Reconstructing Existing Levels through Level Inpainting

Johor Jara Gonzalez, Mathew Guzdial

Computing Science Department, Alberta Machine Intelligence Institute
University of Alberta
jaragonz@ualberta.ca, guzdial@ualberta.ca

Abstract

Procedural Content Generation (PCG) and Procedural Content Generation via Machine Learning (PCGML) have been used in prior work for generating levels in various games. This paper introduces Content Augmentation and focuses on the subproblem of level inpainting, which involves reconstructing and extending video game levels. Drawing inspiration from image inpainting, we adapt two techniques from this domain to address our specific use case. We present two approaches for level inpainting: an Autoencoder and a U-net. Through a comprehensive case study, we demonstrate their superior performance compared to a baseline method and discuss their relative merits. Furthermore, we provide a practical demonstration of both approaches for the level inpainting task and offer insights into potential directions for future research.

Introduction

Procedural Content Generation (PCG) (Hendrikx et al. 2013) and Procedural Content Generation via Machine Learning (PCGML) (Summerville et al. 2018), cover techniques used to generate different types of content, such as "levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles, characters, etc" (Shaker, Togelius, and Nelson 2016). While PCG requires hand-authored knowledge from a developer, PCGML instead extracts this knowledge from existing content. One problem with PCGML is the question of where it slots into the game development process. Early in a game's development there won't be sufficient content to train a model. However, if we wait until the end of the development process then there's no point in applying PCGML, as all the content has already been produced. But if we consider instead not helping with the initial development of the game, but applying PCGML to extend or augment an already completed game, we might sidestep this issue.

We establish a novel PCG problem called Content Augmentation (CA), the main characteristic of CA is that it extends original content. CA might, for example, be used in the generation of DLC, mods, or updates in the case of a live service game. We'll discuss CA applications further in

the future work section at the end of the paper. The difference between CA and other applications of PCG is that they tend to focus on generating content from scratch, based on user input or hand authored rules. In comparison, CA must take into account the existing work that it seeks to extend or augment.

Many branches of PCG have some overlap with CA. Classical PCG approaches like search or grammars could be applied to CA. But these approaches do not traditionally focus on creating content based on some designer's prior work (Shaker, Togelius, and Nelson 2016). PCGML techniques have attempted to learn to emulate a particular designer, however this has thus far been focused on generating whole new pieces of content with that same style, rather than extending existing content (Summerville et al. 2018).

Some co-creative systems (Deterding et al. 2017), where a human works with a PCG system to produce content, do attempt to extend or modify output from a human, but typically as part of an initial design process. In comparison, CA focuses on taking complete, output content and extending it after it has been published or released. PCGML techniques have been proposed to try to generate whole new games, a problem sometimes called automated game generation (Snodgrass and Sarkar 2020a; Guzdial and Riedl 2018). This is related to CA, in terms of taking as input existing games, but we focus on augmenting or extending game content instead of generating whole new games from scratch.

Rather than attempting to solve CA as a single problem, we identify and attempt to solve an initial CA subproblem that we call level inpainting. In level inpainting we focus on reconstructing the existing structures in a game. Level inpainting is directly inspired by image inpainting (Zeng et al. 2020), a computer vision problem domain focused on reconstructing missing or damaged structures in images. Similarly, we aim to train a model to reconstruct damaged or missing structures in levels. While prior work has included models capable of repairing levels (Jain et al. 2016), it has focused on making changes to the levels to make them playable. Instead, we focus on learning the style of a set of game levels and then filling in missing, "masked out" sections of other levels. Our future goal is to use these models to expand levels or to create new level structure that can connect disparate areas as part of a larger CA system.

As we mentioned above, for level inpainting we take in-

spiration from image inpainting (Elharrouss et al. 2020). "Image inpainting is a task of reconstructing missing regions in an image" (Zeng et al. 2020). We adapted two existing image inpainting techniques to our level inpainting task, changing them to model level structure instead of pixels. While there has been a great deal of prior image inpainting work (Zeng et al. 2020), it is not the case that we can just adapt them to game content naively due to a number of factors. For example, unlike images, game content must have certain characteristics to be valid (e.g. playable for levels).

In this paper, we introduce level inpainting: the process of fixing or restoring missing parts of a level. We consider this to be an initial exploration of Content Augmentation (CA). We validate our approach on existing game levels, by comparing the output (inpainted level) and the original level. As an initial exploration, we adapt two existing image inpainting approaches to level inpainting.

This paper includes the following contributions:

- We propose Content Augmentation (CA) as a PCG problem.

- We propose level inpainting as a CA subproblem.

- We present an approach to process an existing dataset for level inpainting.

- We present two modifications of image inpainting architectures adapted to level inpainting.

- We present our results in comparison to a traditional PCGML baseline.

Related Work

We imagine that classical PCG approaches like search-based and constructive PCG could be applied to Content Augmentation (CA) (Shaker, Togelius, and Nelson 2016). However, in this paper we focus on Procedural Content Generation via Machine Learning (PCGML) (Summerville et al. 2018), as we argue that CA can serve as a natural way to include PCGML in the game development process.

Super Mario Bros. has been one of the most common problem domains in PCGML. From generating levels (Summerville and Mateas 2016), blending levels (Sarkar and Cooper 2021), reinforcement learning (Shu, Liu, and Yannakakis 2021), or to studying new representations for level generation (Jadhav and Guzdial 2021). There are different factors that make *Super Mario* levels useful to test different approaches. One factor is that it is a finished product, for this reason the outputs of the models are easy to compare with the original content. Another factor is the popularity of the game, Mario is well recognized and the majority of people know how it is played. Because of these reasons, we decided to employ Mario as the problem domain for this initial experiment into level inpainting. The fact that we have the original levels for comparison will allow us to evaluate our approach quantitatively. Further, we hope that Mario's well-known design style will help clarify the problem of level inpainting to readers.

Wave Function Collapse (WFC)(Kim et al. 2019) is one particular PCGML algorithm. WFC focuses on extracting local patterns and then using the extracted patterns to incrementally generate an output that, in our opinion, is surprisingly consistent with the original author's style. Researchers have previously explored the generation of level designs using WFC. Some of those works include Sandhu et. al. (Sandhu, Chen, and McCoy 2019) who add design constraints to WFC in order to use them at runtime to design levels. Similarly Kim et. al. (Kim et al. 2019) add more constraints to the original WFC approach in order to create multi-layer levels. In order to use WFC some works like Cheng et. al. (Cheng, Han, and Fei 2020) use a graph-based input to generate more content. Due to it being based on an incremental improvement of an initially random piece of content, one might consider WFC to be a natural fit for level inpainting. We do not draw on WFC in this work, as it is the initial exploration of level inpainting and so we instead focus on adapting image inpainting models. However, we hope to explore WFC for level inpainting in future work.

Recently, diffusion-based approaches have achieved great success in generating images (Nichol et al. 2021). Diffusion works by essentially learning to reverse a function that converts structured data (such as images) into noise (Song and Ermon 2019). Recently, Siper et al. (Siper, Khalifa, and Togelius 2022) introduced a similar idea, but applied it to level generation. Similar to WFC, this style of approach iteratively converts an apparently random initial state into structured output, making it a good fit for level inpainting. To our knowledge, there are no existing PCGML implementations for Content Augmentation. The closest prior work has attempted to generate new content for new games with PCGML (Sarkar and Cooper 2018; Guzdial and Riedl 2018; Snodgrass and Sarkar 2020b). For this work, in comparison to this prior work, we aim to train a model on only one domain. In addition, we are not trying to create new content from scratch, instead we are focusing on reconstructing areas of the same levels using observed similarities with the rest of the map. We use this level inpainting as a proxy for extending existing levels, which we also demonstrate below as an example.

Level repair is a related PCG problem that focuses on altering levels such that the level matches some mechanical (e.g. playable) or aesthetic requirements (Jain et al. 2016; Cooper and Sarkar 2020; Zhang et al. 2020). Jain et al. demonstrated level repair by mapping some unplayable section of a game level to the closest playable level section (Jain et al. 2016). In comparison, Cooper and Sarkar employed an agent-based approach, which attempted to pathfind across a level and could alter the level to make it playable (Cooper and Sarkar 2020). In another approach, Zhang et al. used generative adversarial networks to "generate-then-repair" using hand-author maps to train the model, then generate new maps that may be unplayable and repair them (Zhang et al. 2020). These approaches were altering content, but for the purpose of level playability, not to match a style or replace an undefined section.

Image inpainting (Elharrouss et al. 2020) uses different image processing techniques to reconstruct damaged areas of an image, a variety of approaches have been explored including Encoder-decoders (Liu et al. 2019), CNNs (Liu

et al. 2019), and GANs (Demir and Unal 2018). The principal characteristic of image inpainting is that in order to reconstruct a damaged image it uses nearby pixels to the corrupted part to achieve coherence with the reconstructed image. In our research we modify two image inpainting architectures, so that instead of working with pixels they work with tiles. The goal of level inpainting then is to fill in damaged or missing tiles in order to achieve coherence with the entire level.

System Overview

To reach our goal of level inpainting, we cannot directly use the architectures of image inpainting models because they are made to reconstruct pixels in an image. Given that we want to capture the same style of the original game, we need to adapt an image inpainting architecture to our needs. In order to get a functional system first we need a level inpainting dataset we can use to train our models. To get a level inpainting dataset, we make use of an existing tile-based representation (Summerville et al. 2016), but post-process the data to fit the requirements of our level inpainting problem. Finally, we train two models adapted from image inpainting to level inpainting: an autoencoder and a U-net.

Dataset

For image inpainting, the datasets consist of images. However in our approach, using only images of the level will be impossible, as if we use pixels we cannot load the results into a game engine. Alternatively, we could post-process the output of our model to convert from pixels to tiles, but this would create undue complexity.

For our dataset we make use of the Video Game Level Corpus (VGLC) (Summerville et al. 2016). The VGLC contains a tile representation of levels that can be easily adapted for our purpose of level inpainting. Our goal for level inpainting is to extend an author's design with the same style. We wanted to use a set of levels with a clear design vision. With this in mind, we chose to use the tile representation from *Super Mario Bros.* and *Super Mario Bros. 2* (the Japanese version), given that both games were made by the same designer. We split the dataset into 26 levels for training and 6 levels for testing, selecting the longest levels across the dataset for the test set (From Super Mario: 3-1, 6-2, 8-1, and from Super Mario 2: 1-1, 5-2). We chose to use whole levels for our test set as this will let us investigate whether our models have learned the level design style, not just replicating a particular level's geometry. If we did not use whole levels but had the same level spread across our train and test sets, then the model would see the same patterns across both sets. With this mind, the selection of the longest levels gave us the most opportunities to test how our models performed on different structures.

Data Processing

For our data processing first we represent each level as a matrix of the same size as the level. The information inside of the matrix is based on a one-hot encoding representing the tile at that position. The one-hot encoding used is of length
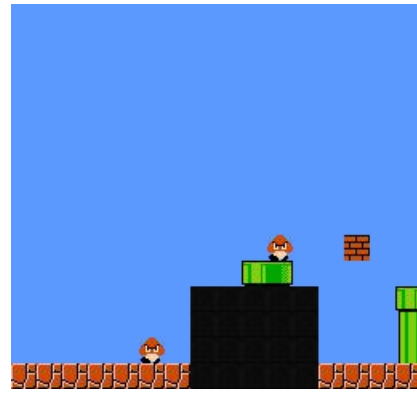

Figure 1: Size of the mask in one example. We note that not all parts of the pipe are fully covered, which requires the model to reconstruct the missing tiles.

13 given the tiles representation for *Super Mario Bros.* in the VGLC (e.g. brick, emptyBlock, enemy,topLeftPipe, leftPipe, etc.). The level can thus be represented as a matrix with a height of 16, a length of the size of the level, and a depth of 13 for the one-hot encoded representation. Then, in order to get our final training data we split each matrix into submatrices of size 16x16x13. With this step we get around the varying width of the levels. The representation for the sub matrices $16 \times 16 \times D$ where D is the dimension of a one-hot encoding is a common representation in PCGML (Yang, Sarkar, and Cooper 2020).

Once we have all the submatrices processed, we introduce a mask area for each submatrix. We inherit the concept of a mask from image inpainting (Zeng et al. 2020). In the area of the mask, we delete the information of the tiles and replace it with all zeros. The mask has a size of 5x4 and we produced 11 masked inputs for every submatrix, by placing the 5x4 mask along each possible position at the bottom of the submatrix. We chose this setup for our mask for several reasons. First, it has a reasonable size that allows it to at least partially cover common structures like pipes and stairs. Second, we place the mask along the bottom of the level to be able to cover these structures. Third, in the game *Super Mario Bros.* the majority of structures and movement are located along the bottom of the level. In Figure 1 we can demonstrate an example of the mask, and the area that the model will predict. We note that finding useful shapes and positions for masks is an open resource problem for image inpainting (Zeng et al. 2020), and so the same will likely be true for level inpainting.

Model Architecture

In this section we discuss how we adapted two image inpainting architectures for level inpainting. The first architecture is based on a convolutional autoencoder (Zhang 2018), as it is a basic architectures for image processing, including image inpainting. Autoencoders have been commonly applied in PCGML (Thakkar et al. 2019; Snodgrass and Sarkar 2020a; Sarkar and Cooper 2021), as they tend to be less data-hungry than other models. The second architecture proposed
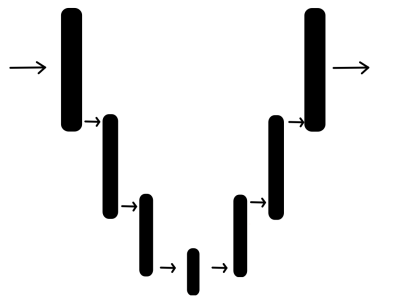
a) Convolutional autoencoder architecture
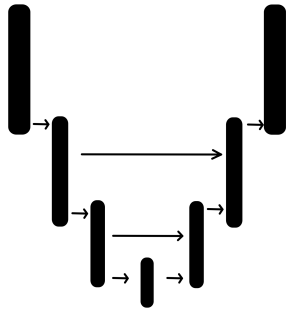
b) U-net architecture

Figure 2: Our Two proposed models. a): the convolutional autoencoder. b): the U-net. Layers of ([16x16x13],[16x16x16],[8x8x32],[8x8x64]).

is a U-net (Ronneberger, Fischer, and Brox 2015) architecture, which was designed explicitly for image inpainting. To have a fair comparison between our alternative models we used the same number of convolutional layers for both models. However, the U-net architecture was specifically designed to use small amounts of data, which is often the case for PCGML applications (Summerville et al. 2018).

Our architectures are illustrated in Figure 2, both contain two paths: an encoder path with layers shown in that figure and a decoder path with the same layers but mirrored. Figure 2a is a relatively simple convolutional autoencoder, but this is appropriate for the problem and amount of training data, as supported by prior PCGML approaches using autoencoders. Figure 2b demonstrates the U-net with the same layers, which supplements the encoding and decoding paths by propagating context to higher resolution layers.

In order to improve our architectures, we opted to use transpose convolution layers. We found they outperformed the up-conv layers typically used in image inpainting according to our experiments. We anticipate that this is because the extra convolution made by the transpose gave us clearer results, which may be due to the differences in level and tile inpainting. In image inpainting, being a single pixel off won't have a large impact in comparison to being a single tile off in level inpainting. We used binary cross entropy as a loss function for both models. We used the Adam optimizer with a learning rate of 0.0001, with a batch size of 10 (using as reference the quantity of submatrices generated in

our dataset), and trained both models until convergence.

Evaluation

Since our final goal is to apply level inpainting to Content Augmentation (CA), for our evaluation we would ideally want to create augmentations to Mario levels and then see how players react to these augmentations. But for an initial exploration of this task, we're instead focused on an approximation of this problem where we know the ground truth. We have our models attempt to recreate existing Mario level structure and then compare their performance to a baseline. A good result would be to perfectly reproduce the area of the mask. In our evaluation we only focus on the interior of the mask, because we are not concerned with representational ability generally. We only care about a level inapinting model's ability to do the required level inpainting task.

We use three different metrics in our evaluation:

1. **Tile-by-Tile (TbyT)**: First, we compare tile-by-tile accuracy. This is a standard accuracy measure used in prior PCGML work (Summerville et al. 2018). Essentially, we are measuring how close the output of the approach is to the original considering all tiles.

2. **No Sky (NoSky)**: The second metric is again a tile-by-tile comparison but we exclude the sky tiles. Since the majority of the samples contains sky tiles a simple strategy would be to always predict sky tiles. So here we measure the non-sky accuracy.

3. **Structures (Struct.)**: The third metric, is based on the tile accuracy for the pipe and stair tiles only. This is due to the fact that these structures are some of the more complicated parts of *Super Mario Bros.* levels to reconstruct, and because they are used to test the mechanics of the game (e.g. jumping, running, etc.). Further, checking for "broken" pipes is a common PCGML metric (Summerville and Mateas 2016; Snodgrass and Ontañón 2014), which this approximates.

Baseline

In order to better understand the results for the convolutional autoencoder and the U-net, we implement a Markov chain baseline. We chose a Markov baseline because it is one of the most common and oldest PCGML approaches for *Super Mario Bros.* levels (Snodgrass and Ontañón 2016). In addition, given that a Markov chain was the first PCGML approach for level generation (Snodgrass and Ontanón 2013) it make sense to apply it on this new task of level inpainting. We use the original formulation of the Markov Chain from (Snodgrass and Ontanón 2013) trained on the same levels that make up the training set for our two models. During testing, we query the model to fill in only the masked section of the input.

Results

We include the average and standard deviation values of our three metrics in Table 1 for the convolutional autoencoder (AE), the U-net (UNet), and the Markov approach (Markov). Each row represents a particular model's results for one of
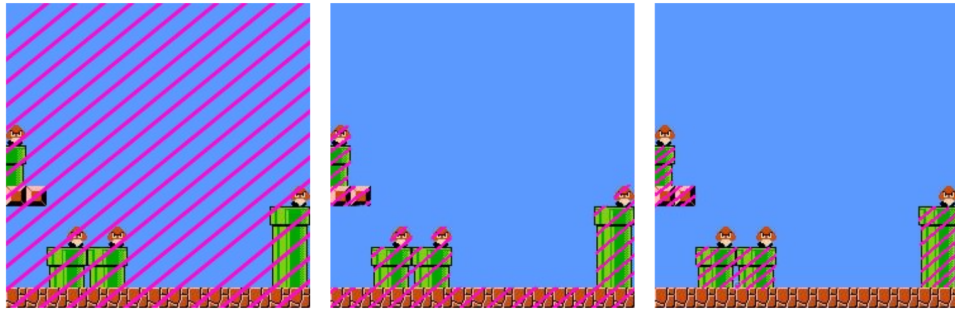
Figure 3: A visualization of our three metrics: the Tile-by-Tile (Left), No Sky (Middle), and Structures (Right).

Metrics	SM1-Level 3-1	SM1-Level 4-2	SM1-Level 6-2	SM1-Level 8-1	SM2-Level 1-1	SM2-Level 5-2	Avg
AE-TbyT	**91.86 ± 0.42**	**84.07 ± 0.94**	**88.65 ± 0.71**	**92.02 ± 0.23**	**91.82 ± 0.25**	**79.49 ± 0.90**	**88.03**
UNet-TbyT	88.44 ± 1.12	78.75 ± 1.12	87.13 ± 0.67	91.04 ± 0.21	91.20 ± 0.30	75.80 ± 0.80	85.39
Markov-TbyT	75.44 ± 5.45	67.58 ± 4.58	71.21 ± 2.56	80.23 ± 2.56	79.88 ± 2.57	47.36 ± 9.40	58.24
AE-NoSky	84.75 ± 0.88	**74.32 ± 2.52**	**77.68 ± 2.52**	84.24 ± 0.792	**84.36 ± 0.73**	**63.98 ± 2.87**	**78.10**
UNet-NoSky	**87.20 ± 0.93**	66.36 ± 1.40	75.97 ± 1.40	**85.73 ± 0.66**	83.93 ± 1.00	54.76 ± 1.42	75.65
Markov-NoSky	53.17 ± 3.10	41.76 ± 4.77	48.29 ± 2.67	60.89 ± 0.93	57.73 ± 10.42	35.14 ± 2.86	49.49
AE-Struct	82.98 ± 2.87	79.24 ± 2.61	56.43 ± 2.07	83.44 ± 2.65	**79.60 ± 1.31**	**57.30 ± 6.05**	73.16
UNet-Struct	**88.31 ± 0.57**	**80.54 ± 1.54**	**59.76 ± 1.30**	**87.70 ± 1.42**	76.76 ± 1.89	48.19 ± 3.37	**73.54**
Markov-Struct	0	1.53 ± 0.10	2.74 ± 0.22	0	0	0	0.71

Table 1: This table contains the the average and standard deviation values for our three different metrics across our two approaches and baseline. All three of the metrics are based on a per-tile accuracy, but focus on increasingly particular tile types.

our three different metrics. For each column we have each test level, and then a final column giving the average over these values.

In the first three rows, we can see the results of our different implementations evaluated using the Tile-by-Tile (TbyT) metric. Of these three implementations we can see that the convolutional autoencoder and the U-net achieve similar results compared to the Markov implementation. In addition, the convolutional autoencoder consistently outperforms the U-net. We anticipate this was due to the convolutional autoencoder being a simpler model than the U-net, and so it did a slightly better job generalizing over the same data.

In the middle three rows of Table 1 we can see the results using the No Sky metric (NoSky), in which we exclude the sky tiles. All models did slightly worse in this case, indicating an over-reliance on predicting sky tiles. In these results we can see that the autoencoder again had a slight advantage in the reconstruction accuracy.

For our last three rows, we used the Structures (Struct.) metric. With this metric we check the tile accuracy only in terms of pipes and stairs. As we can see the U-net ended up with a small advantage compared to the autoencoder. We anticipate this is because the U-net is better at reproducing less common structure (Ronneberger, Fischer, and Brox 2015). In comparison, the Markov chain had the worst results, unable to produce any pipes or stairs for most levels. It consistently performed worse compared to the other models, suggesting that even techniques that work well for level generation may not be suitable for content augmentation or specifically level inpainting.

In Figure 4 we include some of the output generated by

the autoencoder and the U-net in comparison to the original VGLC dataset. All of these examples were taken from the test set. The performance is close, which matches the quantitative results in the table. However, the U-net seems to have done a slightly better job at capturing the location of the more unique tiles, such as the stairs in the first two examples and the question mark block in the third example.

Case Study

In the last section we demonstrated that a convolutional autoencoder outperforms a U-net for almost all the metrics and both models consistently outperform a baseline Markov chain. We take this to indicate that both of these approaches can successfully be applied for level inpainting. However, we did not show it applied to our high-level task of Content Augmentation (CA) due to the lack of an ability to quantitatively evaluate any output.

As an illustrative example of this initial approach, we demonstrate what one version of augmenting a level might look like in Figure 5. In this figure we present the original map of the classic *Super Mario Bros.* 5-1 and an updated version where the content has been augmented. We did this manually by selecting areas with low structural content (just sky and ground) and applying a mask size where new structures were placed by using the U-net model. One could imagine this as the output of a co-creative level inpainting tool or a "version 2.0" of the level output by a more autonomous tool.
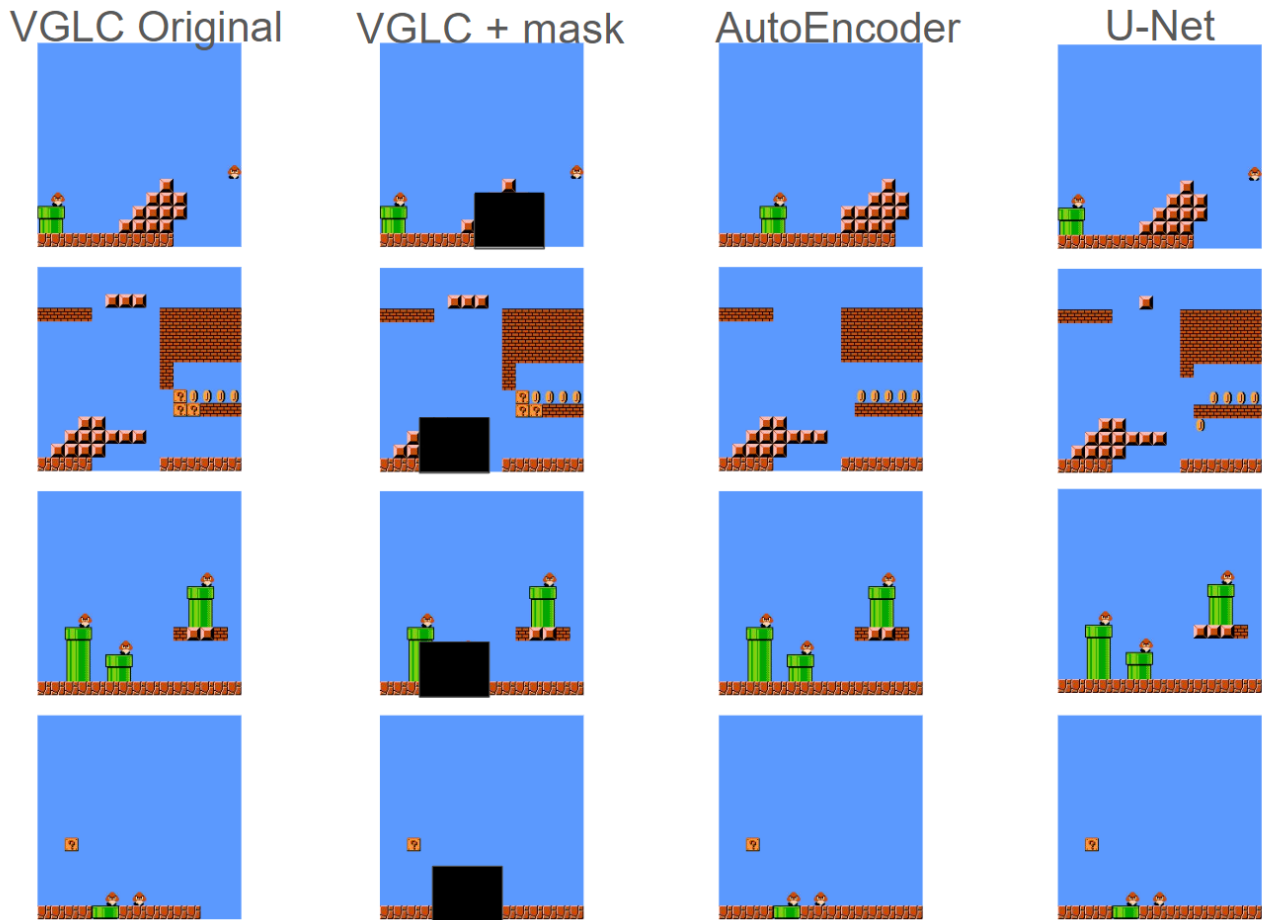
Figure 4: In this image, we include original level sections, one of the masks added to the level section, and the output of the autoencoder and the U-net. These examples were selected randomly from the test dataset.

Limitations

We acknowledge that our approach taken in this paper has a number of limitations. For example, the VGLC dataset does not contain all the underground levels or the castle levels from the original games. Consequently, it is impossible to reconstruct these types of structures for our model. Other limitations to consider are the position and shape of the mask. Given that there are levels where the structures are not only along the bottom of the level, it is not possible to evaluate the reconstruction of some structures. One way to address this issue could be to have the mask follow the player's path. With this, we might end up with a model that can better reproduce structures important to players, similar to the finding from Summerville and Mateas (Summerville and Mateas 2016). Despite these limitations, we consider that our results are still valid for the following reasons. First, the mask that we implemented covered the majority of the structures at the bottom of the levels. Second, most of the levels had content along the bottom. Finally, our models showed a reasonable reconstruction of the level structures.

For this initial exploration of level inpainting we adapted two models previously used for image inpainting. While both models outperformed a PCGML baseline, their quantitative performance and qualitative performance (as demonstrated in Figure 5 could both be improved. Though we feel this is appropriate for an initial exploration, there are definitely avenues to address this limitation through future work.

Future Work

For future work, we plan to focus on different aspects of Content Augmentation (CA). There are a number of different options, with one being to improve the quality of level inpainting. One possible way to improve level inpainting would be to continue trying to adapt different systems from image inpainting. Image inpainting has a significant amount of prior work focused on finding an optimal mask shape. Similarly, we could explore the choice of mask shape in more detail. As an alternative to focusing on adapting image inpainting models to level inpainting, we could adapt additional PCGML models for level generation to this task. For example, WaveFunctionCollapse (WFC) (Sandhu, Chen, and McCoy 2019), which could be adapted to level
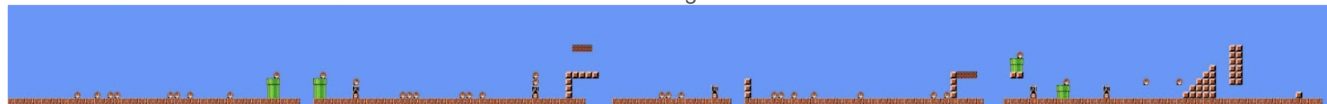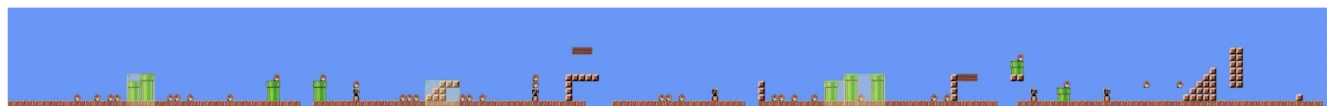
Figure 5: Example based on *Super Mario Bros.* 5-1, at the top we include the original level from VGLC. At the bottom, we present a new version produced with Content Augmentation in the yellow areas, applying only new structures with level inpainting.

inpainting given the way it iteratively generates output content. The key would be to treat the masked out content as yet-ungenerated content, and the areas around the mask as content previously generated by WFC. Though it may be necessary to handle cases where the existing contradicts the constraints WFC has extracted from existing levels.

Outside of improving level inpainting we hope to explore other aspects of CA. In particular, aspects related to game mechanics. The concept would be to use CA to generate additional abilities, items, or enemies for an existing game. The goal would be to get output similar to what one might find in human-authored DLC, mods, or updates in a live service game. Other possible options might be to focus on particular level design applications in CA. For example, merging different levels via level inpainting, or identifying the best locations in a map to add new content while keeping a coherent game design. In the future, we hope that CA can lead to systems to automatically extend existing games, producing DLC, mods, or entirely new kinds of game designs.

Conclusions

We presented a new PCG problem, Content Augmentation (CA), and how it relates to classical PCG and PCGML. We introduced a subproblem of CA: level inpainting, where the main idea is to reconstruct missing information in a level. Our experiments demonstrate that image inpainting architectures can be adapted for level inpainting. Despite this, these adaptations seem to require more modifications to improve their performance for CA. Given these initial positive results, we plan to continue this line of research towards generation of novel CA in the future.

Acknowledgements

References

Cheng, D.; Han, H.; and Fei, G. 2020. Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm. In *International Conference on Entertainment Computing*, 37–50. Springer.

Cooper, S.; and Sarkar, A. 2020. Pathfinding Agents for Platformer Level Repair. In *AIIDE Workshops*.

Demir, U.; and Unal, G. 2018. Patch-based image inpainting with generative adversarial networks. *arXiv preprint arXiv:1803.07422*.

Deterding, S.; Hook, J.; Fiebrink, R.; Gillies, M.; Gow, J.; Akten, M.; Smith, G.; Liapis, A.; and Compton, K. 2017. Mixed-initiative creative interfaces. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 628–635.

Elharrouss, O.; Almaadeed, N.; Al-Maadeed, S.; and Akbari, Y. 2020. Image inpainting: A review. *Neural Processing Letters*, 51: 2007–2028.

Guzdial, M.; and Riedl, M. 2018. Automated game design via conceptual expansion. *Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2018*, 31–37.

Hendrikx, M.; Meijer, S.; Van Der Velden, J.; and Iosup, A. 2013. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1): 1–22.

Jadhav, M.; and Guzdial, M. 2021. Tile embedding: a general representation for level generation. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 17, 34–41.

Jain, R.; Isaksen, A.; Holmgård, C.; and Togelius, J. 2016. Autoencoders for level generation, repair, and recognition. In *Proceedings of the ICCC workshop on computational creativity and games*, volume 9.

Kim, H.; Lee, S.; Lee, H.; Hahn, T.; and Kang, S. 2019. Automatic generation of game content using a graph-based wave function collapse algorithm. In *2019 IEEE Conference on Games (CoG)*, 1–4. IEEE.

Liu, H.; Jiang, B.; Xiao, Y.; and Yang, C. 2019. Coherent semantic attention for image inpainting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 4170–4179.

Nichol, A.; Dhariwal, P.; Ramesh, A.; Shyam, P.; Mishkin, P.; McGrew, B.; Sutskever, I.; and Chen, M. 2021. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 234–241. Springer.

Sandhu, A.; Chen, Z.; and McCoy, J. 2019. Enhancing wave function collapse with design-level constraints. *ACM International Conference Proceeding Series*.

Sarkar, A.; and Cooper, S. 2018. Blending levels from different games using LSTMs. *CEUR Workshop Proceedings*, 2282.

Sarkar, A.; and Cooper, S. 2021. Generating and blending game levels via quality-diversity in the latent space of a variational autoencoder. In *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*, 1–11.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural content generation in games*. Springer.

Shu, T.; Liu, J.; and Yannakakis, G. N. 2021. Experience-driven PCG via reinforcement learning: A Super Mario Bros study. In *2021 IEEE Conference on Games (CoG)*, 1–9. IEEE.

Siper, M.; Khalifa, A.; and Togelius, J. 2022. Path of Destruction: Learning an Iterative Level Generator Using a Small Dataset. *arXiv preprint arXiv:2202.10184*.

Snodgrass, S.; and Ontanón, S. 2013. Generating maps using markov chains. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Snodgrass, S.; and Ontañón, S. 2014. Experiments in map generation using Markov chains. In *FDG*.

Snodgrass, S.; and Ontanón, S. 2016. Controllable Procedural Content Generation via Constrained Multi-Dimensional Markov Chain Sampling. In *IJCAI*, 780–786.

Snodgrass, S.; and Sarkar, A. 2020a. Multi-domain level generation and blending with sketches via example-driven bsp and variational autoencoders. In *International Conference on the Foundations of Digital Games*, 1–11.

Snodgrass, S.; and Sarkar, A. 2020b. Multi-Domain Level Generation and Blending with Sketches via Example-Driven BSP and Variational Autoencoders. *ACM International Conference Proceeding Series*.

Song, Y.; and Ermon, S. 2019. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32.

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

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games*, 10(3): 257–270.

Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Ontanón, S. 2016. The vglc: The video game level corpus. *arXiv preprint arXiv:1606.07487*.

Thakkar, S.; Cao, C.; Wang, L.; Choi, T. J.; and Togelius, J. 2019. Autoencoder and evolutionary algorithm for level generation in lode runner. In *2019 IEEE Conference on Games (CoG)*, 1–4. IEEE.

Yang, Z.; Sarkar, A.; and Cooper, S. 2020. Game level clustering and generation using Gaussian mixture VAEs. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, 137–143.

Zeng, Y.; Lin, Z.; Yang, J.; Zhang, J.; Shechtman, E.; and Lu, H. 2020. High-resolution image inpainting with iterative confidence feedback and guided upsampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16*, 1–17. Springer.

Zhang, H.; Fontaine, M.; Hoover, A.; Togelius, J.; Dilkina, B.; and Nikolaidis, S. 2020. Video game level repair via mixed integer linear programming. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, 151–158.

Zhang, Y. 2018. A better autoencoder for image: Convolutional autoencoder. In *ICONIP17-DCEC. Available online: http://users. cecs. anu. edu. au/Tom. Gedeon/conf/ABCs2018/paper/ABCs2018_paper_58. pdf (accessed on 23 March 2017)*.