair" blocks by compressing them into the "Slab" texture. This compressed representation becomes our "categorical dataset," while a binarized form becomes the "binary dataset". In the binary dataset, all air blocks are represented by 0, while non-air blocks are represented by 1. Through this compression scheme, many block types were deleted from structures and replaced with air. This led to some structures with "floating" blocks. To remove these, a 26-neighbor connected component algorithm was used on each structure, and only the largest connected component was kept.

3.2.3 Dataset Augmentation. To compensate for the relatively small dataset of only 379 unique structures, we used various augmentation strategies for both the Generator and Painter Model datasets.

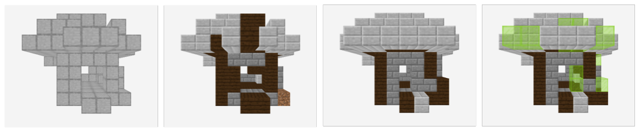
Figure 3: Effects of subtractive noise augmentation. From left to right: binary structure, painter without noise, painter with 30% subtractive noise, highlighted additions from #1 to #3

For the Generator dataset, we combined multiple augmentation strategies: rotation about the vertical axis, stretching the structure along one or multiple axes, transposition across the X-Y plane, and voxel noise. The dataset was also up-sampled to a size of 64^3 blocks, which resulted in much more stable performance and realistic outputs by our generative models. Our best performing Generator was trained on a dataset of in-game houses with transposition and rotation augmentations. Although generators trained on a stretched dataset were able to learn to create "longhouse" and "tower" shaped structures, the overall realism of the builds was diminished.

We noticed during preliminary experiments that the majority of the structures output by our generative model appeared to be missing blocks. Inspired by masked autoencoders, we added a "subtractive noise" augmentation to the Painter model's dataset. This was performed by randomly removing a percentage of non-air blocks from the binary training data, while leaving the "true" data (the textured structures) untouched. This augmentation gave the Painter a dual goal: to paint the house, and to repair it by fixing "holes" where it learned was appropriate. Figure 3 depicts some of the repairs made by the augmented Painter models, such as adding blocks to complete a sloped roof, or creating a 2-block tall doorway. We found that 30% removal gave the best performance, and led to a noticeable improvement in the perceived quality versus non-augmented Painter models.

3.3 Generative Structure Model

For our generator model, we used a Generative Adversarial Network, with similar architecture to the 3D-GAN model used by Wu et al. [18]. Rather than a standard 3D-GAN, we used a Wasserstein GAN with Gradient Penalty (WGAN-GP). WGAN-GP is a state-of-the-art GAN technique utilizing the Wasserstein distance as a loss function, as well as a gradient penalty term to constrain the critic (discriminator) network function to be 1-Lipshitz. In practice, WGAN-GPs seem to offer better convergence than a standard WGAN[9], and have been shown to work with LVE[4]. We found that a WGAN-GP outperformed 3D-GANs and standard WGANs on our dataset.

The generator model was trained on the in-game binary structure dataset. Our best performing WGAN-GP model was trained for 3000 epochs, with batch size 32 and latent vector size 200. The Critic network consists of four 3D convolutional layers with kernel size 4. Downsampling is performed each layer via striding. Leaky Relu activation layers follow every convolution layer.

Our Generator network consists of 4 3D convolutional layers with kernel size 4, each followed by a 3D upsampling layer. Relu activation functions and batch normalization layers follow every

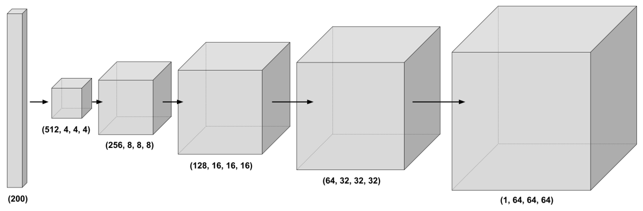

Figure 4: Architecture diagram of the Generator portion of the WGAN-GP network. The critic network is mirrored in layers and filters

convolution. Batch normalization is purposefully omitted from the critic network to avoid issues specific to WGAN-GPs. We train the critic 5 times for every 1 generator update, following the original WGAN-GP paper[9]. We use a RMSProp optimizer with a learning rate of 8e-5 for both networks.

Though we hoped a WGAN-GP trained on categorical data would be able to generate realistic Minecraft structures, attempts to train one were unsuccessful. However, models trained on binary data were much more successful in capturing the structure of the training data. This motivated the dual model approach, using a second model to reintroduce block variety into the generated data.

3.4 Painter Model

The Painter model was trained on a categorical dataset of both in-game and Craftassist structures. We chose a 3D Convolutional Neural Network consisting of four 3D convolutional layers. The first 3 layers are followed by dropout layers, with dropout rate = 0.3. Our best painter model was trained for 500 epochs, with a batch size 64, using categorical cross entropy loss and an Adam optimizer.

The output of the Painter model is a one-hot encoded representation of the input structure (size = $16^3 * n$, where n is the number of possible texture blocks). Figure 5 demonstrates the training pipeline for the painter model with architecture details.

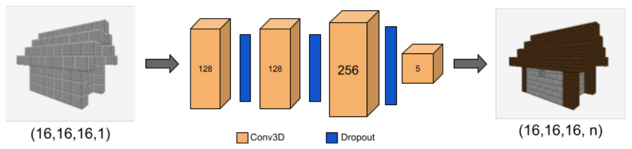

Figure 5: Architecture diagram of the Painter Model with the binary form of the house as the input and the textured output of the same house.

Combined with the generator model, the painter model can produce a structure with a variety of block types that are placed in positions that are similar to the real Minecraft and human-authored structures (i.e. using wood-like textures for "roof" blocks, and stone-like textures for "wall" blocks), This process also splits the training workload and learning curve for the generator model.

It is important to note that the output of the Painter model represents the final phenotype, the structure to be evaluated. However, this split training process does not learn a direct genotype-to-phenotype mapping. This means that while small latent vector mutations cause small changes in the structure of a building, they may have more dramatic changes in the "style" of the building.

We tested other architectures for Painter models, such as autoencoders, CNNs with different kernel sizes, and different layers such as attention, layer normalization, and squeeze-and-excite convolutional layers. However, this simple CNN architecture had the best aesthetic result on the generated houses, and was therefore used as the Painter for the experiment.

3.5 Structure Rendering

Previous works involving Minecraft-based PCG systems have used the actual game of Minecraft as render system verification for their generation outputs [7, 10, 12]. However, to maximize accessibility to users who may not have access to the game, we needed a system that did not require the game's engine to render a given structure.

As such, we created an open-source Javascript interface called VoxWorldJS [2] that allows for quick rendering of structures from 3D integer arrays using Minecraft textures. The structures are rendered using the Three.js library and allows rapid exporting of structures as PNG images or rotating GIFs via NodeJS libraries. This method speeds up the rendering process and visual output for the user, and allows the generated structures to be rendered browser-side with multiple camera angles without the need for a player navigating the 3d space in the Minecraft game engine.

3.6 Online Interactive Evolution

Users can interact with the system online and evolve a population of generated structures directly from the pipeline. The website [3] is hosted on an Amazon Web Services server and uses a Flask library setup to communicate between the Python-based neural networks, Javascript-based structure rendering, and the user's real-time input. Users can hover their mouse pointer over the generated structure images to see a GIF of the structure rotating around the vertical axis.

The evolutionary process uses a genetic search evolutionary algorithm, with the latent vectors as the "genotype," and the corresponding generated structure as the "phenotype". We use mutation and crossover operations on the latent vectors as our genetic operators. Selection is done entirely via user selection; only selected houses are used for the next generation as "parent" vectors. We use a population size of six for all generations.

Each vector in the starting population is a Gaussian random vector $\vec{X} = (X_1, ..., X_{200}), X_i \sim \mathcal{N}(0, 1)$ — identical to the distribution of latent variables used during GAN training. Mutation is done by adding random noise vectors to the copies of the parent latent vector. The noise vectors are also Gaussian random vectors, $\vec{Z}_{noise} = (Z_1, ..., Z_{200}), Z_i \sim \mathcal{N}(0, 0.3)$. Crossover is done by first averaging all parent vectors into $\vec{X}_{avg}$, then applying the same mutation process to four copies of $\vec{X}_{avg}$. Additionaly, one parent vector is selected to remain unchanged in the next population, and one new latent vector is generated to add diversity. The value $\sigma = 0.3$ for the noise vectors was determined through experimentation and subjective evaluation of the evolution process. We sought a balance between user fatigue and stability of evolution, where the change

[2]https://github.com/MasterMilkX/VoxWorldJS/
[3]http://www.minecraft-house-evolver.xyz/

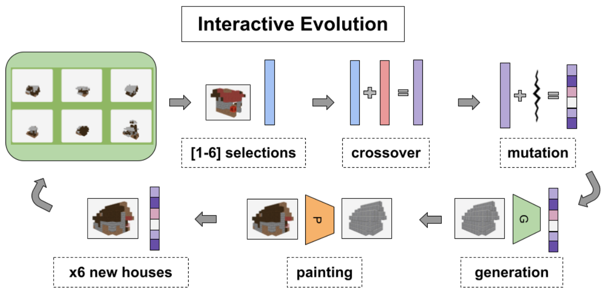

Figure 6: A diagram of the interactive evolution pipeline.

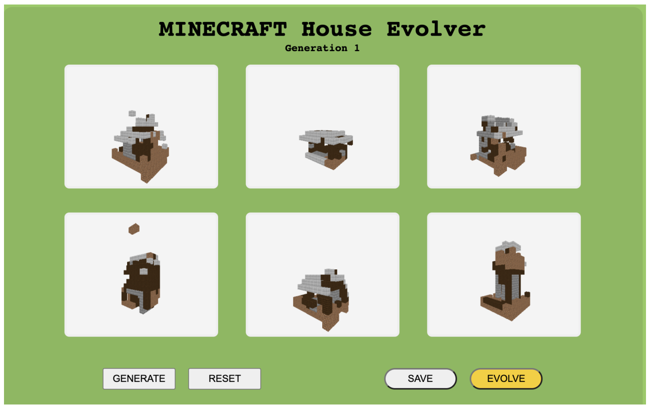

Figure 7: The Minecraft Interactive Evolution study website.

between generations would be dramatic enough to reach a target evolution goal within a few generations, but small enough that each member of the new population seems like a natural evolution from the selected parent(s).

The size of the evolved population is limited to six to reduce user-end wait time for generating, painting, and rendering each structure. The average wait time for six structures takes 25 seconds. Users can also export the 3D integer array of a generated structure as raw text for use in other systems (such as the VoxWorldJS system.) Figure 7 shows a screenshot of the interactive evolution website.

4 USER STUDY

A user study was performed to collect information about user design choices, and evaluate the efficacy of the system and the interactive evolver. This study was modeled after the interactive LVE experiment performed by Bontrager et. al. [4] and the Minecraft structure generation experiment by Grbic et. al. [8]. Participants volunteered to answer questions on a Google Form about their experience with the system.

Demographic questions were presented first to gather insight into each user's general design style. These included questions about the users' ages, experiences playing 3D voxel-based and sandbox games, and using interactive evolvers. These questions allowed us to gauge how familiar particpants were with Minecraft and the aesthetics of Minecraft structures, as well as their overall experience level with AI-assisted interfaces.

Similar to the IE experiments in Evocraft, we tasked users with completing two different evolution experiments using the system. The first task was a guided evolution: users were asked to select 1 of 3 design descriptions, then evolve a structure until it reached that design goal. The 3 options for design were: 'A house with a door and a window', 'A tower (approximately 8 blocks high)', or 'Two separate houses in a single sample'. This aimed to measure how effective the system would be for a user who already had an idea of what they wanted to build.

The second task was an open ended evolution. Users were asked to evolve a structure of any design they wanted, then write a text description of their generated structure. This experiment aimed to gauge human creative preferences, as well as explore the variety of structures our model were capable of producing.

In both experiments, users reported the number of generations needed to reach their goal. This gave insight into user fatigue, an important consideration in any human-in-the-loop system and a primary motivation behind our ILVE approach. Finally, users were asked to rate their experience using a five point Likert scale, representing their agreement with 12 provided statements (Figure 1). An optional free response section was included for comments and criticisms.

5 RESULTS

We released the study online over the authors' social groups such as Twitter, Discord, and Slack. Over the course of five days, we received 25 responses. The study took an average of 10 minutes to complete, which included learning to use the interactive system and answering questions on the form. No sensitive or identifying information was required to be provided by users.

5.1 Demographic Information

The self-reported ages of the users were between 14-35 years of age, with a strong median of users of aged 26. A large portion of users (40%) reported preference for goal-oriented tasks over open-ended tasks, and 52% of users said that they enjoyed both tasks equally. Many users (60%) had used an AI-assisted generation tool (i.e. DreamFusion, ChatGPT, Picbreeder) before the experiment and were familiar with online AI generators.

All participants reported playing video games on a weekly basis, with more than 84% reporting that they played for 2 or more hours a week. 92% of users had played a sandbox game (games involving crafting, designing, or open-ended gameplay), while only 64% had specifically played Minecraft. Of that 64%, we asked users what style of gameplay persona they identify most with. The possible personas are inspired by Bartle's Taxonomy of Player Types[3], but were specifically catered for Minecraft: Builders, Explorers, Fighters, and Achievers. Half of the participants in this subgroup identified most with the 'Builders' persona.

5.2 House Generation

Figure 8 shows some generated houses submitted by participants for the first task (goal-oriented generation). 72% of users chose to evolve a house with a door and window — most likely because the generator's default structure output were house designs. 24% of users chose to design towers, and a single respondent chose

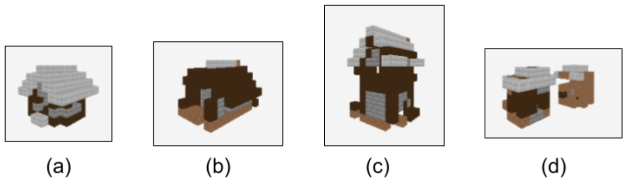

Figure 8: Goal-oriented houses evolved by participants. a,b) 'A house with a door and a window', c) 'A tower' and d) 'Two separate houses'

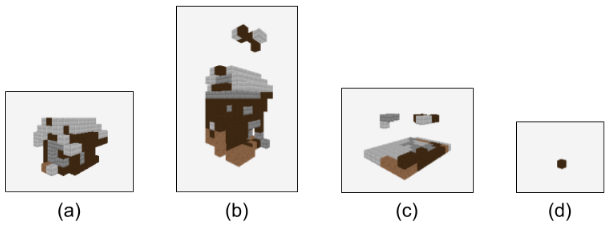

Figure 9: Open-ended houses evolved by participants labeled a) 'a more expensive house' b) 'a tower with a cloud idk' c) 'A Mario game level' and d) 'A single block!'

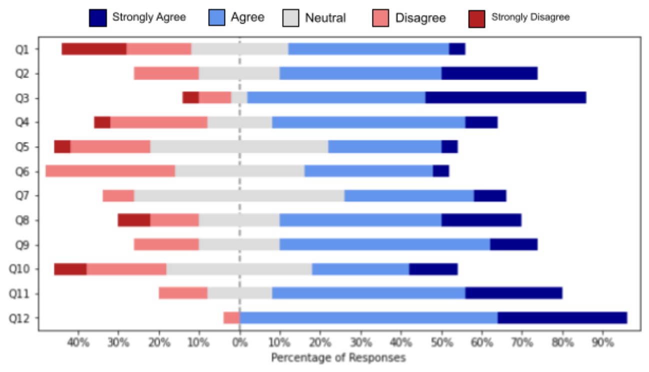

Figure 10: Results from the Likert-scale user experience questions.

to evolve two separate structures. We found that the majority of user-generated structures achieved the design goals described in their selected prompt. We consider this a success for the guided evolution aspect of our system.

Figure 9 shows some sample houses submitted by participants for the second task (open ended generation). Many users still chose to evolve a structure similar to the ones prompted from the previous task (houses and towers.) However, a few participants were extremely creative, either in description or ending structure. Some participants tried to make the most "chaotic" structures possible. Others gave very thorough descriptions of their structures such as "A fortress with stone foundation and roof". The majority of users were able to reach their goal in under 10 generations (64% for task 1, 72% for task 2).

5.3 User Experience Rating and Response

After completing the 2 tasks for structure generation, users were asked to complete a survey of about their experience using a Likert-scale format. Table 1 shows the list of statements given to participants. Q1-Q4 evaluate the usability and intuitiveness of the IE system. Q5-Q9 evaluate the output of the generator pipeline and gauge user satisfaction with evolution tasks. Q10-12 evaluate long-term satisfaction of the IE system and other AI-assisted design tools.

Users were in agreement with most statements concerning the aspects of the system. The statements pertaining to the generator pipeline had the most lukewarm reception. Most users were ambivalent in satisfaction to the generator's structure output (Q5), the painter's texture output (Q6), and how their selections were reflected in the output of the next generation (Q7.) To improve this, we will look to evaluating each model individually to further refine the output and make a more cohesive system.

Only 36% of users reported that they would like to use the system again (Q10.), though many agreed that generating the structures using this PCG method was faster than designing a house from scratch in Minecraft (Q8), and that the generated houses were novel and interesting (Q9.)

Conversely, users were especially satisfied with the GIF rotation images for their house selections, with 84% reporting 'Agree' or 'Strongly Agree' to Q3. Users also reported especially positive experiences for the long-term AI assisted question. 72% of participants were interested in 'creating procedurally generated 3d objects' (Q11) and almost 100% of participants reported 'Agree' or 'Strongly Agree' to being interested in using more interactive evolution tools for creative tasks (Q12.) Overall, users were enthusiastic to use more AI-assisted tools in the future for design and construction focused tasks — especially in a 3-dimensional domain.

Participants had the option of providing comments and criticisms they had about the system. As indicated by the ratings of the survey, a few users had difficulty understanding the UI of the system - particularly the difference between the 'Save' and 'Evolve' modes of the system. However, some users mentioned that the 'Help' menu we included to demonstrate how to use the system alleviated some of this confusion. Other users mentioned their disappointment with some of the generated outputs — mostly concerning the painter model's texture placements — noting as a recurring theme that they would "generate in places a human user would not be likely to place it for aesthetic purposes." Nonetheless, users reported an overall interest in the generation process and would like to see 'more variant structure types' and others were interested in '[creating] really weird structures.'

6 FUTURE WORK

To generate structures that truly feel like they belong in Minecraft, a wider range of blocks is needed. Further experiments can be done using higher-dimensional data. While compressing our datasets led to much better performance by our models, this represents only a small fraction of the 1000+ block types present in Minecraft. Our compression method also requires manual mapping of each block to a compressed category. Using a one-hot representation for each voxel led to sparsity, which scaled with the number of block types

#	Statement
Q1	I found the system intuitive and easy to use
Q2	The number of houses shown is satisfactory (not too many and not too few)
Q3	The GIF rotations of the houses were helpful in my selection process
Q4	The time it takes the model to generate new houses is reasonable
Q5	I am satisfied with the structural output of the generated houses
Q6	I am satisfied with the texture designs of the generated houses
Q7	I am satisfied with how my selections were reflected in the output of the next generation
Q8	Evolving the set of 6 different houses is faster than designing and constructing 6 houses from scratch in a 3d environment such as Minecraft
Q9	The generated houses are interesting and novel
Q10	I would use this system again in the future
Q11	I am interested in creating procedurally generated 3d objects.
Q12	I am interested in using more interactive evolution tools like this in the future for building designs either in video games or in other creative mediums.

Table 1: Table of statements used in the Likert-scale experience reporting of the survey.

being represented. It may be possible to avoid sparsity issues while increasing the number of represented block types via advanced feature processing methods. One such method is the *block2vec* token embedding used by Awiszus et al.[1].

One interesting area of research concerns the latent space of generative models such as GANs or autoencoders. Previous research has demonstrated the power of latent vector arithmetic within the latent space of a GAN [11]. It may be possible to isolate meaningful "attribute vectors" within the latent space of our generator (i.e. vectors that control the height or width of structures). These attribute vectors could be used to give users more control, and guide evolution in intuitive ways. We experimented with using interpolations between latent vectors as a way to measure the performance of our GANs, looking for "smooth" transitional structures at various points between 2 vectors. Our GANs were consistently able to interpolate smoothly between generated buildings, "morphing" from one building to another. Incorporating interpolations between members of a population can give a user more control than our current crossover mutation method does.

Although we had moderate success with this binary generator to categorical painter pipeline, for future work, we would like to develop a generator model capable of directly generating categorical structures. Preliminary results with this type of model were unsuccessful, mostly due to computational constraints, as the number of parameters in the network scaled with the dimension of the training data. Increasing both the depth of the GAN networks and the number of filters present in each convolutional layer may be the key to a successful categorical generative model.

7 CONCLUSION

This paper introduces a novel PCG method for generating structures within the game Minecraft by combining existing 3D voxel generative models with a binary-to-categorical transformation network. Using this pipeline, we demonstrate the ability to generate coherent structures in a higher-dimensional space. We also explore the use of Interactive Latent Variable Evolution within the generated structure space. Though Minecraft is very well known, this system is not intended to be limited to the Minecraft engine. We hope this system allows for more work to be done involving interactively evolved content generation systems in 3D domains.

REFERENCES

[1] Maren Awiszus, Frederik Schubert, and Bodo Rosenhahn. 2021. World-gan: a generative model for minecraft worlds. In *2021 IEEE Conference on Games (CoG)*. IEEE, 1–8.
[2] Matthew Barthet, Antonios Liapis, and Georgios N Yannakakis. 2022. Open-ended evolution for Minecraft building generation. *IEEE Transactions on Games* (2022).
[3] Richard Bartle. 1996. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD research* 1, 1 (1996), 19.
[4] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. 2018. Deep Interactive Evolution. In *Computational Intelligence in Music, Sound, Art and Design - 7th International Conference, EvoMUSART 2018, Parma, Italy, April 4-6, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10783)*, Antonios Liapis, Juan Jesús Romero Cardalda, and Anikó Ekárt (Eds.). Springer, 267–282.
[5] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. 2011. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 395–402.
[6] M Charity and Julian Togelius. 2022. Aesthetic Bot: Interactively Evolving Game Maps on Twitter. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 18. 18–25.
[7] Jonathan Gray, Kavya Srinet, Yacine Jernite, Haonan Yu, Zhuoyuan Chen, Demi Guo, Siddharth Goyal, C Lawrence Zitnick, and Arthur Szlam. 2019. Craftassist: A framework for dialogue-enabled interactive agents. *arXiv preprint arXiv:1907.08584* (2019).
[8] Djordje Grbic, Rasmus Berg Palm, Elias Najarro, Claire Glanois, and Sebastian Risi. 2021. EvoCraft: A New Challenge for Open-Endedness. Springer-Verlag, Berlin, Heidelberg.
[9] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved Training of Wasserstein GANs.
[10] Zehua Jiang, Sam Earle, Michael Green, and Julian Togelius. 2022. Learning Controllable 3D Level Generators. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*. 1–9.
[11] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. https://doi.org/10.48550/ARXIV.1511.06434
[12] Christoph Salge, Michael Cerny Green, Rodgrigo Canaan, and Julian Togelius. 2018. Generative design in minecraft (gdmc) settlement generation competition. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*. 1–10.
[13] Jacob Schrum, Jake Gutierrez, Vanessa Volz, Jialin Liu, Simon Lucas, and Sebastian Risi. 2020. Interactive evolution and exploration within latent level-design space of generative adversarial networks. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 148–156.
[14] Shyam Sudhakaran, Djordje Grbic, Siyan Li, Adam Katona, Elias Najarro, Claire Glanois, and Sebastian Risi. 2021. Growing 3d artefacts and functional machines with neural cellular automata. *arXiv preprint arXiv:2103.08737* (2021).
[15] Hideyuki Takagi. 2001. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proc. IEEE* 89, 9 (2001), 1275–1296.
[16] Sarjak Thakkar, Changxing Cao, Lifan Wang, Tae Jong Choi, and Julian Togelius. 2019. Autoencoder and Evolutionary Algorithm for Level Generation in Lode Runner. In *2019 IEEE Conference on Games (CoG)*. 1–4.
[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. Association for Computing Machinery, New

York, NY, USA.

[18] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. 2016. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 82–90.

[19] Du-Mim Yoon and Kyung-Joong Kim. 2013. Interactive Evolution of 3D Models based on Direct Manipulation for Video Games. *Procedia Computer Science* 24 (2013), 137–142.