Deep Learning Novelty Exploration for Minecraft Building Generation.

L-Università ta' Malta
Institute of Digital Games

Matthew Barthet
Institute of Digital Games
University of Malta

A thesis submitted for the degree of
Masters of Science in Digital Games

June, 2021

Abstract

Computational creativity (CC) refers to the study of computational systems that exhibit behaviors that an unbiased observed would consider creative. CC systems have shifted focus toward intrinsic motivation (IM) and open-endedness (OE), which are at the heart of creative behavior in biological systems. In this project, we apply these concepts to a procedural content generator that autonomously creates Minecraft buildings according to its own evolving definition of novelty. This work addresses a research gap in PCG, specifically in Minecraft, which currently focuses on generating adaptive settlement layouts without prioritizing creativity.

This system follows the fundamentals of the DeLeNoX algorithm. An autoencoder identifies the high-level features of buildings, compressing them into one-dimensional latent vectors. The system alternates between phases of exploration and transformation. In exploration, CPPN-NEAT is used to evolve populations of buildings using constrained novelty search. We calculate an individual's novelty as the average euclidean distance to the nearest K neighbors in the latent space. In transformation, the autoencoder is retrained with a dataset of the most novel individuals created in the previous exploration phase/s.

We experiment with different approaches to the retraining of the autoencoder and observe their impact on the diversity and complexity of the content generated. We assess the results quantitatively by comparing population diversities across experiments, and by visualizing their expressive range using a set of building properties. Finally, we compare the structures qualitatively and observe the effective change in complexity in the structures over time.

Our results show that the transformation phase is most effective when it uses larger training sets and includes examples from all previous iterations of the algorithm. This allows the system to more effectively scale in effective complexity of building features, which become more similar to examples of realistic buildings over time.

Acknowledgements

I would like to thank my supervisor Prof. Georgios N. Yannakakis for his constant support and feedback, and keeping me motivated throughout this project. I would also like to show my appreciation to the game AI group at the institute of digital games for providing support and input whenever I was in need. Finally, I would like to thank my family and friends for their moral support over the past two years.

Statement of Originality

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Can artificial systems create content that an unbiased observer would call creative? Is it possible for this process to be done intrinsically, without any externally designed and hard-coded function for quality? Are such systems capable of open-ended creativity, perpetually increasing in complexity by redefining themselves according to the content they generate? These questions are at the center of the field of *computational creativity* (CC) and artificial general intelligence, and define the objectives and motivations of this project.

Traditional research on CC typically generated content by searching a problem space according to some predefined, hard-coded objective function. The most commonly used algorithms for this form of search are evolutionary algorithms. They mimic natural evolution by evolving a population of genomes through mutating and combining high-performing individuals, thus promoting "survival of the fittest". As the goal of a system being modelled becomes more complex, the difficulty in designing its objective function increases dramatically, and evolution tends to fail by running into dead ends. Novelty search (NS) was created by Lehman and Stanley (2011) in response to the problems of traditional objective search. NS abandons objectives in favor of rewarding individuals based on their novelty, i.e., how different they are compared to the content seen so far. This approach searches the problem space more thoroughly and avoids running into dead ends, but loses the ability to promote desired qualities in the population due to the lack of an objective. Quality-diversity (QD) algorithms build upon the original goal of novelty search, simultaneously promoting diversity in the population and high performance in terms of one or more objectives (Mouret & Clune, 2015).

This shift toward abandoning objectives and focusing on novelty for computational creativity is in line with observations in the field of psychology, which has been an inspiration for AI since its inception. It is well established that external motivators oppress the creativity of individuals, who typically flourish when acting based off their own inherent curiosity and no ulterior motive (Guckelsberger, 2020). In computational systems, intrinsically defined reward functions are critical to achieving open-ended creativity and effectively exploring a search space. By allowing the generator to redefine its measure for novelty/creativity according to its own observations, we allow it to adapt and expand its capabilities beyond its initial capabilities. In section 2.1.1 and section 2.1.2 we cover this theoretical background in more detail.

Sandbox games such as Minecraft are arguably the perfect canvas to illustrate a system's creativity due to its open-ended gameplay and lack of an immediate, linear objective for the player. Minecraft generates infinite worlds and fills them with structures, resources, and monsters that the players must exploit to survive. However, whilst buildings and settlements in Minecraft need a degree of functionality and adaptability to their environment to be useful, the game also allows the player to creatively express themselves through their designs. Current research into PCG for Minecraft places more focus on the objectives laid out in the Generative Design in Minecraft Competition (Salge, Green, Canaan, & Togelius, 2018) to design settlements and buildings, without emphasizing creativity in the generation process. In fact, the buildings generated are often static, rule-based system that do not involve any optimization methods at all. We position this work to fill in this space, and potentially get the ball rolling for creative building generators, as well as other aspects of Minecraft settlement generation. The objectives of such systems are complex and multi-faceted, and could fall victim to the problems seen in traditional objective search, which could generate more interesting and usable content through a diversity-centric approach.

We adapt and extend the DeLeNoX system (Liapis, Martínez, Togelius, & Yannakakis, 2011) for the task of autonomously creating increasingly complex and novel Minecraft buildings. On a high level, the algorithm alternates between two phases; exploration of the search space through constrained NS using CPPN-NEAT, and transformation of the novelty function through training deep learning autoencoders. DeLeNoX was chosen as the platform for this project as it aligns with our goal of an intrinsically motivated and open-ended content generator. It achieves open-ended creativity by continuously characterizing its own search biases and redefining its novelty function in response. This allows the generator to continually focus its attention on new promising areas of the search space that would have been unlikely to have been visited due to its original biases. This concept satisfies the properties of IM and allows the generator to more closely exploit its full creative potential that would otherwise be unrealized with an extrinsic objective or static novelty function. Whilst this approach has the potential to create unique content entirely on its own, it could also be applied in a mixed initiative environment and be used alongside humans (Yannakakis, Liapis, & Alexopoulos, 2014). In such a scenario, the generator could help promote the creative process in human designers through evolving its own version of creativity in tandem with its user, improving their potential efficiency and creative output.

To evaluate the generator, we test a variety of configurations of the transformation phase of DeLeNoX and observe the differences in the generated content according to a number of evaluation metrics. We use a number of quantitative measures to analyze the diversity of the content generated, the expressive range of the generator according to a number of building properties and accuracy of the autoencoder for compressing the buildings. We also qualitatively look at the generated content to compare the visible structural complexity between experiments. This allows us to identify the preferred transformation method with the current approach and compare the optimal configuration to realistic, human authored buildings.

1.1 Research Questions

In this project, we attempt to answer the following four core research questions:

RQ1. Can a generator create Minecraft buildings using an intrinsic definition of novelty?

RQ2. Can such a system exhibit effective open-ended creativity?

RQ3. Does the content generated by the system become more realistic over time?

RQ4. Does the system's model become more robust to unseen, realistic content over time?

In RQ1 we seek to identify whether our generator can be effectively used to create detailed and feasible three-dimensional structures. Whilst existing generators have researched generating three-dimensional content using constrained novelty search, they do so using a static definition of novelty, which could restrict their potential for open-ended creativity. Existing work using an evolving novelty function such as DeLeNoX have traditionally done so in a 2D domain, generating interesting arcade assets or shapes. We detail our approach to answering this question throughout chapter 3 and give examples of generated buildings in section 4.2.5.

Through RQ2 we establish whether our approach is capable of producing content which clearly increases in complexity and novelty without converging to an area of the phenotype space. We do not factor genetic complexity when discussing the system's OE, which is guaranteed to increase through CPPN-NEAT. Instead, we focus on how the visible complexity of the generated structures evolves over time. We discuss the system's scalability and performance over time in section 4.3, and it's ability to exhibit different types of OE in section 4.3.1.

Furthermore, in RQ3, we attempt to answer whether our generator is naturally able to produce more realistic structures without providing it with any external definition of a building. In a similar vein, through RQ4 we observe whether the system's model is able to more accurately compress realistic buildings it has never been trained on before. We answer these questions in section 4.2.3 and section 4.2.6 respectively.

Summary

In this chapter, we established the objectives of this project how they relate to the field of CC and the concepts of IM and OE. We underlined why Minecraft is a great platform for testing creative generators, and identified a lack of research on open-ended generators for Minecraft buildings. We gave a high-level overview of the DeLeNoX system and why it was chosen for this project, and our approach to evaluating the proposed system. Finally, we listed and explained our four research questions we aim to answer through this project. In the next chapter, we give a detailed overview of the fields of computational creativity and procedural content generation, and cover some existing work that is relevant to this project.

Chapter 2

Background

The fields of *computational creativity* (CC) and *procedural content generation* (PCG) form the backbone of the motivation and design of this project. What constitutes creative behavior for a computer program? What is *intrinsic motivation* (IM), and how does it relate to *open-endedness* (OE) in evolutionary systems? How does one assess the quality and creativity of a generator? In this chapter, we introduce these concepts and attempt to answer these questions in enough detail to build a solid theoretical framework. In doing so, the generator's design (Chapter 3) and the approach to its evaluation (Chapter 4) can be better understood and justified.

This chapter is organized as follows. In section 2.1 we describe the field of CC, looking at how games can benefit from it, and how they can be used to enrich academic research in the field. This is followed up with a look at OE, outlining the three different types of novelty and OE a system can exhibit. Armed with this foundation, we close off the section with an explanation of IM from a theoretical and computational perspective, highlighting its importance in relation to the objectives of this project. Section 2.2 dives into the field of PCG for games, briefly summarizing its development over time and its contribution to the game industry and CC. We move on to cover the current state-of-the-art methods for PCG with a focus on the *quality-diversity* (QD) family of optimization methods, along with a brief look at some other relevant algorithms used in game AI. We close off this section with a discussion on methods for assessing the expressiveness and creativity of generators, which is a critical for the evaluation of any PCG system. Finally, in section 2.3 we give an overview of existing work on open-ended generators for a variety of use cases. To close off this chapter, we give a short description of Minecraft, the platform used for this project, and the Generative Design in Minecraft Competition (GDMC), giving insight into the current state of PCG in Minecraft.

2.1 Computational Creativity

The field of CC has been actively researched for well over two decades. Early work in the field explored the potential of autonomously generating artifacts in a variety of domains, such as music Wiggins, Papadopoulos, Phon-Amnuaisuk, and Tuson (1998) and stories Peinado and Gervás (2006). Colton and Wiggins (2012) defined CC as:

> The philosophy, science, and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviors that unbiased observers would deem to be creative.

In other words, it is the study of computer systems which generate content/behavior that unbiased humans would consider interesting or creative, and that can challenge humans both creatively and scientifically (Colton, De Mántaras, & Stock, 2009). This approach places humans at the center of definition of creativity, excluding the potential for the generator/agent to be a judge of creativity. Therefore, we use the working definition given by Guckelsberger (2020), as it also ties in with the concept of IM and OE which are central to this project:

> Computational creativity (CC) is the explicit, multi-disciplinary study, both theoretical and applied, of creativity in any type of computational system. It considers creativity as an open-ended concept, at any level of complexity and from any viewpoint.

This early research into CC approached the subject using isolated, well-controlled and single-faceted domains. Liapis, Yannakakis, and Togelius (2014) identify games as the cutting edge application for the study of CC, due to three major factors. First, because games can be considered as *multi-faceted*, offering multiple creative avenues to design sound, visuals, graphics, interaction, narrative, interaction and more. Second, games are *content-intensive* processes with open boundaries with respect to creativity, which vary massively from one facet to the next. Finally, games provide rich interactions with the player and provide the goal of creating systems which are useful (in this case, playable) and creative. They also point out that games have a long-standing history with autonomous creative systems in the form of PCG, which has been a selling point and tool for game development for many years. Therefore, Liapis et al. (2014) introduce the field of *computational game creativity* (CGC), which refers to the study of CC as a method to improve games, as well the use of games as a canvas for creative processes. Whilst these creative systems can be used to autonomously generate interesting content or behavior, they can also be used as a mixed-initiative tool alongside humans (Yannakakis et al., 2014), helping them to understand and promote the creative process (Colton & Wiggins, 2012). In relation to the facets of CGC, this work falls within autonomous generation of level design/architecture, in that the content being created are in-game structures that can be imported into Minecraft.

Before taking a look at OE, we need to establish the difference between two traditional types of creativity in creativity theory; *exploratory* and *transformative* creativity. To help understand this idea, it's helpful to conceptualize creativity as a form of searching within a possibility space of individuals. Boden and et al. (2004) refers to exploratory creativity as the search for interesting points within the boundaries of the possibility space. This form of creativity can be viewed as straightforward problem-solving, looking for new interesting approaches within the current context. In terms of computational models, exploratory creativity can be seen as effectively exploring the entirety of the search space, without attempting to break the mold of the current boundaries of the space. On the other hand, Boden and et al. (2004) refers to transformative creativity as an altering of the boundaries that constrain the search space to gain access to new areas or dimensions previously unexplored. This can be thought of as a major breakthrough which causes the system in question to redefine its original approach to the task.

Another relevant and interesting theory of creativity was introduced by Schmidhuber (2007). In this theory, an agent assesses how interesting or creative an artifact is based on how well it can identify high-level patterns in its structure. From a computational perspective, an agent judges creativity/novelty by its ability to compress the individual into a lower dimensional representation. In this model, agents get bored by patterns it considers predictable, or by individuals it cannot identify at all. Curiosity and creativity are interwoven, meaning agents seek out environments which satisfy their curiosity, but these environments must be understandable to them to consider them creative, requiring a balance between the two. The DeLeNoX system is based off this theory and is described in section 2.3, playing a fundamental role in the implementation of this project.

2.1.1 Open-Endedness in Evolutionary Systems

When looking at early literature on OE, a classical broad definition of the concept emerges; it can be seen as the perpetual, unbounded production of increasing novelty and/or complexity (T. J. Taylor, 1999; Bedau, 1991; Hutton, 2002). Lehman and Stanley (2011) use a general approach to OE in their introduction of the novelty search algorithm, referring to it as the behavior exhibited by evolutionary systems that continually produce novel forms/behaviors. Banzhaf et al. (2016) informally define the concept of OE as the ability of a biological or artificial system to keep producing novelty and/or complexity without exhaustion. They specify three major categories of OE processes; those that do not have a termination condition, those that do not have a specific objective, and those that use a combination of the two. Banzhaf et al. (2016) also introduce the notion of models and meta-models, which are key concepts for this topic. In their definition, the purpose of a model is to provide an abstract language for the concepts in a domain and can fall under two main categories; scientific or engineering models. Scientific models are used to classify, predict, understand and explain something without taking into account outside the scope of the model - in the case the model and reality disagree, the model is wrong and needs to be corrected. Engineering models are models of a system that is going to be built in the world - if the model and reality disagree, reality wrong and must be debugged to conform with the model. Banzhaf et al. (2016) define meta-models as a method for providing a clear language to define models and provide the concepts required to build models. In other words, the meta-model defines the system on a highly abstract level, and is used to define the domain and dimensions of the possibility space.

Through the notions of models and meta-models, Banzhaf et al. (2016) introduce three classes of novelty and OE, which were adopted by T. Taylor (2019) and renamed to Exploratory, Expansive, and Transformative OE. Each class of OE has its own type of novelty, which is defined by the extent to which it necessitates modifications to the system's model and/or meta-model.

1. **Exploratory OE**: the perpetual production of exploratory novelty. This novelty is described using the current model used by the system and does not require any modification of the model or meta-model. In exploratory OE, the system explores a predefined state space given by the model. This follows the definition of exploratory creativity given by Boden and et al. (2004).

2. **Expansive OE**: the perpetual production of expansive novelty. This novelty requires a change to the system's model, but still uses the definitions provided by the meta-model (T. Taylor, 2019). In expansive OE, the model is changed so that the size and structure of the state space are varied to increase the potential types of species that may be observed. The dimensions and domain of the search space remain unchanged as the meta-model is left unaltered, but are expanded to include more potentially interesting data points.

3. **Transformative OE**: the perpetual production of transformative novelty. This novelty requires the introduction of new concepts to the system, requiring changes to be made in the meta-model. These changes introduce the possibility of new states in a different domain/dimension of the search space (T. Taylor, 2019). T. Taylor (2019) gives the example of winged flight being discovered in a biological system that did not previously contain that concept. Expansive and transformative novelty both fall under Boden and et al. (2004)'s definition of transformative creativity.

T. Taylor (2019) identifies multiple routes an evolutionary system can take to achieve OE. The most relevant routes for this project are through the three following evolutionary processes; generation, evaluation, and reproduction. The generation process maps a given genotype to its phenotype (G-P map). Evaluation determines the evolutionary significance of an individual according to the current evaluation function. Reproduction produces new individuals from the past generation according to a reproduction function. He discusses the benefits of intrinsically instantiating these evolutionary processes as opposed to hard coding them, allowing them to evolve alongside the system to be more adaptable and cover more of the search space. For example, by having the GP-mapping intrinsically implemented by each organism and allowing it to evolve, a biological system is likely to produce more adaptive variations in the search space (T. Taylor, 2019).

2.1.2 Intrinsic Motivation

Psychology has been an important source of inspiration throughout the development of AI and CC. In particular, we focus on motivation, which refers to the drive that influences an agent's choice of behavior (Ryan & Deci, 2000), and can be observed in biological and artificial settings. It has been established that motivation varies not just in levels of intensity, but also in orientation (type of motivation), which are called *intrinsic* and *extrinsic* motivation. In this section, we cover the difference between these two orientations based on the theoretical and systematic review done by Guckelsberger (2020), discussing how IM in particular is desirable to promote creativity and OE in artificial evolutionary systems.

IM refers to an individual's desire to take certain actions because they are inherently interested in the outcome or find the activity enjoyable or satisfying (Harlow, 1950). Intrinsically motivated individuals act freely, without influence from external sources, and do so without a separate motive other than their own interest (Ryan & Deci, 2000). On the other hand, extrinsic motivation is accompanied by an external influence assigning value to the task in question. This could be due to an explicit third party influencing the individual into taking an action, but could also be an internal drive that is implicitly driven by an external circumstance (De Charms, 2013). For example, a student can be internally motivated to study and perform well in their exams, but are propelled to do so by an external separable objective (their grades), and not by an inherent curiosity/desire to understand the subject of their studies.

IM is responsible for learning behaviors that are potentially beneficial in the long term, but that would not necessarily have been immediately obvious or imposed on them by their environment (Guckelsberger, 2020). IM plays a big role in the creativity of an individual; it's widely accepted that external factors can hamper the creative behavior of an individual, who tend to thrive creatively their actions are based solely on their own curiosity. Through their psychological account of IM, Guckelsberger (2020) suggests that it is the nature of the reward function, not the action-selection mechanism, that influences the orientation of a computational model's motivation. *Reinforcement learning* (RL) and PCG have typically used extrinsic rewards through objective functions to propel an agent to a desired behavior. However, as we will see in section 2.2, such extrinsic signals are difficult to engineer manually, and are often *sparse* and *deceptive*, harming the agent's realized behavior (Lehman & Stanley, 2011). Therefore, there's been a push to abandon objective functions entirely through algorithms such as novelty search (Lehman & Stanley, 2011) and surprise search (Gravina, Liapis, & Yannakakis, 2016), which have been proven to produce more interesting and creative behavior whilst attaining better task performance.

Guckelsberger (2020) identifies the four following informal properties that a reward function should exhibit to be considered an intrinsic reward signal. The reward must be computed internally (from the perspective of the agent), without using any external components or variables that are not a part of the agent. It must be free of semantics, meaning it's not dependent on a component's assigned meaning, but on the distribution and relationship of the components as a whole. The signal should also be universal, meaning it's compatible with any agent yet sensitive to their inputs and environment. Finally, the agent should exhibit open-ended development, improving its skill and knowledge and leveraging all degrees of freedom provided by the environment.

Finally, Guckelsberger (2020) also provides a detailed review of existing work in IM for computational models. The most critical findings for this project were the uses of IM for increasing creative autonomy, as a substitute for difficult to model extrinsic rewards, and as a method for modelling transformative creativity. Increasing creative autonomy is the fundamental goal for this project's generator, which needs to generate its own subjective definition of novelty and increase in effective complexity to achieve OE, as seen in its definition in 2.1.1. The use of IM as a substitute for extrinsic rewards is also of importance for a building generator, as an objective function is very difficult to define (what makes a good or interesting building?). IM is critical to effective and efficient exploratory behavior, but may also be used as the driving force behind transformative creativity and the OE of a generator. As we've seen, in order to exhibit transformative OE a generator needs to be capable of searching the problem space thoroughly whilst discovering new dimensions and domains in the search space, which requires a completely self-defined and evolving reward for creativity to do so.

2.2 Procedural Content Generation

PCG has a long-standing history with the game industry, dating as far back as the popular dungeon crawler game *Rogue* (AI Design, 1980). The use of AI has provided the opportunity for the game industry to keep up with the ever-growing demand for more content from players, which is expensive to produce both financially and in terms of development time. Over the past decade, PCG has grown in the academic community with the advent of the *search-based* approach, and there now exists a rich amount of literature on the subject.

Togelius, Yannakakis, Stanley, and Browne (2011) define PCG as the algorithmic means for creating game content automatically, be it for digital or analogue domains. They refer to content as any aspect of the game apart from the behavior of NPC's and the game engine itself, and can be decomposed into the facets introduced by Liapis et al. (2014) for CGC. In this project, we are concerned with *offline* search-based procedural content generators that produce *necessary content* (Togelius et al., 2011). Search-based PCG is a special approach to generating content which treats the problem as a heuristic/stochastic optimization task

(Togelius et al., 2011). The critical part of any search-based generator is the design of its objective/fitness function, as well as the constraints used to ensure the feasibility of its output. Traditionally, the design of the fitness function is highly-dependent on the type of content being generated. For example, when evaluating generated game levels for an RTS game, the fitness function can be inspired by game theory and design patterns, addressing concepts such as area of control, balance, and exploration (Liapis, Yannakakis, & Togelius, 2013). Hard and soft constraints are generally used to restrict the search space of such a generator to a desired subset of possible game levels, and to prevent the generator from creating completely unusable levels.

However, as the objective being modelled becomes more difficult/complex, evolutionary algorithms tend to fail and converge to a local (and generally sub-optimal) optimum due to *deception* caused by the fitness function (Lehman & Stanley, 2011). Deception occurs when the objective function actively misdirects the search towards dead-ends in the search space, tricking it into converging to sub-optimal, local optima. Approaches have been proposed to mitigate this issue by maintaining some level of population diversity. One approach is to change the mechanism for genome selection in an attempt to find solutions in distant parts of the search space and achieve more reliable OE. However, this leaves the root cause of the problem unaddressed. In response to this issue, focus shifted onto algorithms that optimize populations for diversity through some novelty function. Lehman and Stanley (2011) found that by abandoning the objective function entirely through novelty search, they observed a significant increase in performance in maze navigation and biped robot movement. This method avoids the pitfalls of deceptive objectives and searches the behavior space thoroughly, identifying macro-behaviors that would have otherwise remained unexplored in an objective based approach. Gravina et al. (2016) introduced surprise search, which differs from novelty search in that it models expected behavior and rewards individuals that differ from what is expected. Gravina et al. (2016) found that surprise search matched novelty search in terms of identifying optimal individuals in maze navigation, whilst being more efficient and producing more diverse genomes in evolution.

However, as Lehman and Stanley (2011) point out, there are limitations to ignoring objectives entirely, the biggest being that optimal individuals may be ignored as they happen to be classified as not novel. QD algorithms aim to address this issue by augmenting algorithms like novelty search with a push toward identifying unique individuals with the best possible performance (Pugh, Soros, Szerlip, & Stanley, 2015). One of the most notable QD algorithms is MAP-Elites (Mouret & Clune, 2015), which is an illumination algorithm designed to return a set of diverse, high-performing individuals. It does so by defining a number of dimensions that make up the features space, and uses an objective function to evaluate individuals. The feature space is discretized and the highest performing individuals (elites) are kept in each cell. These are evolved by selecting elites according to a selection function and producing new offspring. A new individual is stored in a cell if the cell is empty or the new individual has a better fitness than the current elite. Once terminated, the algorithm returns the map of elite individuals, giving a set of high performing individuals that is diverse in terms of the feature space provided. This method produces individuals that have good quality (elites score well with the objective function), but preserves diversity through the archive of elites which are diverse along the dimensions of the feature space.

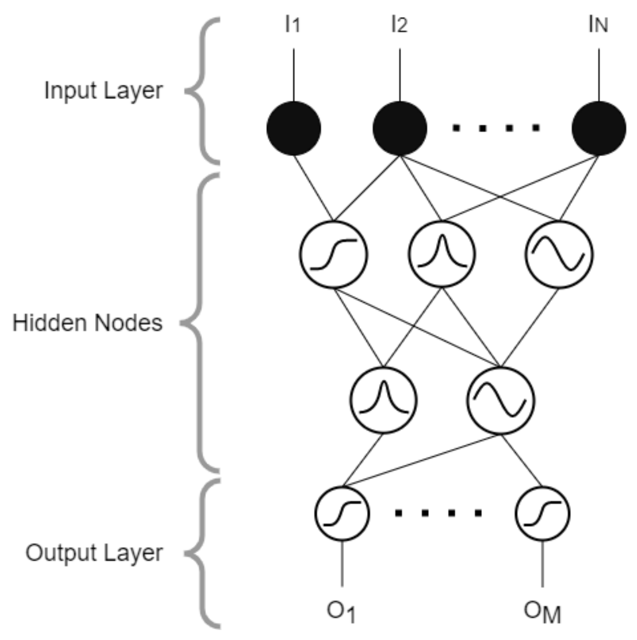

Figure 2.1: Example of a generic CPPN.

2.2.1 CPPN-NEAT

We've seen that novelty search mimics natural evolution in that it lacks an objective function and has a constant and open-ended drive toward complexity. The complexity of individuals in artificial systems is defined, in large part, by the genotype representation being used for the evolutionary algorithm. A fixed genotype representation has a ceiling to how complex an individual it can represent. Neuroevolution of augmenting topologies (NEAT) is a popular algorithm in QD and open-ended systems for its ability to create increasingly genetically complex individuals (Stanley & Miikkulainen, 2002). It was originally developed to evolve artificial neural networks (ANN's) to solve difficult problems, selecting actions based on sensory inputs. The algorithm starts with a population of small, low complexity networks and incrementally builds on them over a number of generations. The topology (nodes and connections) and weights of the network are evolved through mutation (add, remove, modify) and through crossover between two individuals. The algorithm maintains genetic diversity through speciation, separating the population into groups according to their similarity with one another, evolving them as separate populations. It is important to note that whilst NEAT guarantees increasingly complex genomes, it does not guarantee that the phenotypes generated exhibit a matched increase in complexity, which is required for effective OE.

CPPN-NEAT is a variant of the algorithm which evolves *compositional pattern producing networks* (Stanley, 2007) rather than typical ANN's. CPPN's differ from ANN's in that they allow for the use of multiple concurrent activation functions in the network topology. Whilst ANN's can only use one such function, CPPN's employ a large variety of them (e.g. sine, cosine, Gaussian, sigmoid, etc.). They are especially useful for the generation of visuals and aesthetic structures, as they exhibit a number of natural patterns such as repetition, repetition with variation, symmetry, and more (Stanley, 2007). Another desirable quality of CPPN's is that a sufficiently complex network is capable of encoding a phenotype of an infinite resolution, making them scalable for tasks such as building generation. Due to their

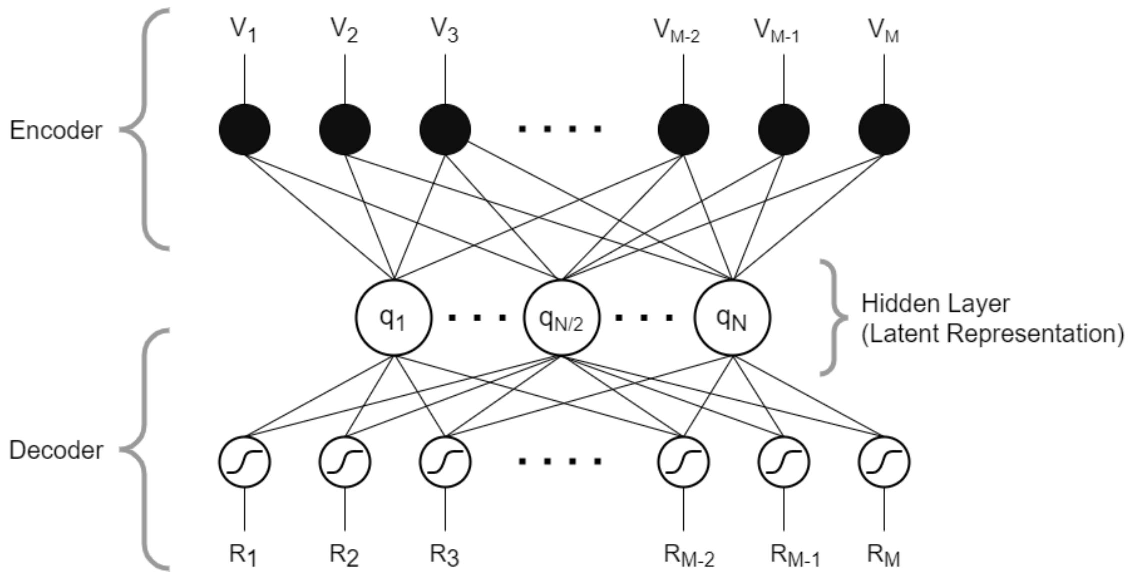

Figure 2.2: High-level topology of an autoencoder.

similarity to standard networks, they can be easily evolved using NEAT without any major modifications to the algorithm.

2.2.2 Autoencoders

Autoencoders (AE's) are a form of neural network used for principal component analysis (PCA). They are trained through unsupervised learning to compress an input data structure into a smaller representation, which is used to approximately reconstruct the original data (Hinton & Zemel, 1994). At a high level, autoencoders can be seen as two separate subnetworks (the encoder and decoder), separated by a hidden layer of neurons which is used as a latent state representation. The encoder's job is to map an input data structure to this latent representation by passing it through a series of hidden layers. The decoder uses the latent vector outputted by the encoder to reconstruct the original data as accurately as possible. For a vanilla autoencoder, the model is trained to minimize the reconstruction loss, which measures the difference between the original data and the reconstructed output. Once trained, the encoder can be used as a lossy compression tool, as the latent vector is effectively a more compact version of the original data.

In a vanilla autoencoder, the model is given the input data and attempts to reconstruct it as accurately as possible. This means the input and the target for the model are exactly the same. *Denoising autoencoders* (DAE's) differ from vanilla autoencoders, in that they are trained to reconstruct the original data from a corrupted version of the data. In this approach, the noise is added to the training set and the autoencoder is trained to map the noisy versions to their original unaltered state. Vincent et al. (2010) found that through this approach rather than a linear model resulted in a far more powerful tool for PCA (less prone to overfitting).

Other types of autoencoders have been introduced that build upon this foundation and produce models that do not easily overfit to the training set and that are better suited for generative systems. To achieve this goal, *variational autoencoders* (Kingma & Welling, 2019) differ from the simpler autoencoders mentioned above in that they attempt to regularize the latent space in training. A regularized latent space is far more desirable to a generative system, as points that are similar in the latent space are also similar in the original, decoded space. In the case of novelty search using latent vector data, this is crucial, as novel individuals in the latent space need to match up with novel individuals in the original space. VAE's accomplish this task by encoding the input as a probability distribution over the latent space (rather than a single data point in the latent space), which is sampled and used to reconstruct the original representation.

2.2.3 Expressive Range of Content Generators

The task of evaluating an autonomous generator was a crucial and challenging part of this project. The goal of evaluating a generator is to understand its capabilities, identify its strengths and weaknesses, and ultimately be able to make guarantees about the content it produces. Through this understanding, we can compare the generator with other systems, and the state of the art of the field can be clearly advanced. The difficulty of the task lies in identifying the qualities desired in the content that is to be generated. Furthermore, such systems are typically complex, sometimes lacking a clear objective (e.g: optimizing for novelty), and produce a large amount of data which needs to be evaluated. The content generated can also be affected by stochastic properties of the generator or by any humans involved in the generation process, further adding to the complexity of this task.

An informal approach might be to gather and subjectively evaluate examples of the generated content, and compliment them with some statistics about fitnesses and the generator's speed. Whilst this might provide some interesting insight to the generator, this approach does not give any information on the range/distribution of content generated and how this range is effected by changes to the generator's parameters (Smith & Whitehead, 2010). This information is critical as it helps refute the suspicion that any desirable content generated by the system is merely an exception and not the norm. It's also easy to see how this approach is infeasible, requiring the subjective evaluation of thousands, if not millions, of pieces of generated content (Shaker, Smith, & Yannakakis, 2016).

One method for addressing these shortcomings is by analyzing the *expressive range* of a PCG system in a top-down approach. This is a robust method for rigorously assessing the quality and diversity of the content generated by PCG systems (Smith & Whitehead, 2010). The expressive range can be thought of as some N-dimensional space, where each dimension is a different quantifiable metric of the content (Shaker et al., 2016), similar to the features space used in MAP-Elites. The feature space is used to visualize the frequency distribution of the content generated with respect to the defined dimensions. It is important to choose the appropriate metrics for this task, which should be as removed as possible from the input parameters of the system to avoid confirmatory results (Shaker et al., 2016). Once content is generated, the expressive range can be visualized in terms of any number of appropriate metrics for the domain in question. An easy approach to visualizing the expressive range is to plot the frequency distribution of two to three dimensions using a discrete histogram or heat map (Smith & Whitehead, 2010; Shaker et al., 2016).

Smith and Whitehead (2010) argue that an expressive system does not necessarily result in a creative system. Taking building generation as an example, a generator may be more than capable of generating a wide range of buildings across numerous dimensions. However, it can do so without necessarily understanding the meaning behind its expressivity, leaving it to humans to create the dimensions and assign meaning to its actions. In this case, the generator may be producing levels of a wide range as a result of the objective function and the dimensions used by humans to understand it. Therefore, they argue that whilst such a system is expressive, it is not actually a creative system as it does not comprehend or assign any creative meaning (e.g. novelty) to the individuals it creates. The expressive range is also heavily dependent on the features chosen for the analysis, meaning a generator can be deemed to behave creatively but not be expressive in terms of one or more dimensions being studied.

2.3 Related Material

This section covers some key examples of existing systems that exhibit some level of OE and IM, and were used as a foundation for the work done in this project. Each piece of work and its contribution are summarized in relation to the concepts mentioned in this chapter.

2.3.1 Evolving Diverse Robots

Existing research into evolving robot morphologies hold many similarities with the aims for this project. Most importantly, they seek to evolve a three-dimensional structure according to some objective function. They are also subjected to a set of constraints that govern the robot's movement and the environment it must traverse, much like constraints would to ensure feasible buildings. This subsection explores the work of Gravina, Liapis, and Yannakakis (2018) which combines novelty and surprise search to evolve soft robots and observes the individuals' resulting behavior. This example was also chosen as it follows the notions of IM and delivers a system that exhibits exploratory OE.

Gravina et al. (2018) represent the robots as a three-dimensional lattice of voxels, where each element encodes a material type, in the same way a biological creature is made up of various tissue forms. CPPN's are used to describe the lattices, which are evolved using NEAT and one of four fitness functions. Objective search evolves robots to travel as far as possible from the starting point to achieve a positive fitness. Novelty search using the K-nearest neighbors, surprise search, and a combination of the two were also tested as fitness functions for the generator. The algorithms are assessed using a set of measures that evaluate their quality. The efficiency of robots refers to the objective function described above, rewarding distance from the starting point. Robustness measures the number of individuals that are able to cover a threshold distances, giving a measure of the overall population quality. Structural variety was another measure used to illustrate the diversity of the feature space, comparing the number of filled voxels and bone voxels. The robots were tested using a set of resolutions ranging from 3x3x3 to 10x10x10 voxels.

Their results showed that combining novelty and surprise search together yields better results both in terms of efficiency and robustness. The difference between novelty and surprise search when used individually was not found to be significantly different. As one would expect, objective search evolved robots that were more similar to one another in terms of structure compared to the divergent methods. Larger lattice resolutions were found to yield more expressive morphologies, and the CPPN's were found to scale well in response, though CPU time was still found to increase to 88 hours for one evolutionary run. This paper's use of CPPN-NEAT and individual representation using lattices were a fundamental inspiration for the method of this project.

2.3.2 Open-Ended Generators

Whilst there doesn't seem to be any clear existing work on open-ended building generation for games such as Minecraft, there are a number of good examples of OE generators applied to other domains. This subsection outlines some notable examples of such generators, looking at how they achieve OE through IM and the case studies used to evaluate their implementation.

Liapis et al. (2011) introduced the deep learning novelty explorer (DeLeNoX), which is a system for autonomously creating artifacts in constrained search spaces according to an evolving definition of novelty. DeLeNoX follows Schmiduber's theory of creativity by characterizing novelty through a model/agent trained to identify high-level patterns in the data (i.e. compress into a smaller representation). To accomplish this task, DeLeNoX cycles between two phases; exploration and transformation. Novelty is calculated by compressing the phenotypes of the population into a latent vector using an autoencoder. Exploration phases use an evolutionary algorithm, in this case CPPN-NEAT, to evolve a set of populations using feasible-infeasible two population novelty search. Transformation takes the most novel individuals from exploration and uses them to retrain the autoencoder, thus shifting the search bias, which is dictated by the weights of the model. This change allows the system to search new parts of the search space that would have been unlikely to be visited with the original bias. The case study given in the paper is that of generating diverse two-dimensional arcade style spaceships whilst ensuring believability through a set of constraints. The results showed that DeLeNoX produces autoencoders which are better able to identify features in the data and produce more diverse individuals compared to simply exploring the search space without transformation. Through the transformation phases, the generator is ultimately able to more thoroughly search the problem space over time.

Grillotti and Cully (2021) implemented a system very similar to DeLeNoX (using the same two phase approach) with the goal of autonomously discovering robot behavior without hard-coding any behavioral descriptors. Behavioral descriptors are similar to the latent vectors of high-level features used in DeLeNoX, consisting of a low-dimensional latent vector (mapped by an autoencoder) to describe a behavior of the robot. The QD (exploration) phase consists of using any standard QD algorithm (MAP-Elites, CPPN-NEAT with novelty search, etc.) and the encoder model to evolve a container of individuals using their behavioral descriptors. The encoder-update phase (transformation) consists of updating the encoder model and novelty archive of latent vectors according to the latest novel data. The experiment results showed that the algorithm performs similarly to a traditional hard coded behavior descriptor approach in a set of three tasks; maze navigation, hexapod robot control and air-hockey. This algorithm differs from DeLeNoX in that it includes a mechanism to maximize fitness through QD, whereas DeLeNoX performs a simple novelty search and does not consider the quality explicitly. Hagg, Berns, Asteroth, Colton, and Bäck (2021) follow a similar approach to these two systems, using QD search on the latent representation found through VAE's to evolve and identify interesting two-dimensional shapes. Their findings highlight that directly evolving the latent representation of individuals is less effective at generating diverse content compared to using the latent space as a distance measure and evolving the individuals with a separate genotype.

It is clear that these generators are governed by IM. Each generator uses a self-defined definition of novelty, which is constantly evolving according to the content it generates. The reward function is universal and can be used with any generator setup provided it feeds it a one dimensional latent vector. The generators are not influenced by any external definitions

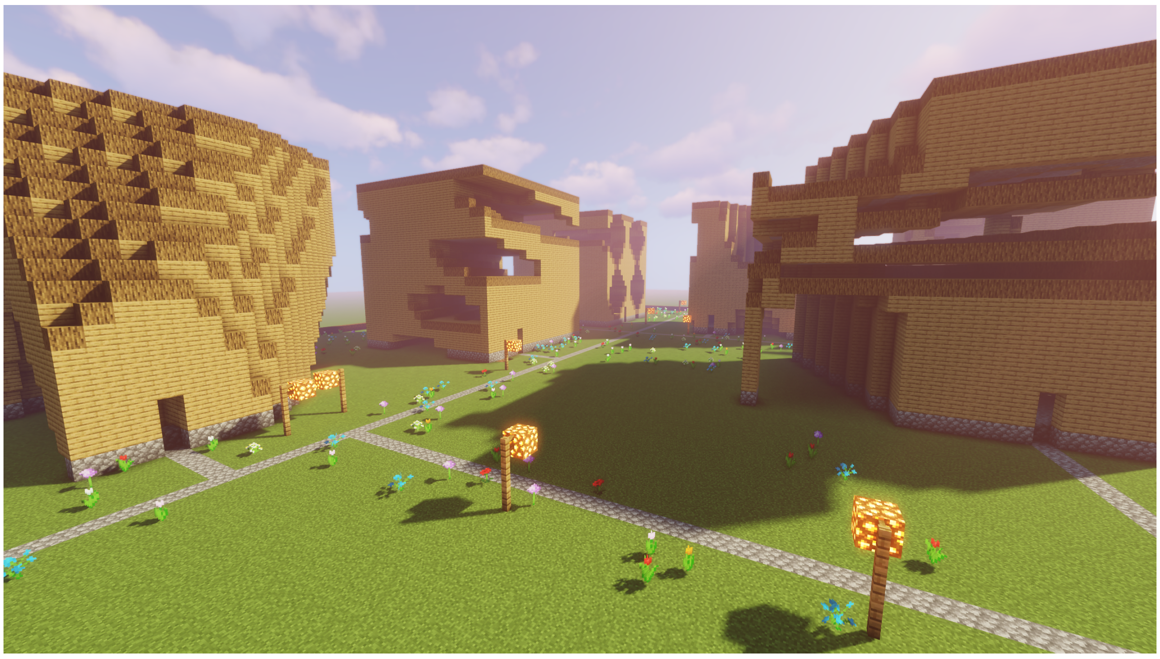

Figure 2.3: Example screenshot of buildings imported into Minecraft and arranged into a basic settlement.

of novelty, and do not use any data outside their internal components. It can be argued that depending on how one defines the meta-model, these systems exhibit all three types of OE (exploratory, expansive and transformative), which is discussed further in section 4.3. These systems, DeLeNoX in particular, formed the foundation for the method of this project, so the finer details on the overall implementation can be found in chapter 3.

2.3.3 PCG in Minecraft

Minecraft is a sandbox video game developed by Mojang and was released in 2009. The game consists of an infinite, procedurally generated world populated with resources and monsters. All structures in Minecraft are made up of blocks, or voxels, of different materials with different properties. The game has two modes, a survival mode where the players must harvest resources and survive against the elements and surrounding monsters, and a creative mode where the players have infinite resources and are free to use the game as a canvas for their imagination. As a sandbox game, Minecraft's gameplay is considered open-ended, with no linear objectives immediately presented to the player, affording a high level of freedom to their decision-making. For these reasons, Minecraft has become a very appealing domain for game AI and CC.

The generative design in Minecraft competition (GDMC) was introduced to the public with the aim of encouraging people to design AI programs for Minecraft that achieve human levels of performance in the domain of CC (Salge et al., 2018). The task presented by the competition is to develop a Minecraft settlement generator that produces functional settlements, is capable of adapting to any environment, evokes an interesting narrative to the viewer, and is aesthetically pleasing. Whilst the majority of the focus is on the settlement layout and composition, some attention must be paid to the method for generating the buildings that make up the settlements. In the competition's first year report (Salge et al., 2021), it's clear that the contestants did not focus on the evolution of creative buildings, opting for simple generative models. This project seeks to focus the creative effort of a generator on the structure of the buildings themselves, ignoring the design of the settlements for the time being (the building generator can be easily be paired with a settlement generator in the future).

Some existing works on building generators for Minecraft use Cellular automata (CA) to create organic, realistic buildings, which include the design of the interior space. Green, Salge, and Togelius (2019) use a constrained growth process to generate an ASCII map of the interior using one of three building size (small, long, large), and use CA to design the exterior walls of the building. Sudhakaran et al. (2021) extend the use of neural cellular automata (NCA) into 3D for creating Minecraft structures, including buildings, caves, and trees. Their approach allows them to create increasingly complex 3D structures with the ability to regenerate/repair themselves. They also applied their approach to generate moving structures such as flying machines and caterpillars using Minecraft's piston blocks.

The generation of the worlds themselves is also a focus of existing work. The default Minecraft world generator is handcrafted to generate vast landscapes consisting of different categories (forest, desert, etc.) and containing various structures such as caves, villages and more. However, the default generator cannot create new structures or styles on its own. Works such as World-GAN (Awiszus, Schubert, & Rosenhahn, 2021) attempt to address this shortcoming. It uses a 3D general adversarial network architecture to generate levels and block2vec token embeddings to create convincing examples of worlds with good variability. By editing the token representation, the generator is capable of creating worlds using different styles (e.g., change the style or ruins from a forest to a desert). However, the system is not optimized to create coherent, functioning structures, producing buildings and villages that are not entirely correct. The authors cite this pitfall as a promising area of future work and allows for the generator to be aligned better with the GDMC by allowing it to generate convincing settlements and structures in its landscapes.

Summary

In this chapter, we established the theoretical framework that will be used for the rest of the project. We started by giving an overview of CC and established our working definition of the concept in relation to games. We briefly covered creativity theory and described what it means for a system to achieve open-ended creativity, the different types of OE, as well as how they can be achieved. We followed this up with a basic definition of IM and why it's desirable for creative systems. We covered the field of PCG and the shift towards abandoning objective searches in favor of novelty. We briefly covered CPPN-NEAT and autoencoders which are important for the implementation of our system, and looked at methods for evaluation content generators. Finally, we summarized related works to establish the current state-of-the-art and justify our design decisions later in the project. In the next chapter, we give a detailed description of the design of our generator and how it relates to the concepts and existing approaches we've covered in this chapter.

Chapter 3

Method

Having established the objectives and background for this project, we can now take a look at the implementation of the generator. We based the fundamental design of the system off the DeLeNoX system (Liapis et al., 2011), which was chosen because it ticks many boxes with regard to OE and IM in content generators. Recall that the main objective of the system is to autonomously generate Minecraft buildings within a constrained search space according to the system's own evolving idea of novelty/creativity. DeLeNoX is tried and tested for this task, requiring minor adaptation to the core algorithm to generate 3D structures rather than 2D sprites. It's quite clearly also an intrinsically motivated system, with the generator not given any explicit or external reward to optimize other than its own inherent curiosity to find more interesting and complex content in the search space.

Following the original DeLeNoX method, the system alternates between phases of exploration and transformation, as seen in Figure 3.1. At the heart of both phases is the 3D autoencoder, which is trained to compress the generated structures into a one-dimensional latent vector of high-level features. In the exploration phase, the system uses CPPN-NEAT to evolve a set of populations of buildings according to the current novelty function. The autoencoder dictates the current novelty function, which calculates the average pairwise euclidean distance to the k-nearest neighbors in the latent space. In phases of transformation, the autoencoder is re-trained using a training set of the most novel individuals generated by the previous exploration phase/s. The new autoencoder has a modified search bias due to the re-training process and can better identify more complex high-level features as a result of its better compression accuracy. The transformation phase is critical for the system to exhibit OE, as it is that this point that the boundaries of the search space are affected.

In this chapter, we break down this project's implementation into its major components and describes how they work together to accomplish our objectives. First, we describe the domain representation and method for mapping one representation to another. This is followed by a description of our implementation of novelty search and how the generator handles infeasible individuals during evolution. We then dive into the transformation phase, which is outlined with an emphasis on the relationship between the autoencoder and novelty function, and the resulting impact on the OE of the generator. Finally, a detailed description of the exploration phase of DeLeNoX is given, looking at how CPPN-NEAT is used to evolve populations of buildings.

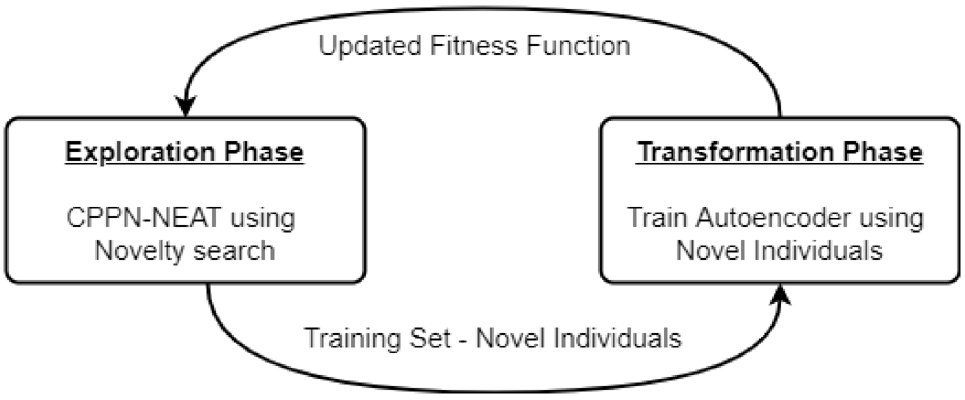
Updated Fitness Function
Exploration Phase
CPPN-NEAT using Novelty search
Transformation Phase
Train Autoencoder using Novel Individuals
Training Set - Novel Individuals

Figure 3.1: High-level overview of DeLeNoX

3.1 Domain Representation

The design of in-game structures in Minecraft largely dictated the design of the phenotype. Minecraft represents almost all in-game content as a set of voxels (i.e., three-dimensional pixels), which each contain material data for that location. Therefore, we represent the generated contents' phenotype as a three-dimensional array of elements, where each location encodes the material ID for that location. This direct representation allows us to easily import the generated content into Minecraft using programs such as MCEdit without any modifications to the data structure. The resolution chosen for the phenotype was 20×20×20 voxels, as this was deemed the sweet spot between computational efficiency and representational power. This trade off had to be made because lattice resolution has one of the biggest impacts on the computational overhead during evolution with the current implementation. For simplicity, each voxel may contain one of the following five materials; outdoor air, indoor air, floor, walls, and ceiling. Since a voxel can contain only one material at any given time, the material data is stored as a one-hot encoded vector, effectively creating a 20x20x20 array of 5 channels (one for each material).

3.1.1 Genotype Representation

One of the critical design decisions when creating a search-based generator is the representation for the genotype (Yannakakis & Togelius, 2018). With higher levels of abstraction, the representation becomes more compact and computationally efficient, but restricts the search space of possible individuals. On the contrary, a more direct low-level representation of the data ensures most if not all the possible individuals can be represented, but comes at the cost of vastly higher computational complexity. In the context of Minecraft buildings, a direct approach could be to treat the building genome as a list of voxels and use crossover and mutation on parts of the list, but it's easy to see that for any meaningful building size this would quickly become infeasible. This design decision is also critical to the generator's ability to exhibit OE, as a genotype which is fixed and capped in terms of representational power prevents open-ended evolution.

Our approach was to represent buildings' genotype using compositional pattern producing networks (CPPN's) where each network takes as input the XYZ coordinates of a voxel and outputs a Boolean value indicating the presence of a voxel. This is similar to the approaches taken by Gravina et al. (2018) and Liapis et al. (2011), which both use

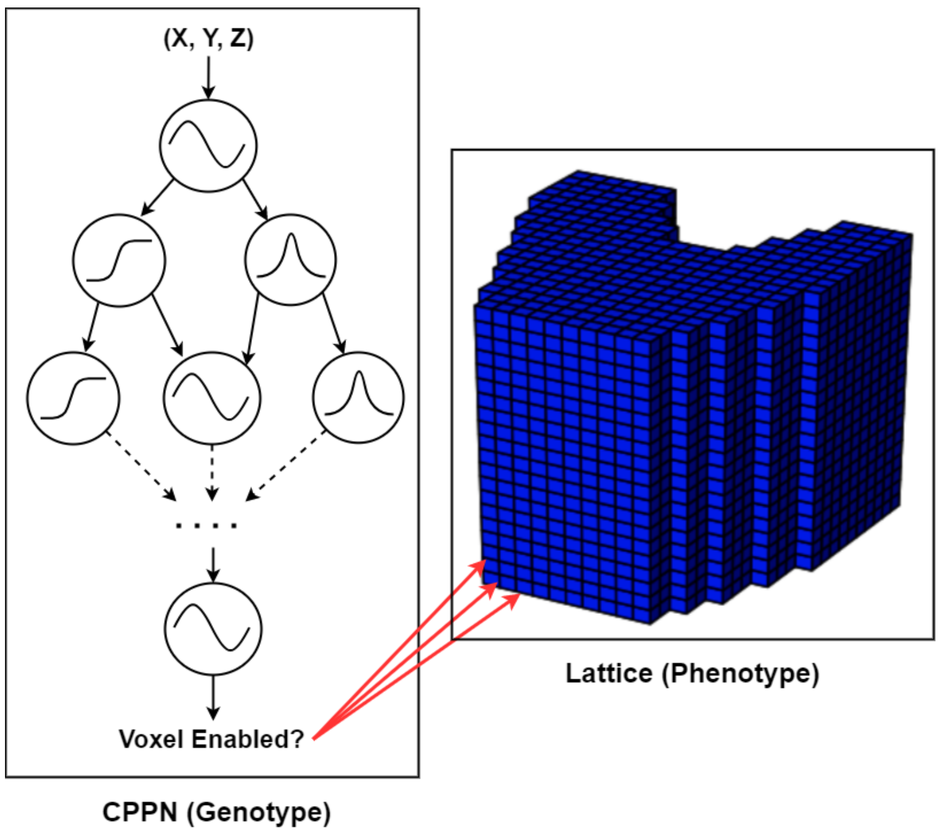

Figure 3.2: Genotype-to-Phenotype mapping

CPPN's and which are mapped to a pixel/voxel based phenotype. CPPN's present the ideal balance between strong representational power (theoretically able to represent any output) and computational efficiency through abstraction, demonstrating strong scalability and the ability to encode any resolution in the phenotype space (Stanley, 2007). As we mentioned in section 3.4, CPPN's can be evolved using a slightly modified version of the NEAT algorithm. This is important as NEAT guarantees increasing genetic complexity over time, which is a critical component for our goal of an open-ended generator.

We elected to use the NEAT-Python library for the implementation of the CPPN's due to its ease of use and extensive documentation. Whilst the library was originally built to use standard ANN's, CPPN's can be generated by modifying its configuration file to allow multiple concurrent activation functions in a network at any given time. For the time being, we use three different activation functions in the topology of the CPPN's; a sine function bounded between 0 and 1, a sigmoid function, and a Gaussian function, all of which were provided by the library. Mapping the CPPN's to the phenotype is the most computationally expensive function in the current implementation of the system. It involves iterating over the entire list of all 8,000 coordinates of a lattice and filling a three-dimensional array with the output of the network for each coordinate.

3.1.2 Latent Representation

The latent representation is critical for DeLeNoX as it is used to calculate an individual's novelty score. The latent vector can be seen as a one-dimensional compressed representation

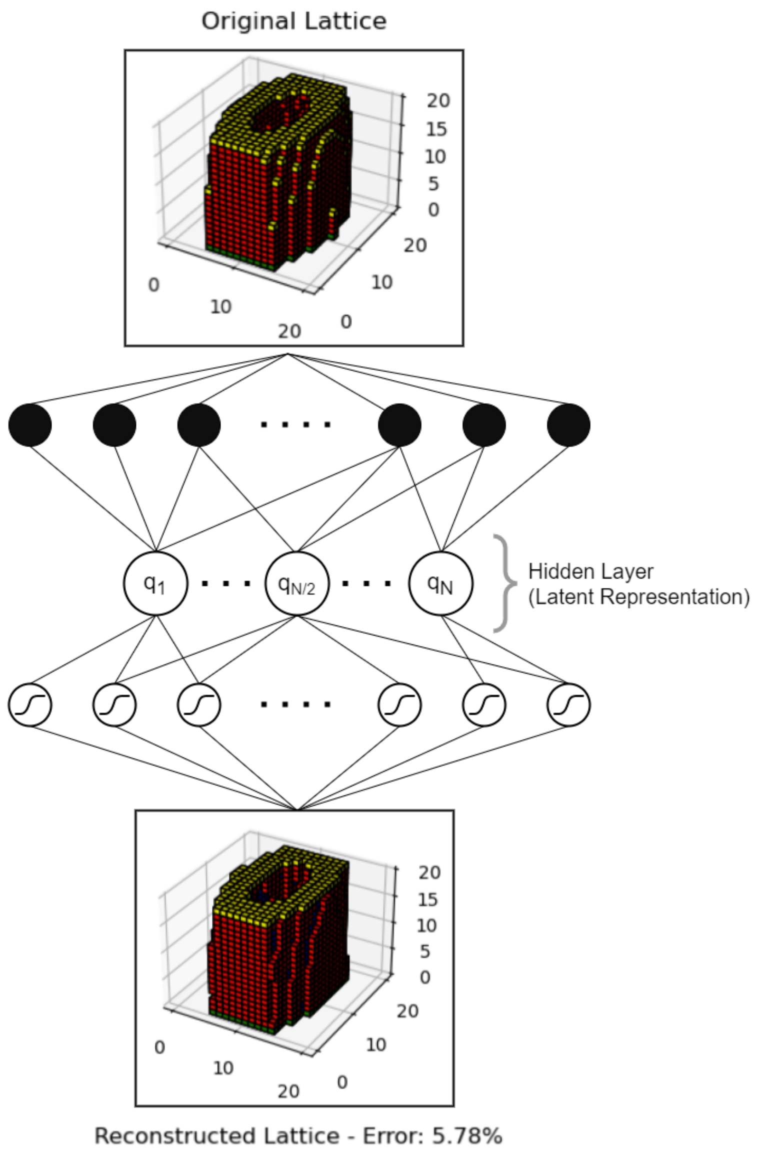

Figure 3.3: Building compression and reconstruction using an Autoencoder.

of the phenotype, containing a set of high-level features observed in the lattice. This system maps the phenotype to the latent space through the use of a 3D convolutional autoencoder, which is described further in section 3.3. The size of the latent vector is important as it has a direct effect on the compression accuracy of the autoencoder. Larger latent vectors are theoretically more capable of storing information about the original lattice, but this comes at the cost of computational complexity. Since the system uses these latent vectors for novelty search, computational efficiency should not be ignored as it can drastically effect training time.

For this project, we implemented the autoencoder model using Keras/Tensor-Flow, with an optional "denoising" mode available for the experiments. The autoencoder's method for compressing and reconstructing buildings is very similar to how one would go about compressing an RGB image. It treats the lattice as a 3D array consisting of five separate channels, one for each possible material in the phenotype, which are taken as input by the

Layer Type	Layer Size	Activation Function	Output Shape
Input Layer	N/A	N/A	(20, 20, 20, 5)
3D Convolution	(3, 3, 3)	Relu	(18, 18, 18, 64)
3D Max Pooling	(2, 2, 2)	N/A	(9, 9, 9, 64)
3D Convolution	(3, 3, 3)	Relu	(8, 8, 8, 128)
3D Max Pooling	(2, 2, 2)	N/A	(4, 4, 4, 128)
3D Convolution	(3, 3, 3)	Relu	(2, 2, 2, 256)
3D Max Pooling	(2, 2, 2)	N/A	(1, 1, 1, 256)
Flatten	N/A	N/A	(256)

Table 3.1: Layers making up the Encoder Model (mirrored for the Decoder).

encoder model. We included the material data in the compression mechanism because we wanted material placement to have an impact on novelty. The encoder consists of a series of 3D convolution layers and down-sampling layers which reduce the dimensions of the data to the shape of the latent vector, as seen in table 3.1. During evolution, the process stops here because we do not need the decoder at any point during evolution. During training, the decoder is used to reconstruct the lattice from the latent vector to calculate the information loss and adjust the weights of the network accordingly.

3.2 Novelty Search

The use of novelty search is important for this generator for several reasons. Divergent searches like novelty search satisfy the following properties that are desirable for intrinsically motivated systems (Guckelsberger, 2020). It is only reliant on raw sensory data (elements of the latent vector) and is not associated with any external reward or desirable outcome. The function also works independent of the semantics of the latent vector, and is generic, meaning it can be used interchangeably with any system. The search for novelty is also a central part of the definition of OE (Lehman & Stanley, 2011; Banzhaf et al., 2016), which by definition cannot be exhibited in a system which is optimized to converge on a set of objectives.

This system defines novelty as the average euclidean distance of an individual's latent vector to its K nearest neighbors. The use of the latent space for the distance function places the autoencoder at the heart of this system's definition of novelty. In other words, we calculate the distance between two individuals based on the high-level features identified by its model. This means we intrinsically define the search for novelty based off the system's current autoencoder and data used to train it. As we will see in section 3.3, this definition of novelty can be transformed, allowing the system to produce expansive novelty (T. Taylor, 2019), on top of the exploratory novelty afforded by a static novelty function.

$$n(i) = \frac{1}{k} \sum_{m=1}^{k} \sqrt{\sum_{n=1}^{N} [q_n(i) - q_n(\mu_m)]^2} \tag{3.1}$$

We follow the implementation for novelty search used by DeLeNoX. The system maintains a novelty archive for each population of buildings it evolves. At the end of each generation during exploration, the three most novel individuals are inserted into the novelty archive, provided that an exact copy does not already exist. The archive needs to be updated whenever the autoencoder is retrained, as their points in the latent space are modified during the transformation process. When calculating the novelty of an individual, we first map all the CPPN's of the population to their corresponding lattices and compress them into latent vectors using the encoder. For each individual, we calculate the distance to every other latent vector in the population (size N) and take the average of the shortest K distances, as seen in equation 3.1.

Infeasible Individuals

Currently, we do not apply any explicit constraint functions on the properties of the generated buildings, other than that they must contain space for an entrance. Other implicit constraints, such as preventing empty lattices or floating voxels, are addressed by the repair functions during the generation process. Since we do not apply other explicit constraints, and the one in use does not have a distance measure to the feasibility threshold, our version of novelty search does not use the feasible-infeasible two-population (FI-2POP) approach (Kimbrough, Koehler, Lu, & Wood, 2008) for handling infeasible individuals. In our approach, any infeasible (empty) lattices are discarded and replaced with new individuals using NEAT's reproduction strategy. The building properties described in Section 3.4.2 are a good starting point for adding more building constraints to the generator and FI-2POP in the future.

3.3 Transformation Phase

The transformation phase is the most influential component of the DeLeNoX algorithm due to its effect on the autoencoder and novelty function, which dictate the behavior of the system in the search space. This phase is responsible for allowing the system to exhibit different forms of OE, by ensuring its definition for novelty is up-to-date with the level of current level of complexity in the population. We consider the autoencoder as the scientific model of the system as defined by Banzhaf et al. (2016), as it is used to understand the underlying features in the population, and requires correction to prevent poor performance. Therefore, the transformation phase can be seen as a modification to the system's model, satisfying the definition of expansive OE (Banzhaf et al., 2016; T. Taylor, 2019) and allowing the system to open up new regions in the current dimensions of the search space. Without transformation, the system would be bound by the original weights of the autoencoder and their restrictions in the search space, preventing the system from exhibiting anything other than exploratory OE, dismantling the effectiveness of the system's creativity.

This makes the autoencoder a "novelty critic" of sorts, as its through the values that it assigns to an individual's latent vector that the distance function calculates their novelty score. The transformation phase shifts the weights of the autoencoder such that its interpretation of an individual's novelty evolves from one iteration to the next, thus shifting its biases and prioritizing other areas of the search space. This new autoencoder is saved and used as a basis for the next iteration's exploration phase, thus completing the current iteration of the algorithm.

3.3.1 Training Method

The retraining process for the autoencoder is straightforward. A training set of novel individuals is assembled from the final populations of the previous exploration phase/s.

This provides the new model with a set of individuals that the previous autoencoder/s identified as interesting, and provides the level of complexity that the new model must learn to compress. The retraining is done from scratch, meaning for each transformation phase, a new model is randomly initialized with the architecture seen in Table 3.1 and trained on the given training set. The rest of this section breaks down the different approaches that may be taken for the transformation phase, with respect to what training set is used for re-training and the type of autoencoders that may be used for the novelty function.

There are a number of approaches that can be taken to the transformation phase in terms of what training set is used for re-training. The training data has a substantial impact on the resulting model, and as a result the distance function and its biases, which will dictate the generator's interpretation of novelty for the next iteration. Whilst the generator is guaranteed to generate more complex genomes through NEAT, the complexity of the high-level features of their phenotypes is not guaranteed. Supplying the model with the most complex and novel individuals observed so far ensures that these features are learned and considered when assessing the next iteration of genomes. Below are the four options that were implemented for this project:

1. **No transformation phases:** in this basic approach, the transformation phase is abandoned entirely and the same autoencoder is used across all the iterations of DeLeNoX. This results in a single long search by NEAT in the search space bound by the biases original autoencoder (i.e., the novelty function remains constant).

2. **Randomly initialized autoencoder:** in this approach, the transformation phase is used to create a new autoencoder, however the autoencoder is not trained to identify the high level features of the input lattices. This means that the autoencoders do not accurately compress the data, producing noisy latent representations.

3. **Trained autoencoder using the latest training set:** in this approach, the autoencoder is trained on the latest training set only, with previous training sets being ignored. This creates an autoencoder that is optimized to compress lattices at the current level of complexity.

4. **Trained autoencoder using the entity history of past training sets:** this approach trains the new autoencoder, however this time using every training set generated in exploration by DeLeNoX so far. This should create an autoencoder that is optimized to compress lattices with all the levels of complexity seen so far.

5. **Trained autoencoder using the novelty archives:** finally, in this approach rather than training the model on the most novel individuals from the final generation of each population, the model is trained on a super set of all the populations' novelty archives. This creates an autoencoder that should perform similarly to one trained on the entire history of training sets.

3.3.2 Autoencoder Type

Aside from varying the training data and method during the transformation phase, we can also vary the type of autoencoder used to compress the data. The performance of the autoencoder is critical for the entire generator, as a model which cannot identify high-level features of buildings produces noisy latent vectors which cannot be used to meaningfully assess the novelty of an individual. There exist a variety of architectures and approaches for implementing autoencoders for such a task, but for simplicity's sake this project implemented the following two approaches:

1. **Vanilla Autoencoder:** in this approach a standard, 3D convolutional autoencoder described in Section 3.1.2 is trained to compress the lattices using the unaltered voxel data of each individual. This is the baseline approach, which can be improved upon with more complex models.

2. **Denoising Autoencoder:** rather than providing the autoencoder the original voxel data, the autoencoder is trained to compress and denoise a corrupted copy of the data which could in theory produce a more robust model (Vincent et al., 2010) with a lower reconstruction error. For this project, negative noise is added to each lattice by disabling a voxel with 2.5% probability. Other noise methods were also included such as positive noise (where voxels are randomly added) and bit-flip noise (where voxels are randomly disabled/enabled).

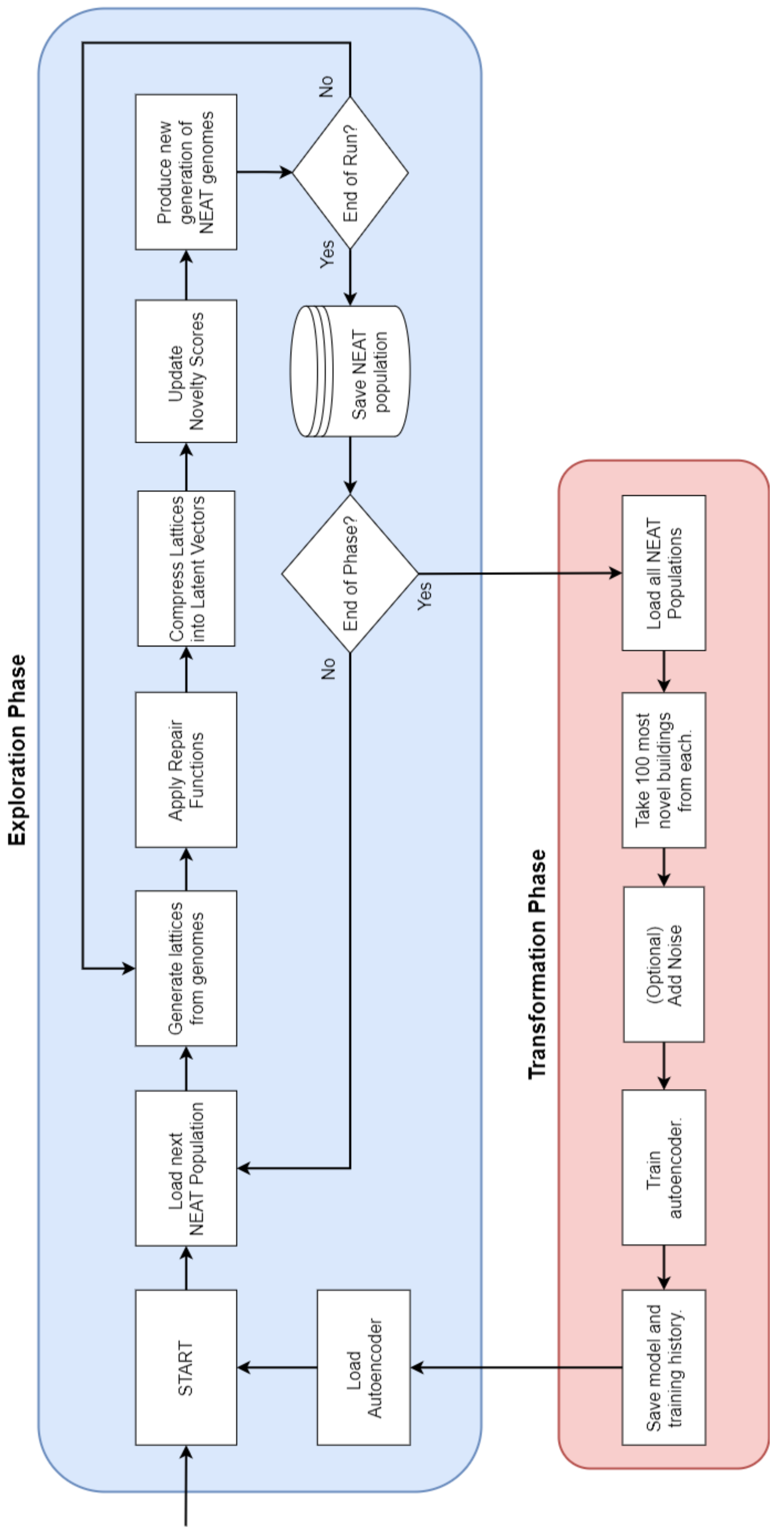
Exploration Phase
START
Load next NEAT Population
Generate lattices from genomes
Apply Repair Functions
Compress Lattices into Latent Vectors
Update Novelty Scores
Produce new generation of NEAT genomes
End of Run?
No
Yes
Save NEAT population
End of Phase?
No
Yes
Load Autoencoder
Transformation Phase
Load all NEAT Populations
Take 100 most novel buildings from each.
(Optional) Add Noise
Train autoencoder.
Save model and training history.

Figure 3.4: DeLeNoX control flow diagram.

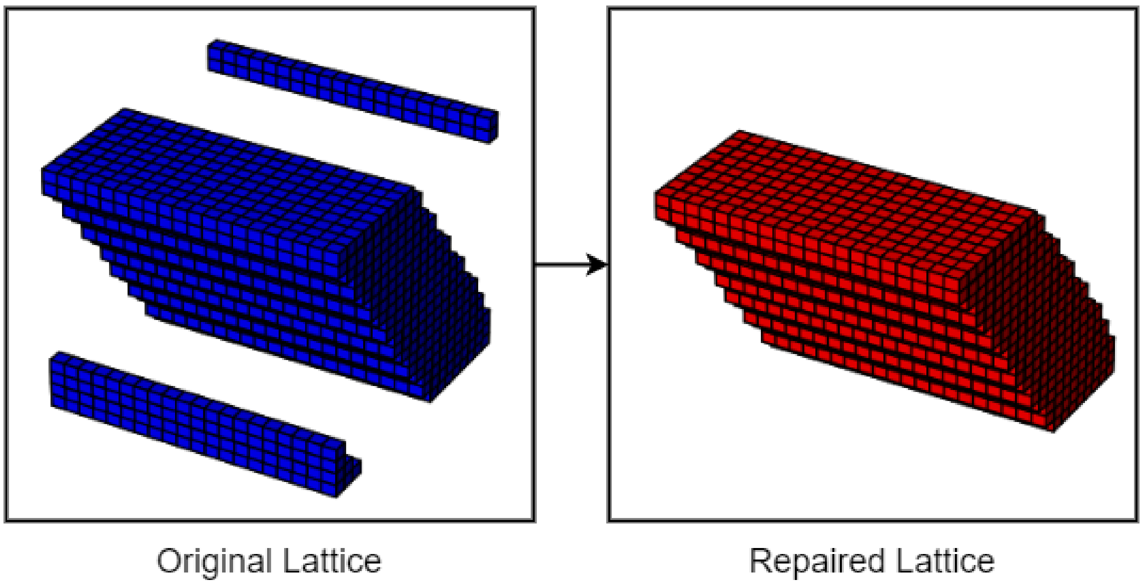

Figure 3.5: Example of flood-fill for lattice repair.

3.4 Exploration Phase

The exploration phase contains the majority of the computational effort for the DeLeNoX generator. The phase consists of running NEAT for a fixed number of generations separately for all the populations of buildings being maintained for the experiment. Buildings are evolved to maximize the current measure of novelty through the use of the current autoencoder. This section dives into the inner-workings of how the exploration loop in NEAT is implemented for this task and outlines the pipeline of functions each individual goes through to go from the genotype to the latent vector used to calculate its novelty.

The NEAT-Python library was used as a foundation for this implementation of NEAT due to its good documentation and ease of use. The library is very straightforward to use, only requiring a configuration file and a fitness function in most cases, whilst taking care of the back-end evolution of the networks itself (selection, mutation etc.). The configuration file contains all the parameters used for the CPPN's and their evolution. The most notable of these parameters are the mutation probabilities (probability to add/remove/modify nodes and connections to the network), the initial starting size of the networks, and the different activation functions available for each node. This project extends some functionality of the library to be more computationally efficient and to tie in better with the DeLeNoX architecture. This is depicted in Figure 3.4, which depicts the control flow of the generator when assessing the individuals of the current generation.

A run of NEAT starts by loading the latest autoencoder that will be used for evaluating the population for this phase. If this is the first iteration of the algorithm, a seed autoencoder and set of randomly initialized populations are used to kick-start evolution and compression. Otherwise, the current population of genomes is loaded from the disk. A pool of python CPU processes is (optionally) initialized for later use in the fitness computation, allowing for much greater computational efficiency than when running in single-threaded mode.

First, the pool of processes is used to generate the lattices for each genome using the approach described in Section 3.1.1. Next, the lattices are repaired using the pipeline of functions described in Section 3.4.1. Any infeasible individuals identified by the repair

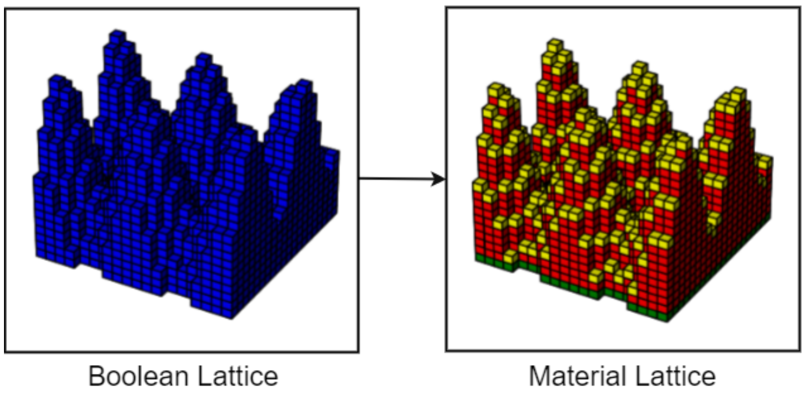

Figure 3.6: Assigning materials to each voxel in the lattice

functions are handled separately, as described in Section 3.2. The main thread retrieves the updated population of lattices, adds noise if necessary, and compresses them using the current autoencoder (using the machine's GPU if available). The process pool is used again to calculate the novelty scores of the compressed individuals using the approach seen in Section 3.2, and their genomes' fitness is updated. Finally, the NEAT-Python library handles the reproduction of genomes for the next generation of the population, following the parameters set in the configuration file. At the end of each generation, a dictionary of NEAT metrics is updated for future analysis, containing information on the average network complexity, archive size, species count, and number of infeasible individuals generated. If this is the final generation of the current population run, the population and NEAT metrics are saved to disk and the next population is loaded. If this is the final population for this phase, the generator moves on to the transformation phase.

3.4.1 Repair Functions

Since there is no guarantee that the population of individuals will conform to the basic properties of buildings, they need to be subjected to a set of repair functions. For this project, a small set of simple repair functions were implemented, though this can be extended as desired. These functions are applied to each building lattice in the order presented below, resulting in a population of repaired individuals which are compressed for the novelty computation.

Remove Floating Voxels

The first repair function of the pipeline works by repeatedly applying a three-dimensional flood-fill algorithm starting from all the active floor voxels. The flood-fill works by iteratively visiting a voxel and adding all its neighbors to the "pending" stack if the current voxel is enabled. The algorithm works its way through the stack until no neighbors are left to visit, and the floor voxels have been added. Any enabled voxels visited by the flood fill are kept enabled, whereas all other voxels are disabled, as this means they are not in direct contact with any grounded structure. This repair function satisfies the implicit constraint that no parts of a building may be free-floating.

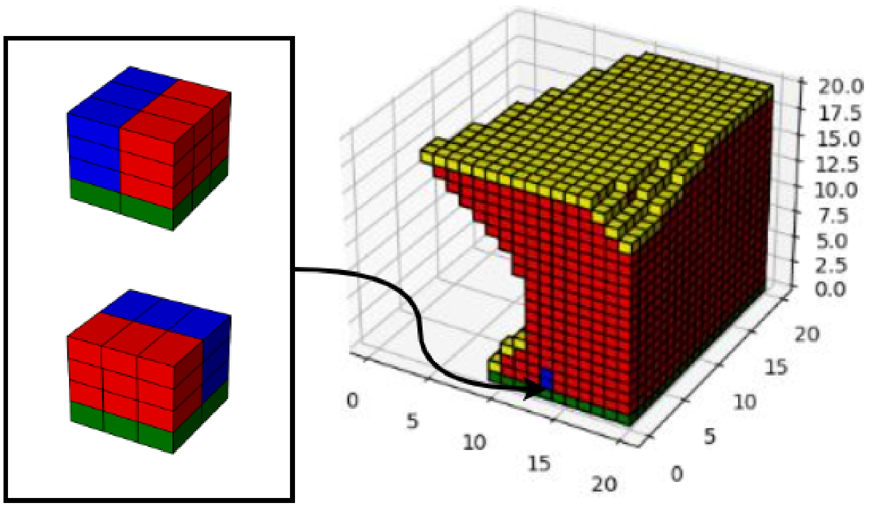

Figure 3.7: Searching for a suitable entrance to a structure.

Keep Largest Structure/s

Next, the flood fill algorithm is used once again to identify any separate, isolated structures in the lattice left behind after the previous function. If one structure is found, it is kept and the lattice is left unaltered. If multiple structures are found, there are several approaches that can be taken. The approach used for this project was to simply keep the largest structure found and discard any other structures present. This was chosen both for simplicity and due to the low resolution of the phenotype being better suited for one structure. Another approach could be to keep any structures that contain a minimum number of voxels, or encompass a minimum bounding box size. This would allow for multiple sub-structures that can still contain a meaningful level of detail if the resolution of the phenotype, and the minimum building size are high enough.

Identify Materials

Since the CPPN's only output the presence of a voxel (true/false), a third function is used to identify the material for each voxel based on their position in the structure. Floor (or ground) voxels are simply any voxel which are present on the plane $Y = 0$, and are used for the flood-fill algorithms seen in the previous two functions. Roof voxels are any voxel which have an "exterior air", or disabled voxel, above them. Wall voxels are any non-roof and non-floor voxels that are neighbored with an exterior air voxel. Finally, the interior air voxels are any of the remaining voxels that have not been assigned a material yet, and make up the interior volume of the building. The material is represented as a one hot encoded vector, as described in section 3.1.

Place Entrance

The final repair function in the pipeline was to identify a region in the structure to place the building's entrance/exit. To accomplish this task, the lower region of the building $(0 < y < 4)$ was iterated over, and the structure was compared with one of four possible templates (entrance facing north, south, east, or west). Figure 3.7 shows two of the entrance templates which consist of a 4x2x2 sub structure of floor, wall voxels followed by interior

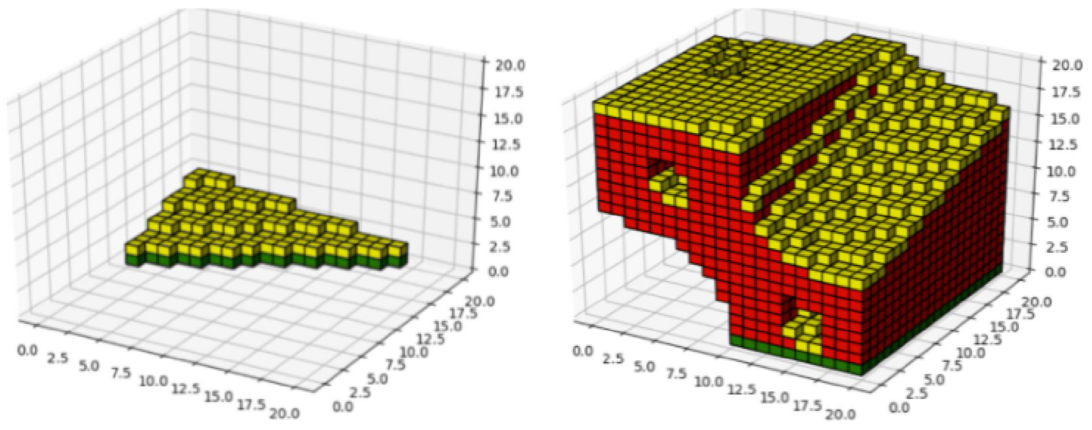

Figure 3.8: Comparison between low and high ratios of surface area compared to the bounding box, respectively.

space to ensure the door frame leads to an interior space that is accessible. If no region of the building contains one of the templates (and therefore cannot house an entrance), the building is flagged as infeasible accordingly and the original lattice is returned. If a region is found that matches on of the four templates, a doorway two voxels high (consisting of interior space) is added to the center of the region.

3.4.2 Building Properties

In order to get a better quantifiable understanding of the content generated, we implemented some structural property functions to analyze the buildings. In its current state, the generator does not use these during evolution, rather they are used after the content is generated for the expressive analysis in the evaluation method (section 4.2.1). However, these properties can be used as constraints in the future, though care must be taken to identify appropriate thresholds to ensure the generator is not over or under restricted. The implementation of these functions draws from similar work done by (Liapis, 2016), who present several methods for visually evaluating 2D spaceship sprites. In this section, we take a brief look at the building property functions implemented so far.

Bounding Box

Before looking at any meaningful properties of structures, we need to identify their bounding box volume to have a basic understanding of their size and location within the lattice. We represent the bounding box using six bounds, an upper and lower bound for each axis. To find the lower and upper bounds, we iterate over each axis and identify the smallest and largest coordinates containing an enabled, solid voxel. Using these bounds we calculate the width, depth, and height of the bounding box, as well as its area These properties allow the generator to identify the rough size of the structures, and could be easily used as a constraint for minimum or maximum desired size. The bounding box forms the foundation for the first two properties described in this section.

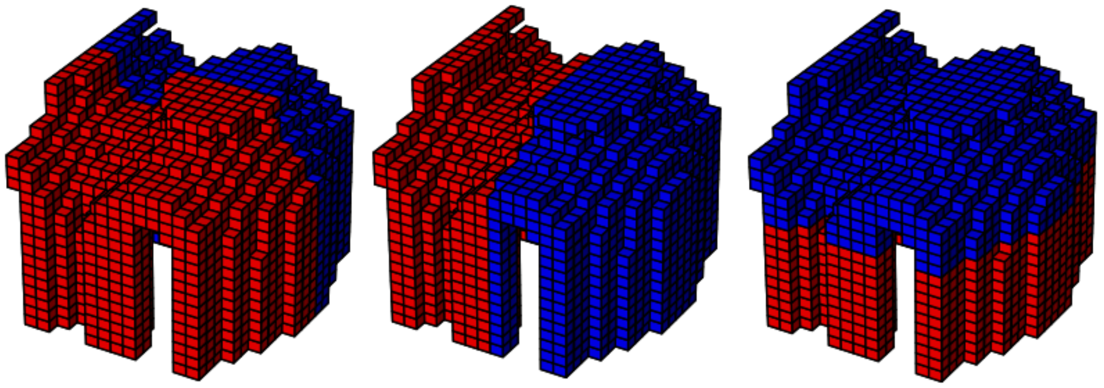

Figure 3.9: The three lines of symmetry used to analyze structures.

Surface Area

In this first property, we use the bounding box data to calculate the normalized surface area of a lattice. First, we iterate over the entire list of voxels in the lattice and store the number of locations with a solid material (floor, wall, roof), finding its absolute surface area. Using the bounds of the structure, we calculate the area of its bounding box by finding its width, depth, and height. We then divide the absolute surface area by the bounding box area to effectively turning it into a ratio of lattice surface area to bounding box area, which is visible in equation 3.2.

$$SurfaceArea = \frac{1}{Area_{BBox}} \times \sum_{i=0}^{8000} solid(i) \tag{3.2}$$

By normalizing the surface area according to the bounding box, we remove the overall building size from the equation, which allows us to get an insight into the complexity of the buildings surface. A structure with a higher surface area compared to its bounding box area has, at least, a more complex shape than a cube. On the other hand, a normalized surface area below 1 indicates the lattice is small with respect to its bounding box, or of simple shape. Figure 3.8 illustrates examples of lattices with surface area ratios below and above 1 respectively, clearly exhibiting the impact of this property.

Symmetry

The positive relationship between symmetry and aesthetically pleasing art has been studied extensively in the past (al Rifaie, Ursyn, Zimmer, & Javid, 2017). Existing work for similar generators have integrated symmetry into the generation process. Gravina, Liapis, and Yannakakis (2017); Gravina et al. (2018) included the distance to the center line for each coordinate as input to the CPPN's when evolving soft robots to promote symmetry in the content. In DeLeNoX, Liapis et al. (2011) generated half of the spaceship sprites and mirrored them across the center line to explicitly add symmetry. In this system, we do not explicitly add symmetry using these approaches to give the generator equal freedom to explore asymmetrical structures.

We implement symmetry as the average symmetry in the horizontal, vertical, and depth axes. We calculate symmetry in each axis by splitting the lattice into two and counting the number of times a voxel matches with its mirrored counterpart across the center line,

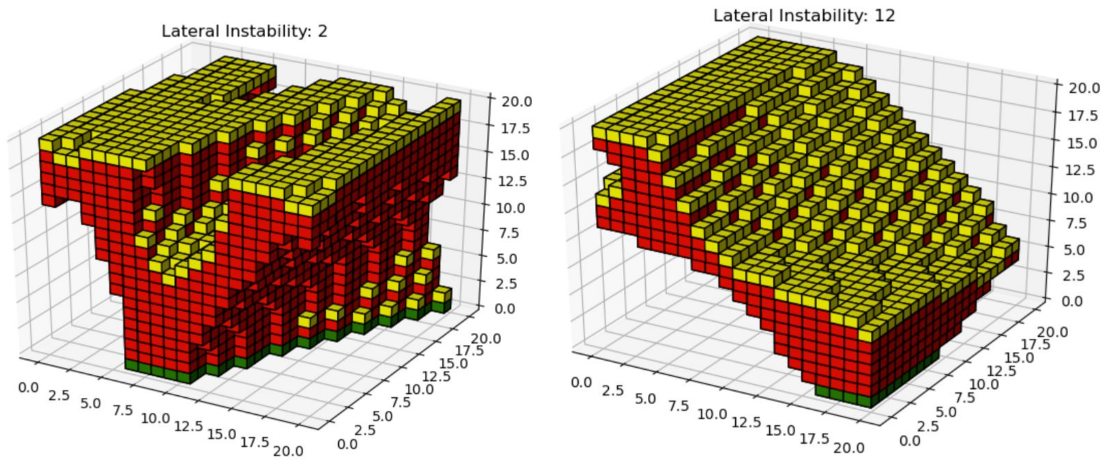

Figure 3.10: Laterally stable and unstable buildings.

which can be seen in figure 3.9. The limits of the bounding box are used to identify the line of symmetry for each axis. We consider a pair of voxels symmetrical if they are both solid or both air (binary comparison), meaning we do not take voxel material into account for symmetry. The symmetry score is normalized between 0 and 1 based on the maximum possible number of symmetrical voxels (4000 per axis, 12000 total).

Lateral Instability

Finally, we implement a property to analyze the structural feasibility of the buildings generated by calculating their instability. For this task, we define lateral instability as how unstable a given structure on the horizontal (XZ) plane. To calculate this value, we first need to calculate the building's center of mass (CM), and the center of mass of its floor plan (FCM). For the sake of simplicity, we assign the same mass to all solid materials, creating a boolean copy of the lattice containing only solid voxels. Using the center of mass function provided by the SciPy library, we identify the coordinates for the center of mass for the overall structure and its floor plan. Equation 3.3 shows the method for calculating the lateral instability using these coordinates. The result is the euclidean distance between the center of mass coordinates on the XZ plane, i.e., this measure does not account for the vertical stability of the structure. We chose to ignore the vertical stability for the time being as it would require a more detailed approach to the masses of each voxel and assumptions about building foundations, which were outside the scope of this project.

$$Instability = \sqrt{(X_{FCM} - X_{CM})^2 + (Z_{FCM} - Z_{CM})^2} \tag{3.3}$$

In figure 3.10, we can see two examples of buildings with low and high levels of lateral instability, respectively. Highly unstable buildings tend to have a small, off-centered floor plan with respect to the rest of the structure. On the contrary, stable structures tend to have a floor plan that follows a similar size and shape as the overall building. In the future, we could extend this property to provide a more low-level look at instability of a structure on the voxel level (e.g., by using a flood fill to assign an instability value to each voxel), though this would come at the cost of computational complexity. This would allow for the implementation of a repair function to address any unstable regions of the lattice by adding supports to unstable regions, and could be used as a quick way of salvaging infeasible individuals.

Summary

In this chapter, we gave a detailed description of the implementation of our building generator. We started by giving an overview of the DeLeNoX algorithm and how its two-phase approach is critical for exhibiting OE. We moved on to cover the domain representation, the mapping between the genotype and phenotype, and the latent representation used to calculate an individual's novelty. Next, we described our implementation of novelty search, and how infeasible individuals are handled by the system. After establishing these critical concepts, we moved on to describe the transformation phase and the different approaches to retraining the autoencoder. Finally, we covered the exploration phase, specifically on how CPPN-NEAT, and a set of repair and analysis functions, are used to evolve and evaluate a population of networks. In the next chapter, we use the system we've described for a set of experiments and evaluate our approach according to a set of performance metrics.

Chapter 4

Evaluation

The testing and evaluation phase of the project was the most challenging and critical part of the project. In section 4.2.1, we covered the challenges of analyzing the capabilities of a generator quantitatively and was used as the starting point for the approach to the system evaluation. The ultimate goal of the evaluation is to assess the creative and expressive capabilities of the generator with respect to OE, IM and QD. This chapter dives into the testing methodology and evaluation strategy used to assess the system with respect to the objectives and research questions of the project.

4.1 Testing Methodology

There are numerous interesting parameters and components that can be experimented with for testing the generator. These can range from experimenting with the parameters of NEAT that influence the exploration from a genetic standpoint, to altering the architecture of the autoencoder model (such as using variational autoencoders), or varying the resolution of the latent vector or building lattices. The issue for testing all these potential components is that in its current implementation, the generator takes a substantial amount of time to train and generate content. In section 5.2 we spend some time outlining some potential additions to the testing method in the future to more exhaustively cover all the potential areas of interest in the system.

For the time being, we identified the configuration of the transformation phase as the most interesting area to experiment with due to its pivotal role in calculating the novelty of generated content. Recall that in section 3.3 we covered the different potential implementations for the transformation phase. Table 4.1 depicts the 5 groups of configurations for the transformation phase that were described and evaluated. Each experiment consists of a single run of 10 iterations (each iteration consists of one phase of exploration and transformation). The experiments were run using 16 CPU-threads of execution for the NEAT-Python module and a single GPU for the autoencoder training and compression[1]. All the experiments started with the same set of seed populations and pre-trained autoencoder, to ensure a common starting point. The seed populations consist of randomly initialized CPPN's with no hidden nodes or evolution applied. The pre-trained autoencoder was trained to compress these seed populations of low complexity individuals and was re-

[1]Through testing, it was noted that increasing the number of CPU threads for exploration was the most effective way of reducing the time required for running an experiment. On average, each experiment took roughly 60 hours to complete using an AMD Ryzen 7 1700 CPU and Nvidia Geforce GTX 1060 GPU.

Experiment Group	Autoencoder Types	Training Data
Static AE	Vanilla	N/A
Random AE's	Vanilla	N/A
Full History AE's	Vanilla/Denoising	All past novel sets
Latest Set AE's	Vanilla/Denoising	Latest novel set
Novelty Archive AE's	Vanilla/Denoising	Novelty archives

Table 4.1: List of transformation setups used in testing.

quired to for the experiments to begin their first iteration. At the end of each iteration all the populations (genotype & phenotype), as well as the training set for transformation and a dictionary of statistics, are saved to storage for later evaluation.

During exploration, 10 populations of CPPN's were evolved using the NEAT module for 100 generations each. As we've covered, the system uses K-Nearest neighbors when calculating the novelty score for each individual, with $K = 20$. A maximum of three individuals are inserted into the novelty archive at the end of each NEAT generation, provided an exact copy does not exist. When the exploration phase is complete, the transformation phase assembles a training set of novel individuals by picking the 100 most novel, feasibly individuals from each population. In the case of the novelty archive experiments, the default method is ignored and the novelty archive from each NEAT population is assembled into a dedicated training set. For transformation, the autoencoder is trained (if applicable to the configuration) to compress the buildings into a one-dimensional latent vector of 256 real values. We chose this vector length to maximize reconstruction accuracy without being computationally infeasible, though testing the system with shorter vectors would be desirable to identify if we can reach the same level of complexity with less computational load. The autoencoder model is trained for 100 epochs with a batch size of 64 individuals, with no early stopping criteria specified. The categorical cross-entropy loss function from the Keras library is used for training, visualizing the training history through the categorical accuracy measure for debugging.

Human Authored Buildings

Some performance measures used for the evaluation required a dataset of life-like buildings. To obtain these examples, the "AHouse" MCEdit filter (Brightmoore, 2016) was used to generate two sets of medieval style houses containing 200 individuals each. The filter takes an empty bounding box volume as input (which determines the building's maximum size), and uses a random seed to determine its outcome. The buildings are assembled according to a set of hard-coded processes, with no evolution or evaluation on the outcome. The generated buildings have a fully designed internal space, including stairs, windows, floors, etc. This provides an ideal example of a realistic building to compare with the generated content. The first set of individuals used a fixed bounding box size of 20x20x20 to provide a variety of examples which maximize the resolution used by this project's generator. The second used a random bounding box size provided it was at least 10 voxels wide, tall, and deep. This was done to generate more variety in the size and shape of the buildings to compare to. Figure 4.1 illustrates an example of a life-like building that was used in this dataset.

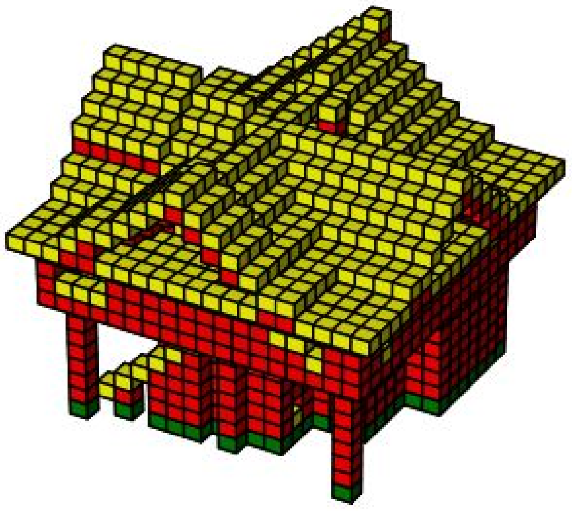

Figure 4.1: Example of a lifelike building used in the dataset.

4.2 Evaluation Strategy

The fundamental challenge faced during this project was comparing the performance of the system as a content generator across multiple configurations. This is due to the fact that each experiment is constantly evolving its own novelty function according to the content it generates, meaning direct comparisons of fitnesses between iterations of the algorithm, as well as separate experiments are not meaningful. Furthermore, whilst the primary objective of the algorithm is to generate buildings it deems to be novel, the content must be assessed on its adherence to certain architectural constraints to ensure a level of functionality and believability. Blending these notions of quality and diversity is a major objective of the generator's creative process and this evaluation method. This section contains a mix of two approaches to evaluation; a quantitative evaluation of the generated content in terms of diversity and expressivity, along with a qualitative analysis of the content generated from each experiment. We follow this up with a discussion of the results in section 4.3 and the limitations of the current implementation.

4.2.1 Expressive Range of the Generator

Recall that the expressive range of a content generator refers to the space of possible individuals it has the potential to create, and how it is biased to certain types of individuals (Shaker et al., 2016; Smith & Whitehead, 2010). We use this approach to illustrate the diversity of the training sets generated by each experiment's final exploration phase with respect to the three building features described in section 3.4.2 (building instability, surface area ratio, and symmetry). More specifically, we plot a 2D histogram for each feature combination, resulting in three separate expressive tests. In this analysis, we are looking for differences in the distribution of frequencies across the feature space, with a more evenly spread population being more expressive in terms of the features we are testing. The shape of the distribution also gives us a rough idea of the overall trend of the populations' features, allowing us to differentiate between the content generated by each experiment. For each

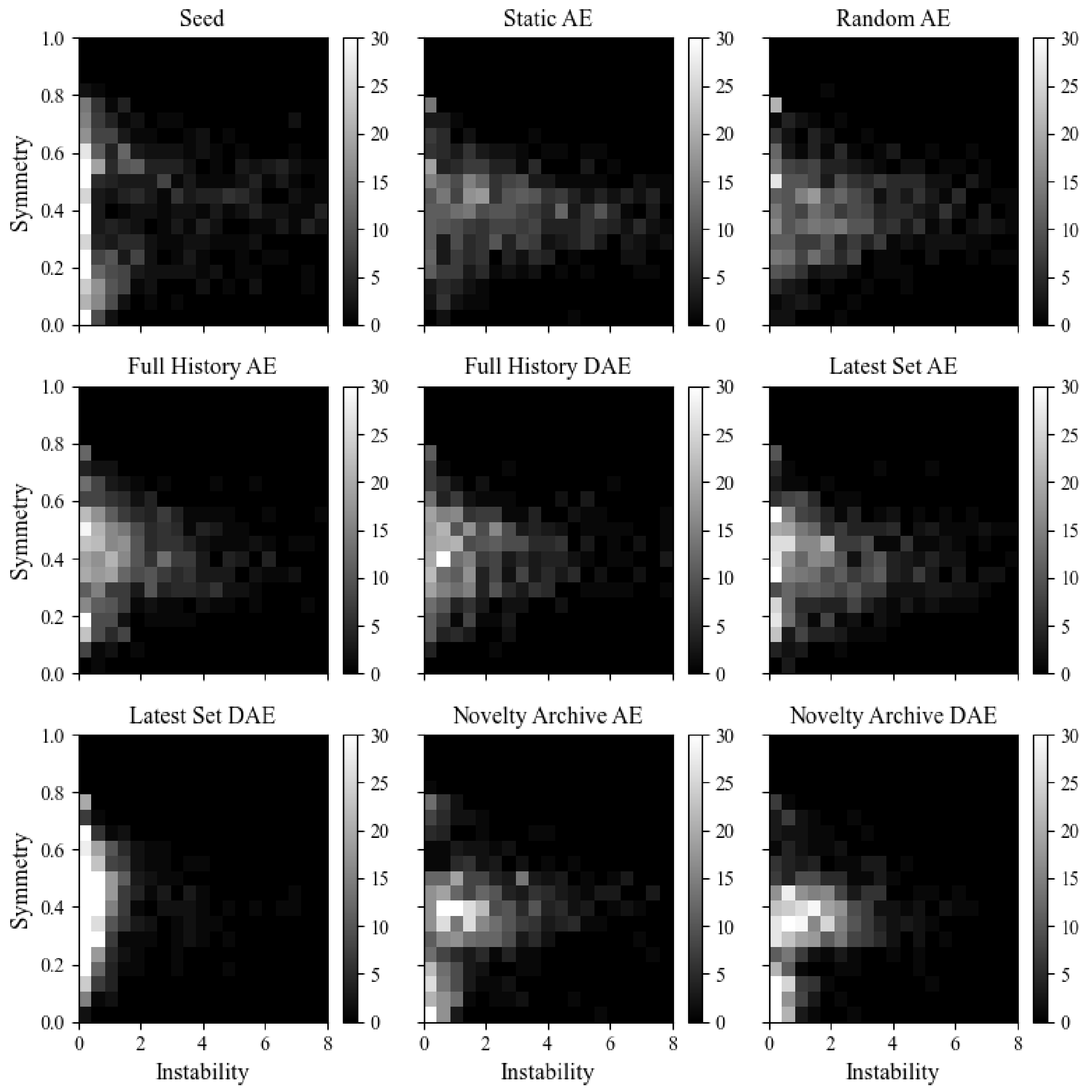

Figure 4.2: Expressive range of the generator in terms of lattice instability and symmetry.

test, we compare the content of each experiment to their shared seed population that was used as their starting point for evolution. This allows us to highlight the difference between the most basic possible individuals (randomly initialized CPPN's with no hidden nodes) and the most complex individuals created by the generator in its different configurations.

In this first test, we analyze the distribution of individuals across the building features of instability (x-axis) and symmetry (y-axis). Looking at figure 4.2 we can see a couple of outliers in the overall trend. Across all the experiments, we observe a tighter distribution of individuals across the symmetry dimension compared to the seed population. The autoencoders trained on the novelty archive both converge to very similar location in the state space, with the majority of their populations being highly stable, relatively asymmetric structures. The latest set denoising autoencoder also converged to this region of the state space, to an even greater degree than the novelty archive tests. The static autoencoder

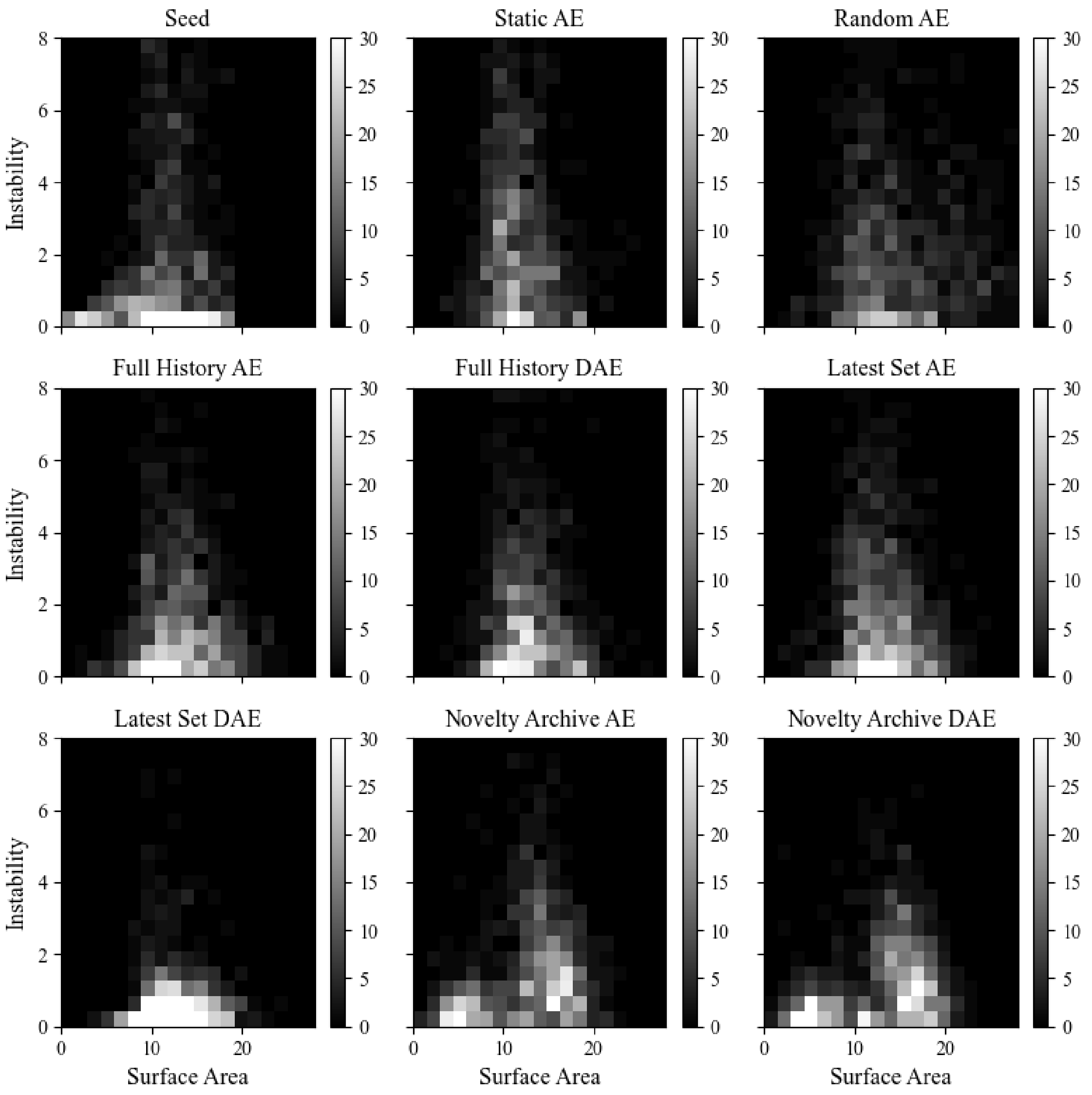

Figure 4.3: Expressive range of the generator in terms of lattice surface area and instability.

showed a similar distribution to the random and latest set vanilla autoencoders. The distributions for the full history autoencoders seem to fall somewhere in between the novelty archive autoencoders and the rest of the experiments for this test.

Figure 4.3 shows the results when comparing the population's surface area (x-axis) to their instability (y-axis). Following from the previous test, which also included instability as a dimension, the novelty archive experiments and latest set denoising autoencoder experiment show the most noticeable difference in shape in this feature space. The rest of the experiments show similar expressive ranges for instability, with the static autoencoder in particular producing the most even distribution across this dimension. The seed population and novelty archive experiments show similar expressive ranges in terms of surface area in this test. The random autoencoder experiment seems to be the only configuration which is capable of generating individuals with the most extreme surface area ratios, which is likely

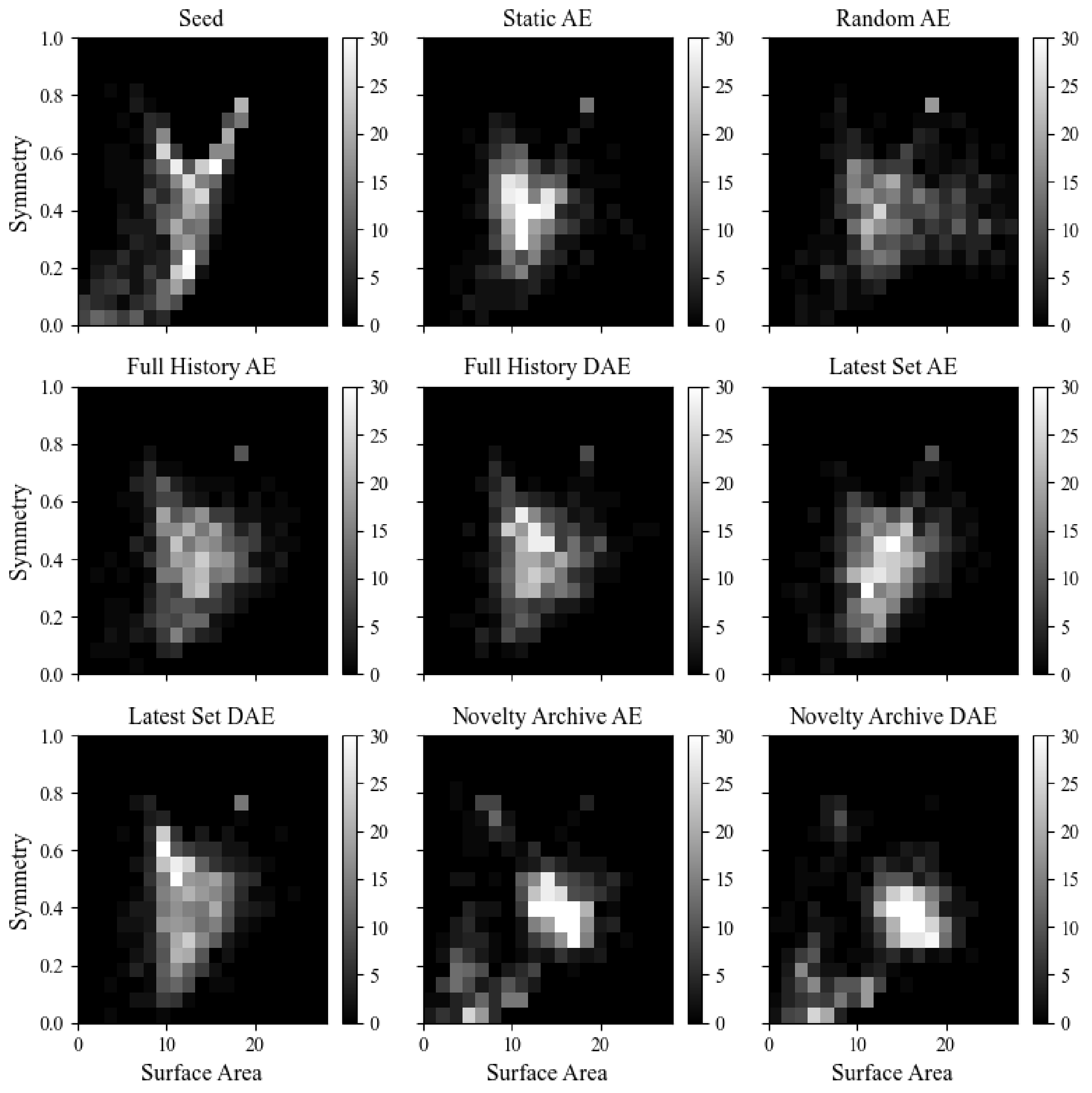

Figure 4.4: Expressive range of the generator in terms of lattice surface area and symmetry.

a result of noisy compression and is explored further in section 4.2.5.

Finally, figure 4.4 shows the results for comparing building surface area (x-axis) with symmetry (y-axis) for each experiment. In this test, we can see the biggest difference between the seed population and the rest of the experiments. As we saw in the first test, the seed population is more expressive across the symmetry dimension compared to the evolved populations. The novelty archive experiments are once again noticeably different in their distribution to the rest of the experiments, which all exhibit very similar shapes in the feature space. There seems to be a relatively common area of the feature space in the center of both dimensions which all the experiments are converging to, regardless of their configuration. The effect of the random autoencoder is once again visible in this test, showing the most evenly distributed population across both dimensions.

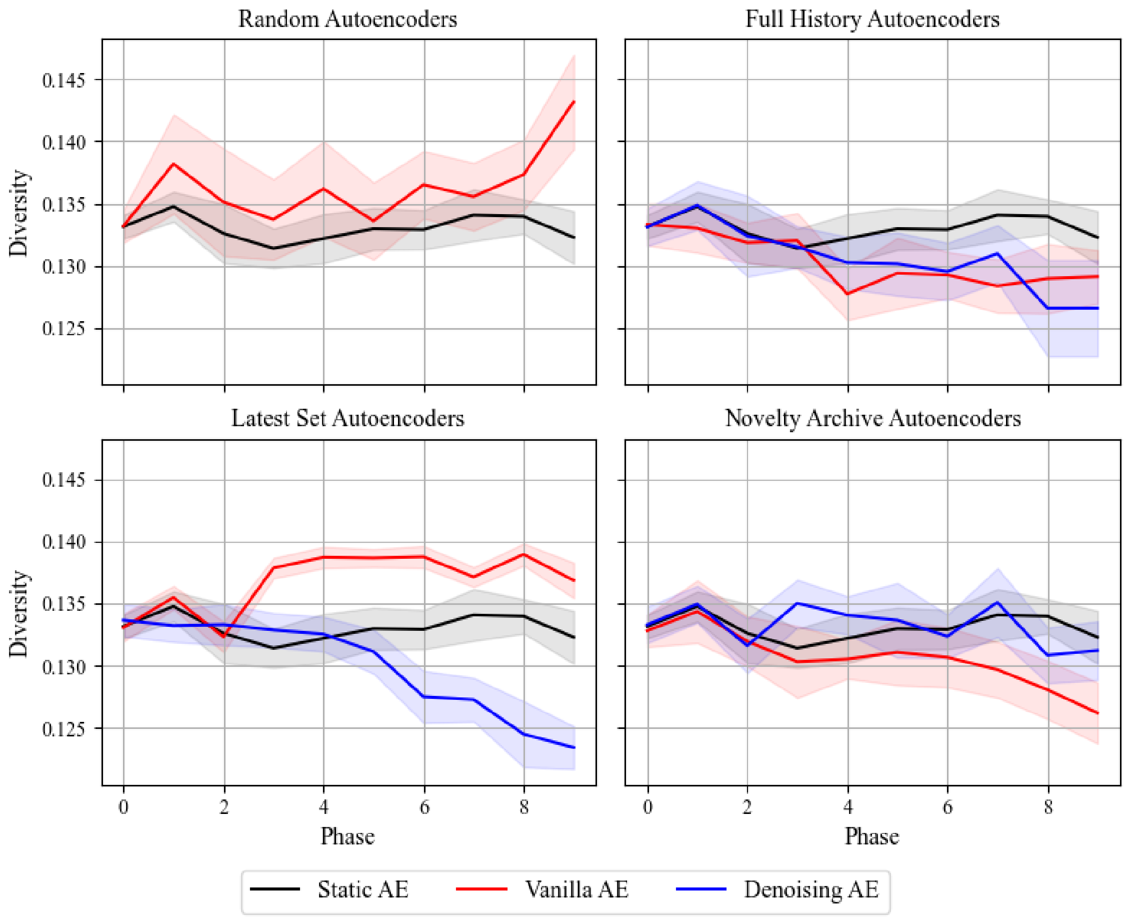

Figure 4.5: Mean population diversity observed in each experiment.

4.2.2 Population Diversity

Whilst the expressive analysis allows us to get a better idea of the distribution of content across a number of features, it does not give an objective value for the overall diversity of the populations. Choosing what building representation to use, and type of diversity measure to apply, has a big impact on the resulting scores. The nuance of calculating the diversity scores of the population lies in choosing the right diversity measure and domain for calculating diversity. As we mentioned, the novelty scores assigned through the latent vectors are not comparable across phases and experiments and so could not be used as the measure of diversity.

To overcome this issue, the simplest method is to calculate the diversity of buildings by comparing the raw lattice data (i.e, on a voxel level). At the current lattice resolution this method is feasible, but in future work with higher resolutions it would not scale well. *Kullback–Leibler* (KL) divergence is commonly used in statistics and pattern recognition for tasks such as speech and image recognition (Hershey & Olsen, 2007) as well as a way to compare and analyze game content (Lucas & Volz, 2019). We use KL divergence to calculate average diversity seen in each population throughout the experiments. To accomplish this task, each individual is flattened into a one dimensional array and a soft max function converts it into a probability distribution (i.e., all its values add up to 1). We assign

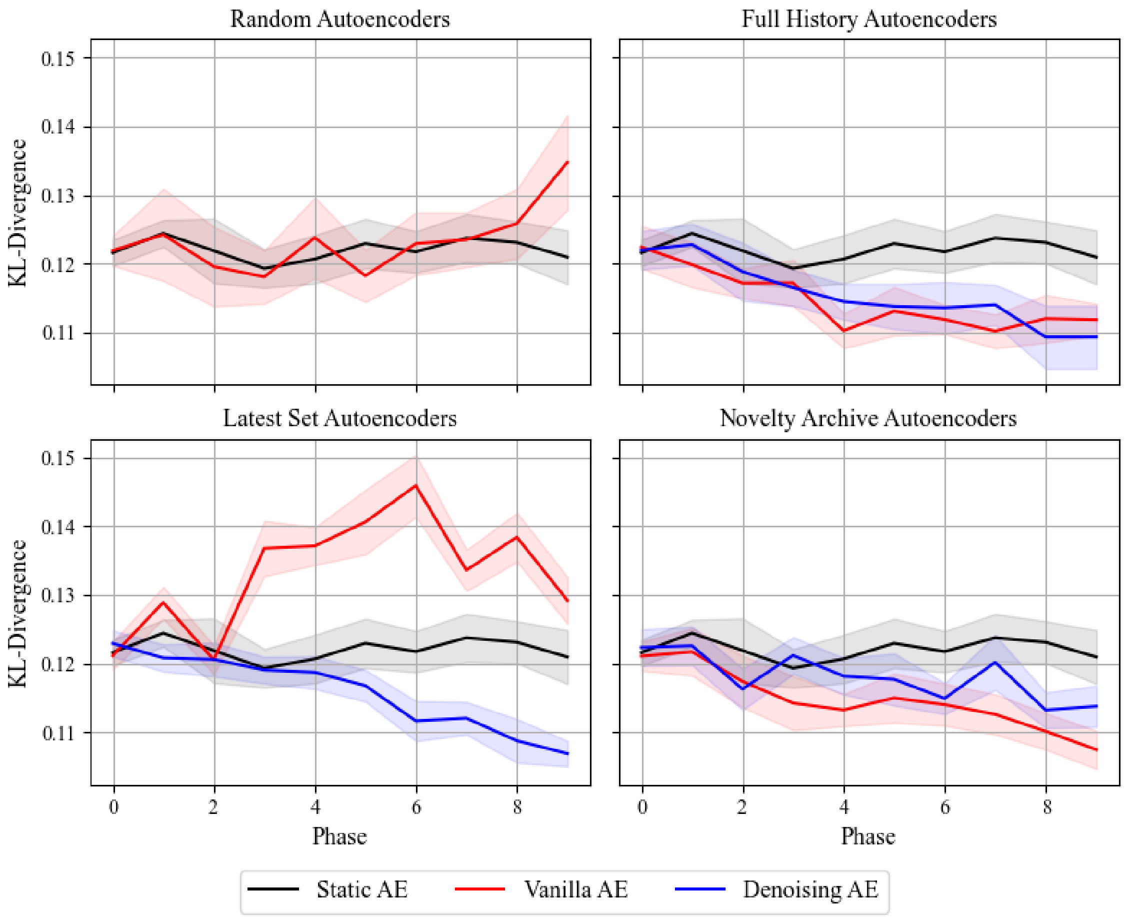

Figure 4.6: Average diversity from lifelike buildings.

diversity scores to each individual by calculating their average pairwise KL divergence to every other individual in the population.

Figure 4.5 shows the results for the KL-divergence performance measure, separated by autoencoder type and compared with the baseline static autoencoder. The static autoencoder showed a stable diversity over time compared to the rest of the experiments, which could be in part due to the lack of retraining of the autoencoder. The random autoencoder experiment was the only configuration that showed a significant increase in voxel diversity across the iterations of the algorithm. The autoencoders trained on the full history and novelty archives showed very similar diversity trends, with their populations on average converging slightly over time. The latest autoencoders have the biggest discrepancy in the results between the vanilla and denoising autoencoders. Apart from this case, using a denoising autoencoder does not seem to cause a significant difference compared to a standard autoencoder. It's helpful to note that a decline in voxel diversity does not mean the algorithm is not working correctly, since it evolves buildings according to novelty of high-level features and not diversity of voxel layouts.

4.2.3 Divergence from Human Authored Buildings

We can expand upon the diversity metric further by approach the task from a different angle. Rather than calculating diversity with respect to other individuals in the current population, we can compare the generated content with the fixed set of realistic buildings described in section 4.1. this allows us to get a better idea how the population is evolving over time, as decreasing diversity scores for this metric indicate the population is converging towards life like buildings. Whilst this should not be taken as indicative of an individuals quality from an architectural standpoint, it does allow us to understand the overall trend in the populations for each experiment as time goes by.

Figure 4.6 shows the results for this evaluation metric for each experiment. Interestingly, the diversities from human building are very similar to the results for the population diversity scores. The static autoencoder once again exhibits a stable diversity score compared to the rest of the experiments. The novelty archive autoencoders and full history autoencoders converge to slightly more human-like buildings compared to the static autoencoder. The latest set denoising autoencoder also followed the trend of its discrepancy with its vanilla autoencoder counterpart, converging to similar diversity levels to the novelty archive and full history experiments. These results show that the generator is capable of creating more and more human like buildings without any explicit information on their structure other than the requirement of an entrance and lack of floating voxels. This is also the case without the generation of an interior design for our buildings, which is present in the human-like structures, and could further improve their similarity in this measure.

4.2.4 PCA of Training Sets

By using dimensionality reduction through a principal component analysis (PCA), we can visualize the generated content in a two-dimensional space in terms of their principal components. This gives us the means to qualitatively analyze the distribution of the components across the component space by looking at the differences between experiments on a scatter plot. This also gives us another domain to calculate the diversity of the generated content, this time using the principal components rather than the raw lattice data. In this performance measure, we use the novel training sets generated by the exploration phase for each experiment, rather than all the population data. This gives us an idea of the diversity of the novel data used for training the autoencoders.

To ensure the PCA is as consistent as possible across experiments, all the training sets from every exploration phase of every experiment were gathered into a super set and fitted to a PCA object. Each three-dimensional lattice was flattened to a single one-dimensional list of voxels to be compatible with the Scikit PCA class. The PCA object was then used to compute the PCA values for each individual in the training set for the final iteration of each experiment. These principal components were plotted on a scatter plot and compared to the seed population, as we did for the expressive analysis. The diversity of the individuals was found by calculating the mean pairwise Euclidean distance for each individual to the rest of the training set.

Figure 4.7 shows the scatter plots for the principal components of the training set from the final iteration of each experiment. Figure 4.8 shows the results for the average diversity score for each individual in the principal component space. The novelty archive autoencoders show a significantly lower population diversity and scatter plot distribution compared to the rest of the experiments, continuing the trend observed in the previous per-

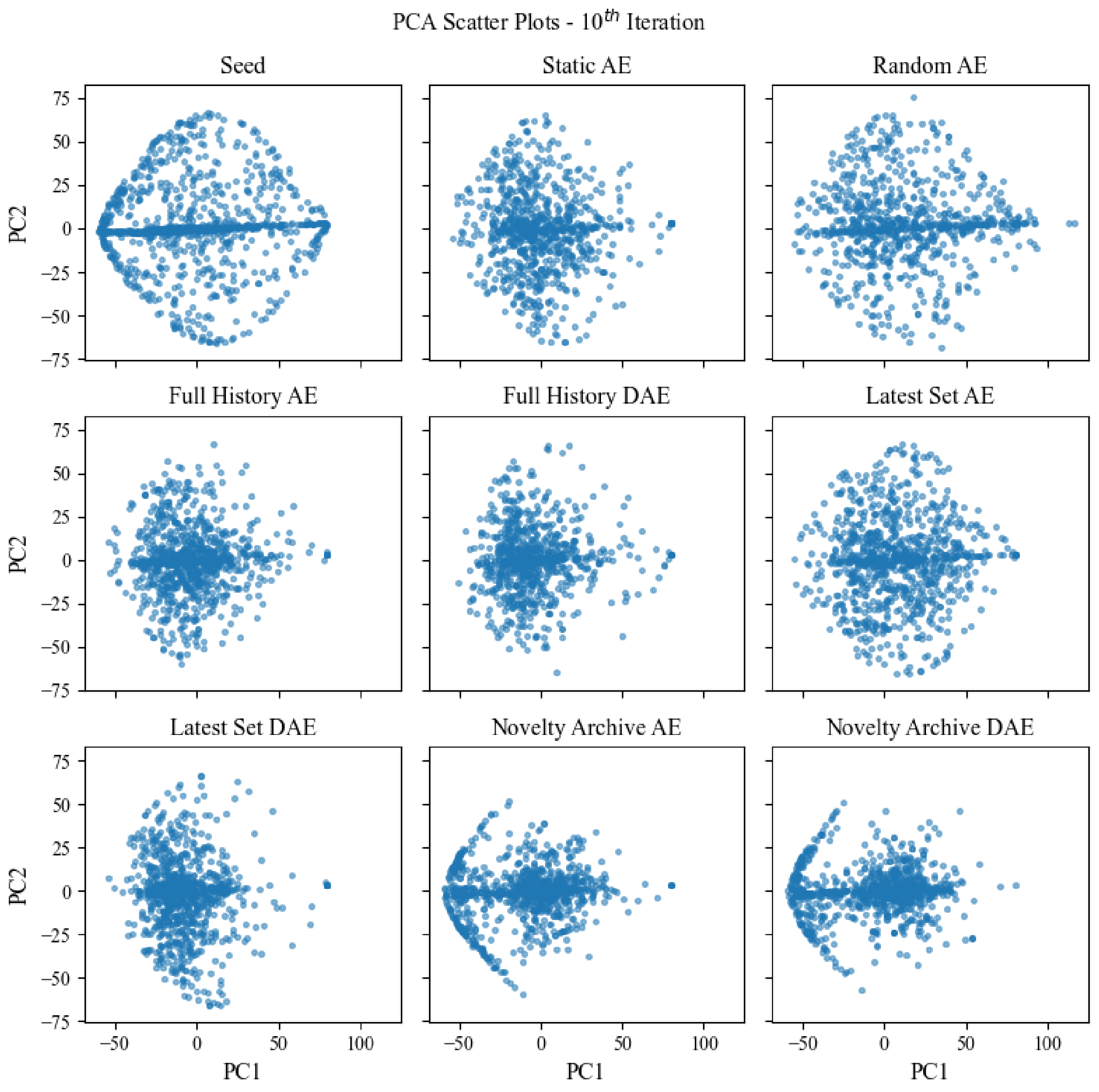

Figure 4.7: Scatter plot of the principal components derived from the novel training sets.

formance measures. The novelty archive experiments consistently converge to some reason of the search space, both in the voxel space and principal component space. The latest set denoising autoencoder also continued its tendency of lower diversity compared to its vanilla autoencoder counterpart. The rest of the experiments show similar diversity scores and distributions in the component space, with the full history autoencoders falling somewhere in between the behavior of the static autoencoder and novelty archive autoencoders.

4.2.5 Examples of Buildings

We round of this section of the evaluation with a qualitative look at the range of buildings generated by each experiment. For each experiment, the populations of buildings from every other iteration of the algorithm were collected and compressed using their respective autoencoders. Each individual was then assigned a novelty value using the generator's

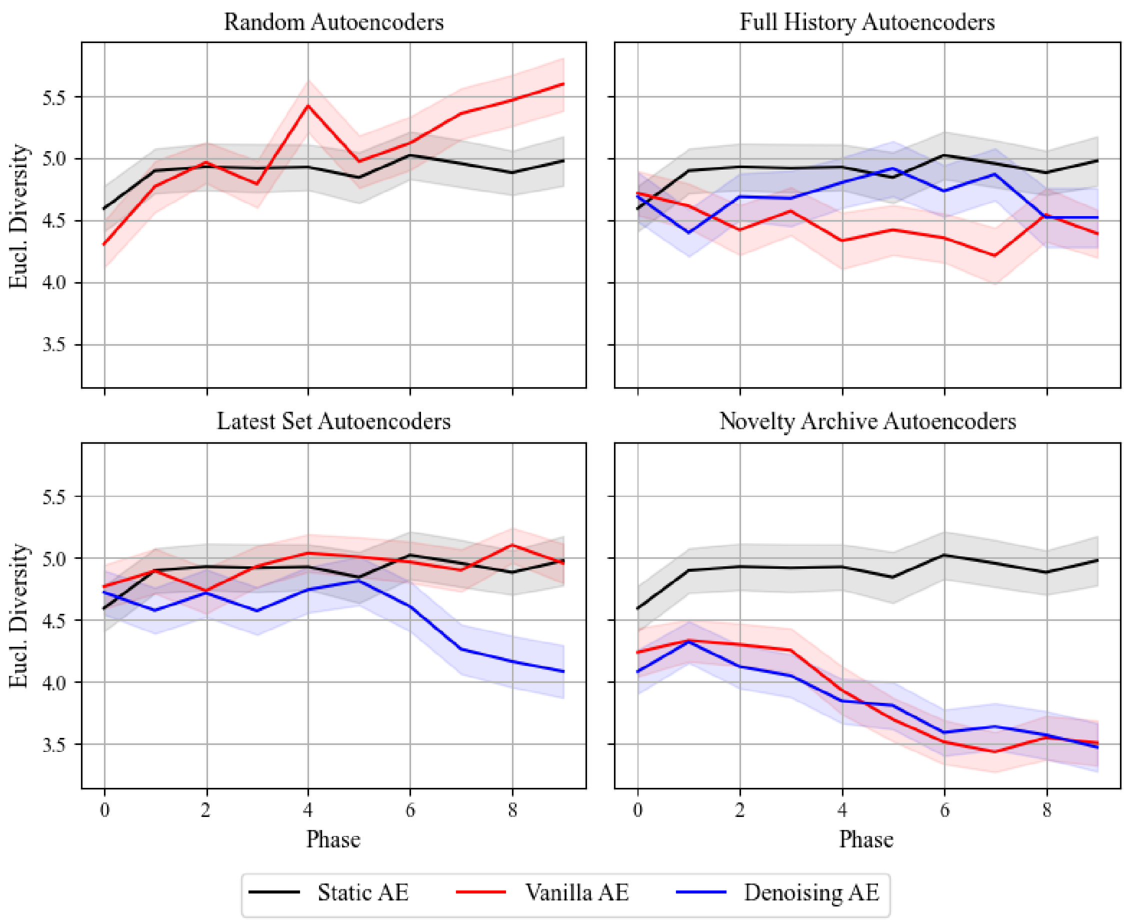

Figure 4.8: Mean diversity of principal components observed in each experiments' training set.

novelty function (defined by their autoencoder), and were sorted according to the resulting novelty scores. These sorted populations were used to plot the least, median and most novel individuals as well as the lower and upper quartile individuals for each phase. This allows us to look at how the examples of low and high novelty evolve over time as the autoencoder is retrained on new data. It also allows us to see the effective increase in complexity in the structures, which as we mentioned is not guaranteed even though their CPPN's may be increasing in genetic complexity.

Figure 4.9 shows the examples of buildings generated for the static autoencoder experiment. It's immediately visible that due to the lack of transformation on the generator's definition of novelty, the complexity of the buildings does not consistently increase over time. In fact, very similar buildings were classified as the most novel for phase 4 and phase 10, highlighting that the generator is stagnating in its search for novel individuals. Figure 4.10 shows the examples of buildings generated by the random autoencoder experiment. In this case, we can see a significant difference to the static autoencoder, in that the patterns of buildings noticeably change over time. By phase 10 we can see the impact of the lack of training on the autoencoder, as the lattices around the median novelty range are noisy and don't have any defining features.

In figure 4.11 we can see the results for the latest set denoising autoencoder. The change in complexity in this case was rather subtle, with the most novel individuals gradually producing more complex combinations of patterns compared to previous iterations. The less novel buildings retain a simple structure, in fact exhibiting simpler features over time compared to early iterations.

The most interesting results for this part of the evaluation come from the full history (figure 4.12) and novelty archive experiments (figure 4.13). The type of autoencoder did not seem to have an effect on the complexity of the buildings seen in both experiments. In both experiments, we can see a clear increase in the complexity of the structures across the board. regardless of their novelty. The most novel buildings in both cases exhibit interesting structures with very complex patterns and shapes. From these results, it is clear to see that these two configurations in particular are capable of iteratively complexifying the population and finding interesting individuals from the problem space more effectively. This is likely due to the size of their training sets, which in both cases grow linearly over time, and is explored further in the coming sections.

4.2.6 Autoencoder Reconstruction Accuracy

So far the performance measures have covered the content generated by the experiments, but the generator can also be assessed on existing and/or unseen data. Whilst DeLeNoX produces novel and increasingly complex artifacts, it also trains an autoencoder whose capability for compressing complex data is increasing in tandem. As we've discussed, this model can be seen as a novelty critic, and therefore can be used to analyze individuals that haven't been produced by the generator itself. In this section of the evaluation, we test the reconstruction accuracy of the autoencoders for each experiment using the set of human authored buildings described in section 4.1. We follow this up with an attempt to use the final autoencoders from each experiment as a judge for the novelty of these human buildings.

We follow the hypothesis that as the complexity of the generated content increases, the autoencoders that are created become more robust to unseen data, and are better able to compress increasingly complex structures. An autoencoder which has a better compression accuracy is better able to identify the high-level patterns of a structure, and therefore is better able to assess its novelty. Therefore, we compare the compression accuracy of the autoencoders from each phase of the experiments using the set of human authored, life like buildings generated for this evaluation. This test allows us to expand upon the results of the human divergence metric, providing another comparison with realistic life like examples to test the capability of the system.

Figure 4.14 shows the results for this reconstruction experiment, showing the reconstruction error on the y-axis (lower is better). As expected, the static autoencoder has a constant reconstruction accuracy, as it remains unchanged across the entirety of its experiment. The random autoencoder also exhibits expected behavior, with very high reconstruction error which is a result of its lack of training in the transformation phase, effectively using random weights for its CNN. Both the full history and novelty archive autoencoders show a clear and consistent decrease in reconstruction error over time, gradually improving and outperforming the static autoencoder. Both latest set autoencoders do not show a clear long term pattern over time, retaining the same rough level of reconstruction error from their first phase. Whilst the reconstruction error is high across the board and there is no clear leader in minimizing the absolute reconstruction error, this test at least shows that with

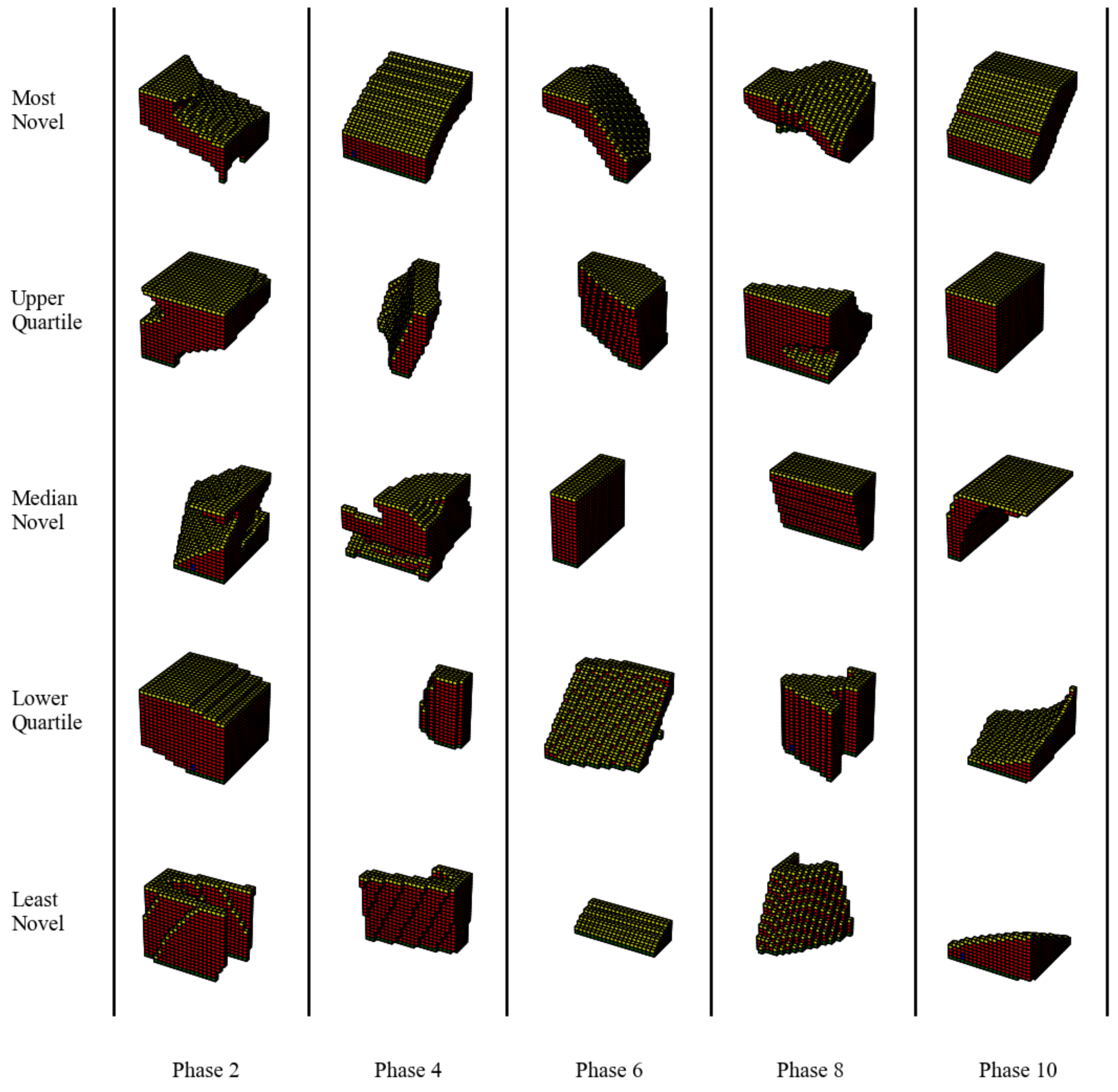

Figure 4.9: Examples of buildings included in the training sets generated by the static autoencoder.

more training data the autoencoder consistently improves, as evident in the full history and novelty archive experiments.

4.3 Discussion

When looking at the results as a whole, we can clearly see the impact of the approach to transformation on the evolution of the system and content it generates. The different categories of experiment produced noticeably different results in almost all the evaluation measures tested, especially in how they develop over time. The type of autoencoder used (vanilla vs denoising) did not seem to make a significant impact on the results in a clear manner, highlighted by the fact that the two types had near identical reconstruction errors for the same transformation method. The implementation of a variational autoencoder is a

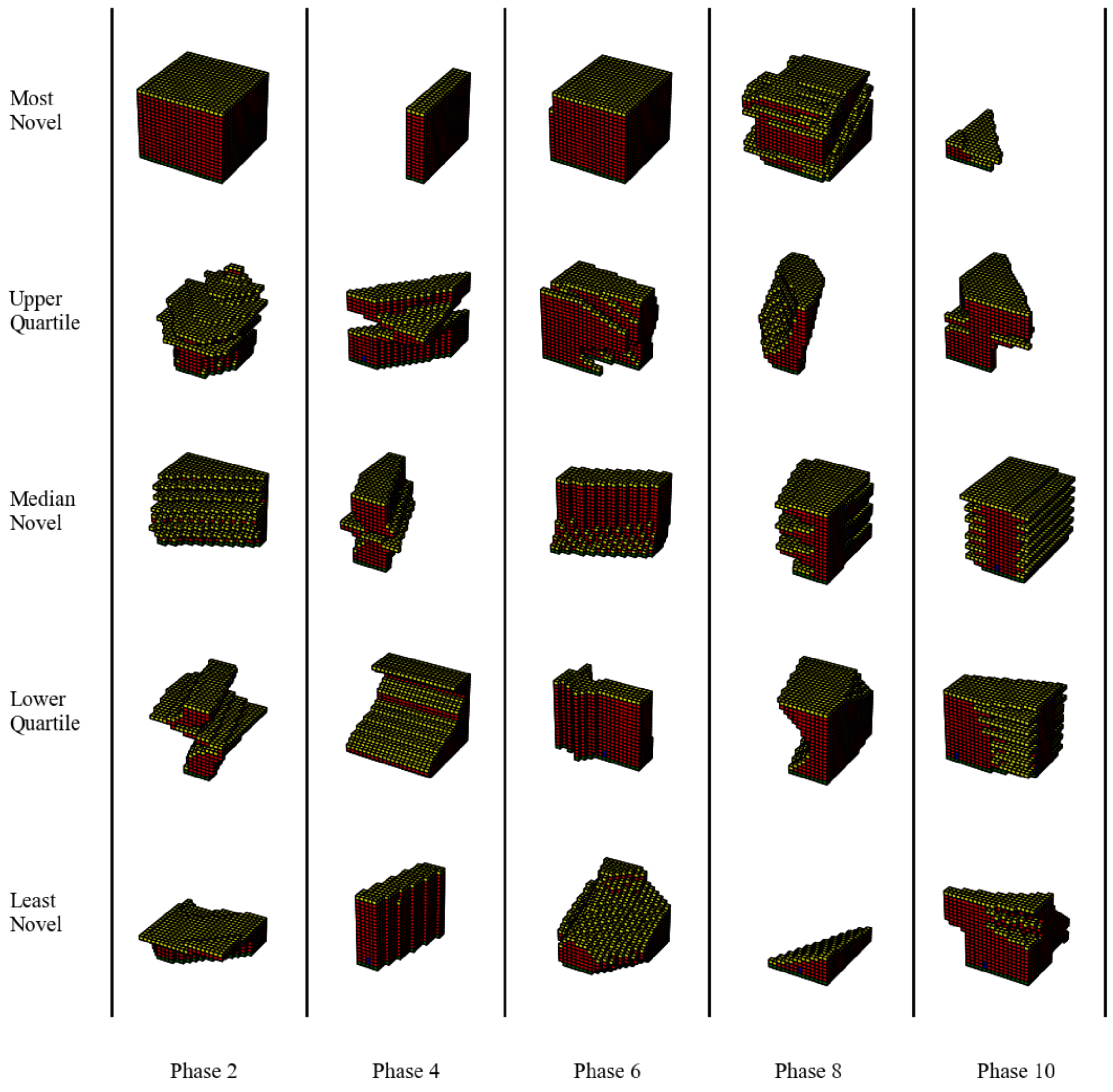

Figure 4.10: Examples of buildings included in the training sets generated by a random autoencoder.

clear area of improvement for this implementation, as in theory it should be better suited for this task due to its regularization of the latent space (Kingma & Welling, 2019). In terms of the training method, it seems that the system is more consistent and capable of exhibiting open-ended complexity when the autoencoder is given more data for the training set. This is clear to see in the building examples of figure 4.13 and figure 4.12, which by iteration 10 are producing more complex novel data than the rest of the experiments (which are trained on less/no data). Therefore, the full history and novelty archive autoencoders seem to be the best approaches to ensuring the system exhibits consistent OE over time.

There's an interesting discussion to be had on the observed diversity of the populations. By analyzing the expressive range of the experiments, we observed that the full history and novelty archive autoencoders produced the most noticeably different distributions in the feature space, converging to smaller ranges across the board of tests. Similarly, these

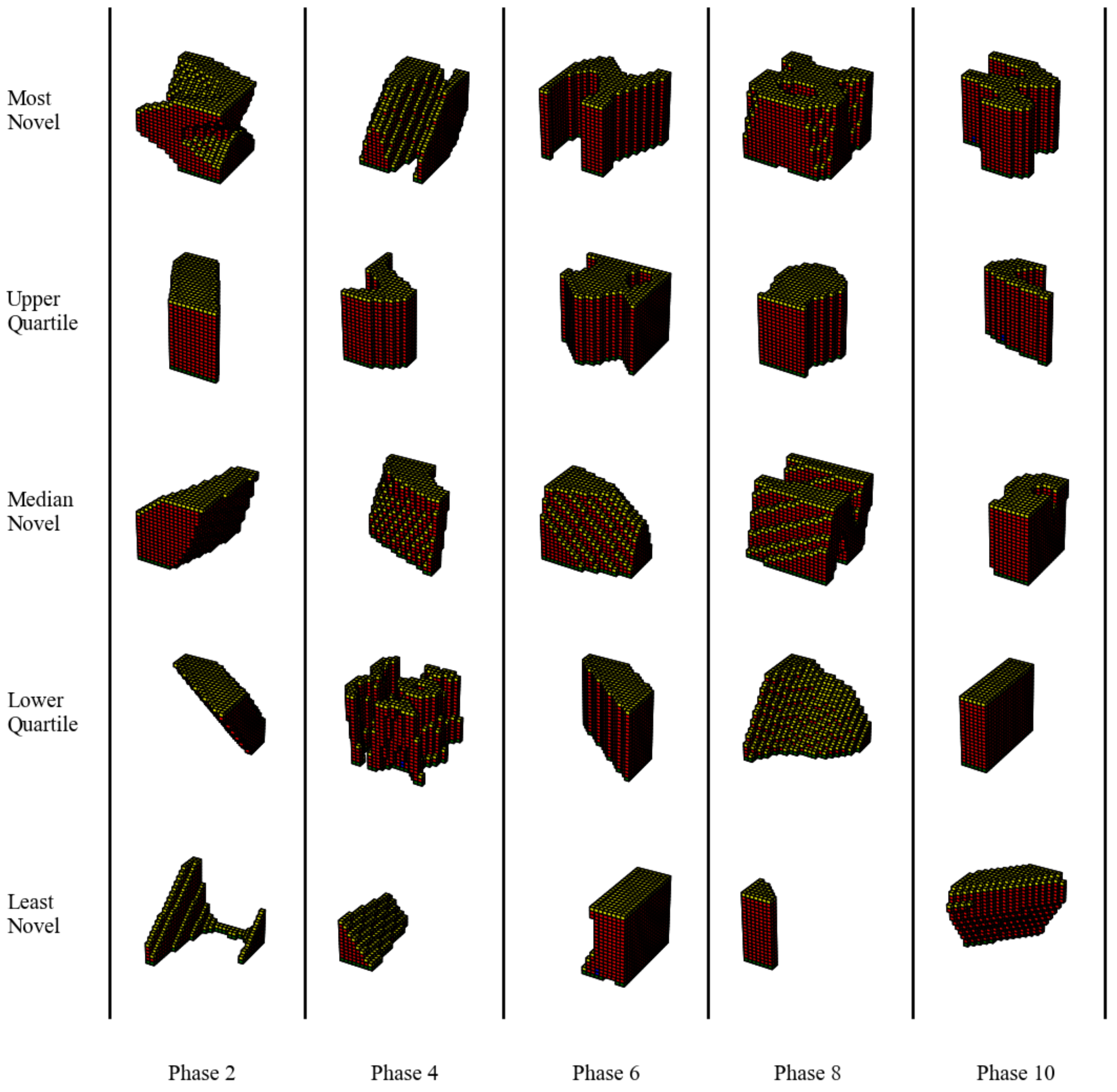

Figure 4.11: Examples of buildings included in the training sets generated by a denoising autoencoder trained on the latest training set.

same experiments also exhibit among the lowest diversity, both when looking at voxels and principal components. It seems that these two training methods, which are most capable of producing complex individuals, become less expressive compared to the other experiments when looking at the raw data. It's important to remember the system optimizes itself according to its own definition of novelty, which is based on the detected high-level features in the structures, not on the divergence of their voxels. Therefore, whilst some experiments see a slight convergence on the voxel level, the system is producing increasingly diverse data from a structural point of view, which is the effective goal of the system. In other words, the creativity of the generator does not lie in the content's voxel diversity, but in the diversity of their latent vectors. More work needs to be put into identifying the right features to be analyzed to get a better quantifiable measure for diversity that is more in line with the creative goal of the generator.

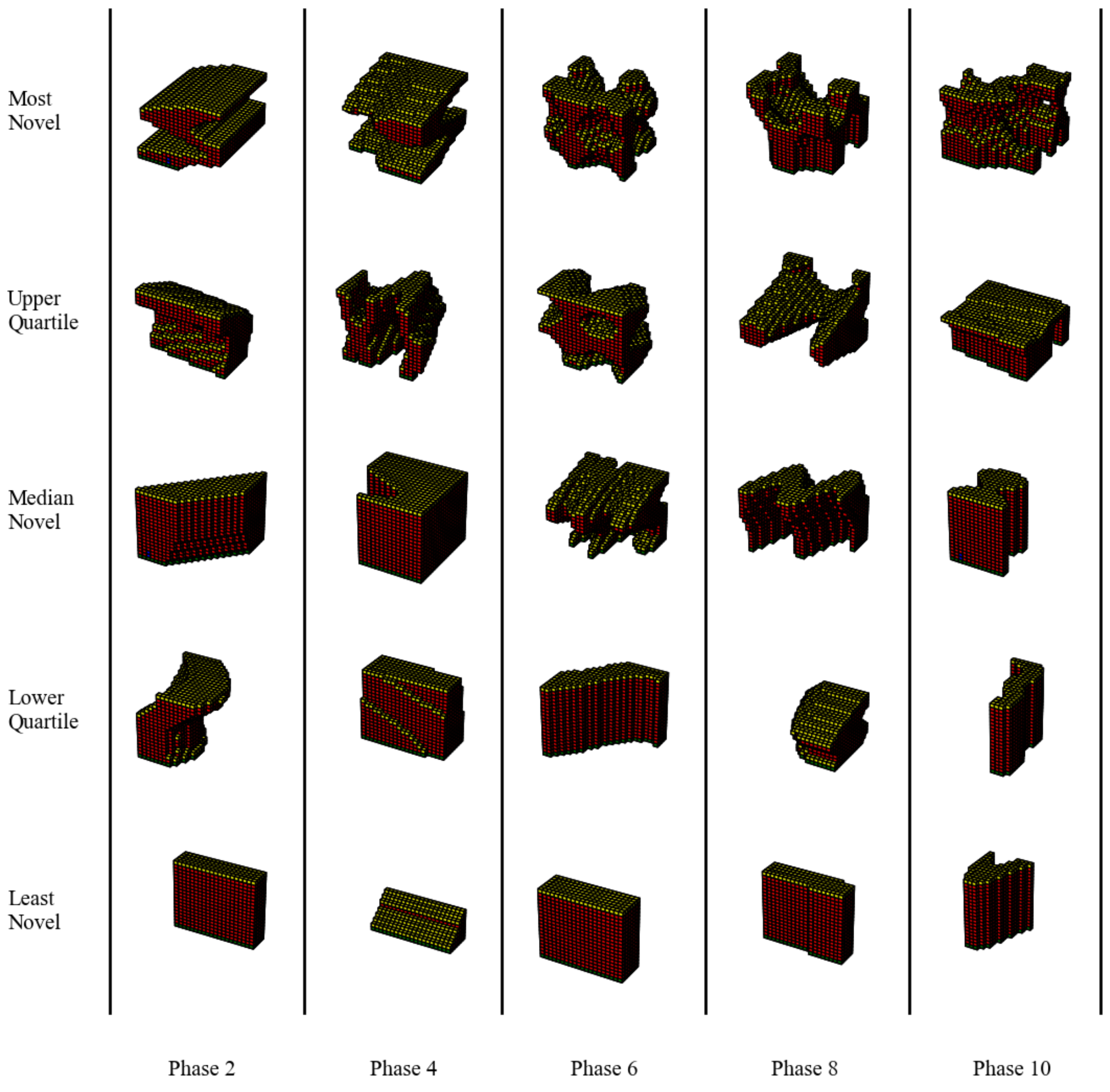

Figure 4.12: Examples of buildings included in the training sets generated by a denoising autoencoder trained on the full history of training sets.

When looking at the random and latest set autoencoders, we can see the impact of the training method on consistency of evolution, creativity, and diversity. The random autoencoder produces the most expressive behavior and diverse content on the voxel level, but we argue that it is the least creative configuration in testing. Echoing the argument presented in Smith and Whitehead (2010), an expressive system is not necessarily a creative one, as it does not assign creative meaning behind the content it generates. The random autoencoder cannot accurately identify high-level patterns in the data to base its novelty function on, meaning the latent representations are effectively random noise. This is clear in the building examples seen in figure 4.10, where the buildings lose any structure and become random patterns, unlike the forms seem in the other experiments. The latest set autoencoders perform better in this regard, producing more subjectively interesting patterns in the structure over time. However, from the diversity and reconstruction tests we can see

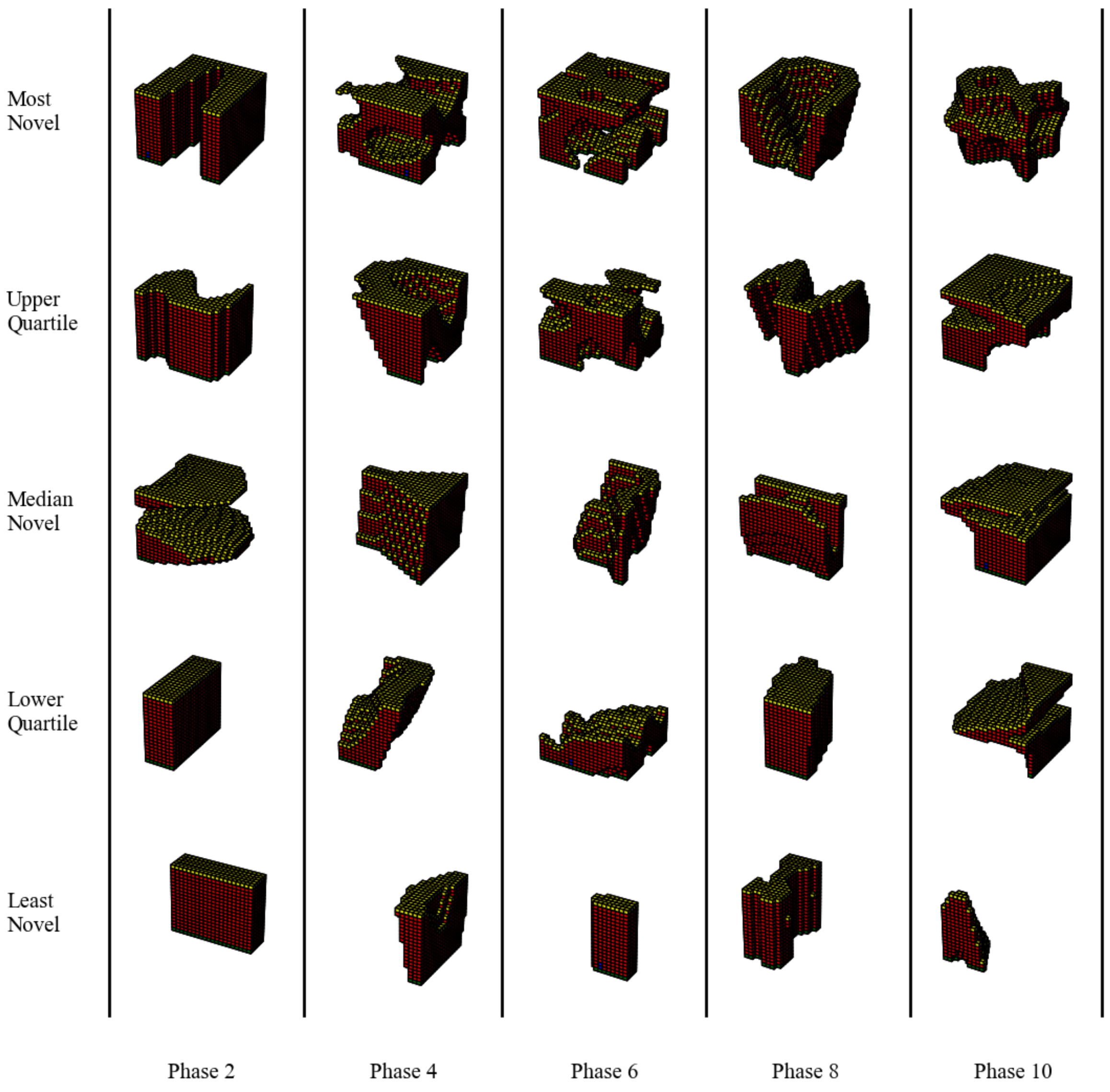

Figure 4.13: Examples of buildings included in the training sets generated by a denoising autoencoder trained on the novelty archive.

the lack of consistency over time, especially with respect to the type of autoencoder used. Without access to the full history of individuals, the latest set autoencoders seem to be less robust in the long run, exhibiting less complex patterns in the later iterations as a result.

4.3.1 OE of the Generator

In this part of the discussion, we focus on identifying the type of OE exhibited by the generator and its configurations (and by extension, the DeLeNoX algorithm as a whole). First, the system needs to be described in terms of the system model and meta-model, following the approach used by Banzhaf et al. (2016). Depending on how the meta-model is defined, the generator is capable of expressing at least two forms of OE. Regardless of the meta-model definition, the generator can be said to exhibit exploratory OE and satisfy other simple definitions of OE (Lehman & Stanley, 2011). This is because it perpetually

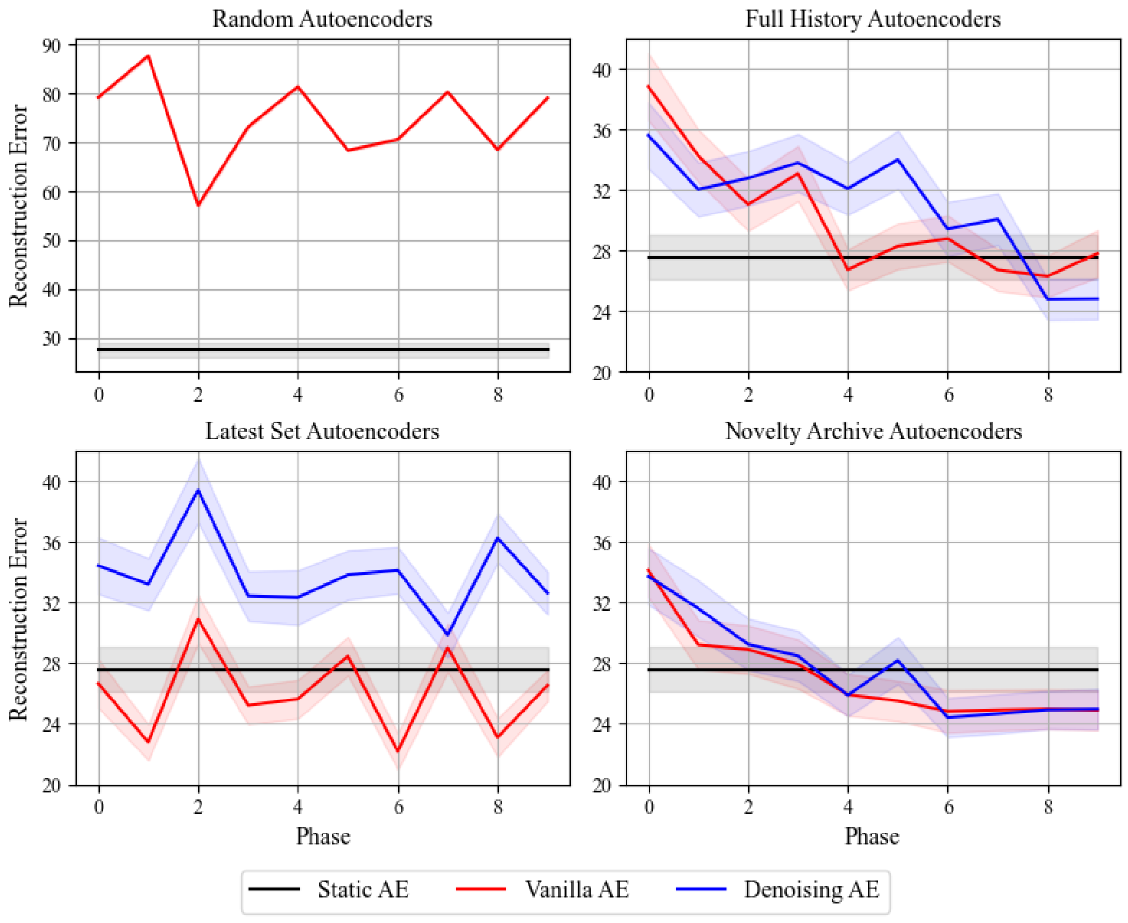

Figure 4.14: Reconstruction error observed for each experiment.

produces content which is optimized for novelty (according to the current model and static search space), and does so with increasing genetic complexity.

Consider the following description of the system; the autoencoder and distance function form the model implemented by the system. A low-level approach to the meta-model would define system concepts such as the lattice data structure, genotype, evolutionary algorithm used for exploration, etc. In this approach, one could consider the problem space as the space of all possible lattices that could be generated. Here the system can be said to exhibit expansive OE, but not transformative OE. This is because whilst the system model is modified through the transformation phase (retraining of the autoencoder/distance function), the meta-model is never modified. The retraining of the autoencoder modifies the fitness function and opens up inaccessible areas of the search space, but does not open up new domains/dimensions to explore. Achieving transformative OE is challenging with this approach, though an interesting avenue could be to modify the lattice resolution and materials available to the generator during transformation phases. Opening up these facets to the system would give it another set of dimensions to modify and exploit in its search for novel structures.

The meta-model could also be interpreted to define the high-level patterns currently learned by the autoencoder. In this approach, we consider the search space as all the possible latent vectors currently accessible to the generator. Whilst transformation phases still cause expansive OE, they may cause transformative OE if a new pattern is learned by the autoencoder as it opens up a new dimension/domain of the search space to explore. This is similar to the example of discovering winged flight in a biological system given by Banzhaf et al. (2016). Taking everything into it account, the system can be said to be expansively open-ended in terms of searching the space of all possible lattices, and transformative open-ended in terms of searching the space of all possible latent vectors.

Summary

In this chapter, we tested the generator using the set of transformation configurations given in table 4.1. We evaluated the configurations using a set of a quantitative and qualitative performance measures. On the quantitative side, we analyzed the expressive range of the generator across three building properties. We also compared the population diversity across experiments using KL-divergence on the voxel level, and euclidean distance with PCA. We measured the similarity of the generated content to a dataset of human buildings using KL-divergence. We tested the reconstruction accuracy of each experiments' autoencoder using the same dataset of human buildings. Finally, we plotted examples of novel buildings to qualitatively compare the complexity of the structures across runs. We closed off this chapter by discussing the results for each experiment and discussing the OE exhibited by the generator. In the next chapter, we conclude by discussing the limitations of the current approach and proposing some avenues for future work.

Chapter 5

Conclusion

The motivation of this project was to demonstrate the power of open-ended creativity/novelty and intrinsic motivation in procedural content generators. We chose Minecraft buildings as the domain for the generator to address the lack of creative generators for this application, and for their potential to complement research currently being made on settlement generators. We based the high-level approach for the system on DeLeNoX (Liapis et al., 2011) and existing work on soft robot evolution using CPPN-NEAT (Gravina et al., 2018). The theoretical framework established for this project was centered around the concepts of intrinsic motivation (Guckelsberger, 2020) and open-endedness (Banzhaf et al., 2016; T. Taylor, 2019). DeLeNoX was chosen due to its transforming novelty function, which allows it to satisfy all three types of OE (Banzhaf et al., 2016). The open-ended nature of this generator, and lack of a set objective function allow it to satisfy the properties of IM, which from a theoretical point of view allows it to fully exploit its creative potential.

Due to our focus on the capabilities of the generator from a creative standpoint, our evaluating method for the system centered around the transformation phase of the algorithm. We varied the training method for the autoencoders to observe the resulting impact on the novelty function and the evolution of the content generated over time. Our evaluation measures focused on the diversity of the content on a voxel level, as well as the expressive range of the generator and its ability to analyze unseen content. We also qualitatively compared a range of examples from each experiment to observe high-level differences in the structures that would not be identifiable in our quantitative tests.

Our results show that a fluid definition of novelty allows the generator to more effectively generate increasingly complex buildings over time. Whilst the content generated by the experiments using the transformation phase did not produce content that was clearly more diverse or expressive compared to a static novelty function, they did produce clearly more interesting and complex structures which varied significantly over time. The experiments which retrained the autoencoder using significantly more training data (full history and novelty archive experiments) showed the clearest increase in building complexity from a qualitative standpoint. These two experiments also showed the most distinct expressive ranges and improvement in their ability to compress realistic buildings over time. This is to be expected as presenting the autoencoder with a larger amount of training data, of a larger variety of complexity, should allow for a more robust autoencoder which can better identify patterns in the data. We did not find any significant difference between the use of a standard and denoising autoencoder in the experiments run so far.

5.1 Limitations

In this section, we discuss some limitations of the current implementation of the system, as well as the evaluation method, and look at how they can be addressed. The majority of the limitations lie in the quality of the generated structures from a functionality standpoint, as currently the system is fully focused on novelty without thinking about structure feasibility.

The lack of constraints on the generator is the first and most pressing limitation of the current state of the system. Currently, the only constraints applied to the system are the requirement of an entrance, no floating voxels, and the lattice must contain at least one active voxel. Whilst this gives the generator a bigger search space to explore and expand its creativity, restricting the search further to more desirable, feasible structures could give the system a better opportunity to focus its creativity in more interesting areas. Care would need to be taken not to over restrict the generator, as this would defeat the purpose of the system if it can only make strictly human like buildings. One easy constraint that could be added is a minimum bounding box size, to ensure only individuals of a meaningful size are considered. Another constraint could be to make sure the interior space is at least a certain volume, and that the volume is traversable by a player, to make sure they are feasible from a gameplay perspective. Finally, adding a stability constraint would prevent the structures from growing in infeasible ways, and could be paired with a repair function to add supports to weak areas. Re-running the experiments with these added constraints and comparing them to the results seen in this evaluation could highlight the system's creativity when constrained to a higher degree to produce structures of higher quality.

With such constraints in place, implementing feasible-infeasible two population evolution is another needed addition to the system. In its current state, the generator discards infeasible individuals and replaces them in the reproduction process, potentially discarding interesting content that could be slightly modified to become feasible. In this updated approach, the infeasible individuals would be modified to minimize the distance to the feasible threshold, i.e., to satisfy all the constraints. Whilst this wasn't a pressing addition with the current limited set of constraints, this would be more important as more meaningful constraints are added.

Another limitation is the lack of designed interior spaces for the buildings. Whilst the generator is not expected to generate fully designed buildings for both the interior and exterior, the interior space could be filled up with a generative method such as wave function collapse, or a search based method to create interior layouts according to a desired objective. This would make the generated structures more functional and allow for a fairer comparison with the dataset of human buildings, which all contain a basic and functional interior.

Finally, we used a single filter to generate the dataset of lifelike buildings for the evaluation method. As a result, all the buildings were of the same style (medieval era buildings) and followed the same fixed rules defined in the filter. Adding other filters with different building styles and rules will provide a more general comparison to lifelike buildings for the generated content, and could give a better idea of the style of building being generated by the system.

In the initial evaluation strategy, we intended to run a novelty critic test where the autoencoders were used to evaluate the novelty of the dataset of lifelike buildings. This would test how robust the autoencoders are to unseen data and could provide some interesting comparisons between experiment. The issue with this test circled back to the problem of comparing fitnesses between iterations or experiments. In the future we could take an informal look at what buildings each autoencoder found most novel and how this evaluation evolved over time, though for the sake of this project this wasn't implemented.

5.2 Future Work

The work done in this project opens up a number of areas to focus further research on. In section 5.1 we covered the current limitations of the generator and how resolving them can improve the content it generates and its utility on existing data. In this section, we build on that discussion and look into areas that weren't necessarily a limitation in the current setup, but could enrich the discussion and contributions of the system.

We believe one of the most interesting areas for future work lies in the integration of a QD optimization method rather than using simple novelty search. We've seen that whilst the generator is perpetually attempting to diversify the content in the latent space, this does not prevent the populations from converging in the voxel-based feature spaces. Using a QD algorithm like MAP-Elites in tandem with the autoencoder could preserve the diversity on both in the high-level structure of the buildings and on the voxel level. The individuals would still be evaluated using novelty search, rewarding individuals for their novelty compared to the rest of the population, but through the MAP-Elites archive we can observe novel elites from all over the feature space, opening the door for an interesting look at how the building properties we defined affect novelty.

The implementation of variational autoencoders could provide an interesting comparison with the current implementation. As we've discussed, VAE's have the potential to improve upon the current architecture and might be able to produce even more complex and interesting structures. Pairing this work with higher resolution lattices has the potential to produce some very interesting structures that could better illustrate the creative and open-ended capacity of this system. This would require a more computationally efficient implementation, as the current library has high overhead in the genotype to phenotype mapping. Fixing this would allow us to observe the representational power of the CPPN's and the ability of the autoencoders to scale with higher resolution input.

Whilst the system currently works in an isolated environment, it could be more closely integrated with the actual Minecraft world data in the future. In the GDMC report (Salge et al., 2021), contestants used common materials observed in the environment to construct the building. Future work could take a similar approach, allowing the environment to contribute to the evolution of buildings, introducing another facet to the creativity in the form of material selection and the resulting structure. This would be a big improvement over the current simple material model used, and would improve the OE exhibited by the generator, which would have another dimension of the search space to exploit. More ambitiously, this building generator could also be paired with a settlement generator in the future, following the objectives proposed in the GDMC (Salge et al., 2018). The system could be given empty plots of land of varying sizes and tasked with filling them in with interesting buildings. The settlement generator could influence the building generator through a set of parameters that cover the constraints, rotation of the entrance, etc.

Summary

In this chapter, we summarized the implementation of our proposed generator and our approach to its evaluation. We highlighted the effect of the transformation phase on the generated content and identified the full history and novelty archive autoencoder experiments as the most promising according to our evaluation method. We then discuss the limitation of the current state of the system, and how these can be solved in the near future to improve the potential of the generator. Finally, we propose a set of avenues for future work to build upon the contributions of this project.

References

AI Design. (1980). *Rogue: Exploring the dungeons of doom.* Epyx. — Cited on page 9.

al Rifaie, M. M., Ursyn, A., Zimmer, R., & Javid, M. A. J. (2017). On symmetry, aesthetics and quantifying symmetrical complexity. In *International conference on evolutionary and biologically inspired music and art* (pp. 17–32). — Cited on page 32.

Awiszus, M., Schubert, F., & Rosenhahn, B. (2021). World-gan: a generative model for minecraft worlds. *arXiv preprint arXiv:2106.10155*. — Cited on page 17.

Banzhaf, W., Baumgaertner, B., Beslon, G., Doursat, R., Foster, J. A., McMullin, B., ... et al. (2016). Defining and simulating open-ended novelty: requirements, guidelines, and challenges. *Theory in Biosciences*, *135*(3), 131–161. — Cited on pages 7, 23, 24, 51, 53, and 55.

Bedau, M. (1991). *Can biological teleology be naturalized?* JSTOR. — Cited on page 7.

Boden, M. A., & et al. (2004). *The creative mind: Myths and mechanisms.* Psychology Press. — Cited on pages 6, 7, and 8.

Brightmoore, A. (2016). *Ahouse building filter.* `http://www.brightmoore.net/mcedit-filters-1/ahouse`. (Accessed: 15-06-2021) — Cited on page 36.

Colton, S., De Mántaras, R. L., & Stock, O. (2009). Computational creativity: Coming of age. *AI Magazine*, *30*(3), 11–11. — Cited on page 6.

Colton, S., & Wiggins, G. A. e. a. (2012). Computational creativity: The final frontier? In *Ecai* (Vol. 12, pp. 21–26). — Cited on pages 5 and 6.

De Charms, R. (2013). *Personal causation: The internal affective determinants of behavior.* Routledge. — Cited on page 8.

Gravina, D., Liapis, A., & Yannakakis, G. (2016). Surprise search: Beyond objectives and novelty. In *Proceedings of the genetic and evolutionary computation conference 2016* (pp. 677–684). — Cited on pages 9 and 10.

Gravina, D., Liapis, A., & Yannakakis, G. N. (2017). Exploring divergence in soft robot evolution. In *Proceedings of the genetic and evolutionary computation conference companion* (pp. 61–62). — Cited on page 32.

Gravina, D., Liapis, A., & Yannakakis, G. N. (2018). Fusing novelty and surprise for evolving robot morphologies. In *Proceedings of the genetic and evolutionary computation conference* (pp. 93–100). — Cited on pages 14, 20, 32, and 55.

Green, M. C., Salge, C., & Togelius, J. (2019). Organic building generation in minecraft. In *Proceedings of the 14th international conference on the foundations of digital games* (pp. 1–7). — Cited on page 17.

Grillotti, L., & Cully, A. (2021). Unsupervised behaviour discovery with quality-diversity optimisation. *arXiv preprint arXiv:2106.05648*. — Cited on page 15.

Guckelsberger, C. (2020). *Intrinsic motivation in computational creativity applied to videogames* (Unpublished doctoral dissertation). Queen Mary University of London. — Cited on pages 1, 6, 8, 9, 23, and 55.

Hagg, A., Berns, S., Asteroth, A., Colton, S., & Bäck, T. (2021). Expressivity of parameterized and data-driven representations in quality diversity search. *arXiv preprint arXiv:2105.04247*. — Cited on page 15.

Harlow, H. F. (1950). Learning and satiation of response in intrinsically motivated complex puzzle performance by monkeys. *Journal of comparative and physiological psychology*, *43*(4), 289. — Cited on page 8.

Hershey, J. R., & Olsen, P. A. (2007). Approximating the kullback leibler divergence between gaussian mixture models. In *2007 ieee international conference on acoustics, speech and signal processing-icassp'07* (Vol. 4, pp. IV–317). — Cited on page 41.

Hinton, G. E., & Zemel, R. S. (1994). Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, *6*, 3–10. — Cited on page 12.

Hutton, T. J. (2002). Evolvable self-replicating molecules in an artificial chemistry. *Artificial life*, *8*(4), 341–356. — Cited on page 7.

Kimbrough, S. O., Koehler, G. J., Lu, M., & Wood, D. H. (2008). On a feasible–infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research*, *190*(2), 310–327. — Cited on page 24.

Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*. — Cited on pages 12 and 48.

Lehman, J., & Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, *19*(2), 189–223. — Cited on pages 1, 7, 9, 10, 23, and 51.

Liapis, A. (2016). Exploring the visual styles of arcade game assets. In *International conference on computational intelligence in music, sound, art and design* (pp. 92–109). — Cited on page 31.

Liapis, A., Martínez, H. P., Togelius, J., & Yannakakis, G. N. (2011). Transforming exploratory creativity with delenox. *arXiv preprint arXiv:2103.11715*. — Cited on pages 2, 15, 19, 20, 32, and 55.

Liapis, A., Yannakakis, G., & Togelius, J. (2013). Towards a generic method of evaluating game levels. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment* (Vol. 9). — Cited on page 10.

Liapis, A., Yannakakis, G. N., & Togelius, J. (2014). Computational game creativity.. — Cited on pages 6 and 9.

Lucas, S. M., & Volz, V. (2019). Tile pattern kl-divergence for analysing and evolving game levels. In *Proceedings of the genetic and evolutionary computation conference* (pp. 170–178). — Cited on page 41.

Mouret, J.-B., & Clune, J. (2015). Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*. — Cited on pages 1 and 10.

Peinado, F., & Gervás, P. (2006). Evaluation of automatic generation of basic stories. *New Generation Computing*, *24*(3), 289–302. — Cited on page 5.

Pugh, J. K., Soros, L. B., Szerlip, P. A., & Stanley, K. O. (2015). Confronting the challenge of quality diversity. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation* (pp. 967–974). — Cited on page 10.

Ryan, R. M., & Deci, E. L. (2000). Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, *25*(1), 54–67. — Cited on page 8.

Salge, C., Green, M. C., Canaan, R., Skwarski, F., Fritsch, R., Brightmoore, A., . . . Togelius, J. (2021). The ai settlement generation challenge in minecraft: First year report. *arXiv preprint arXiv:2103.14950*. — Cited on pages 16, 17, and 57.

Salge, C., Green, M. C., Canaan, R., & Togelius, J. (2018). Generative design in minecraft (gdmc) settlement generation competition. In *Proceedings of the 13th international conference on the foundations of digital games* (pp. 1–10). — Cited on pages 2, 16, and 57.

Schmidhuber, J. (2007). Simple algorithmic principles of discovery, subjective beauty, selective attention, curiosity & creativity. In *International conference on discovery science* (pp. 26–38). — Cited on page 6.

Shaker, N., Smith, G., & Yannakakis, G. N. (2016). Evaluating content generators. In *Procedural content generation in games* (pp. 215–224). Springer. — Cited on pages 13 and 37.

Smith, G., & Whitehead, J. (2010). Analyzing the expressive range of a level generator. In *Proceedings of the 2010 workshop on procedural content generation in games* (pp. 1–7). — Cited on pages 13, 37, and 50.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, *8*(2), 131–162. — Cited on pages 11 and 21.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*(2), 99–127. — Cited on page 11.

Sudhakaran, S., Grbic, D., Li, S., Katona, A., Najarro, E., Glanois, C., & Risi, S. (2021). Growing 3d artefacts and functional machines with neural cellular automata. *arXiv preprint arXiv:2103.08737*. — Cited on page 17.

Taylor, T. (2019). Evolutionary innovations and where to find them: Routes to open-ended evolution in natural and artificial systems. *Artificial life*, *25*(2), 207–224. — Cited on pages 7, 8, 23, 24, and 55.

Taylor, T. J. (1999). From artificial evolution to artificial life.
— Cited on page 7.

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 172–186. — Cited on pages 9 and 10.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., & Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, *11*(12). — Cited on pages 12 and 26.

Wiggins, G. A., Papadopoulos, G., Phon-Amnuaisuk, S., & Tuson, A. (1998). *Evolutionary methods for musical composition.* University of Edinburgh, Department of Artificial Intelligence. — Cited on page 5.

Yannakakis, G. N., Liapis, A., & Alexopoulos, C. (2014). Mixed-initiative co-creativity.
— Cited on pages 2 and 6.

Yannakakis, G. N., & Togelius, J. (2018). *Artificial intelligence and games* (Vol. 2). Springer.
— Cited on page 20.