Image-to-Level: Generation and Repair

Eugene Chen, Christoph Sydora, Brad Burega, Anmol Mahajan, Abdullah, Matthew Gallivan, Matthew Guzdial
University of Alberta, Department of Computing Science
{echen, csydora, burega, mahajan, ab11, mbgalliv, guzdial}@ualberta.ca

Abstract

Procedural content generation via machine learning (PCGML) has recently gained research attention due to its ability to generate new game content with minimal user input. However, thus far those without machine learning expertise have been largely unable to use PCGML to generate content to fit their needs. This paper proposes the use of images as the input for a PCGML process to generate game levels. Intuitively, a user can submit an image, with the system returning the closest valid Super Mario Bros. game level. Our results indicate that at least for domains like Super Mario Bros. we can recreate a target level with high fidelity.

Introduction

Procedural content generation (PCG) is the algorithmic process to generate game content. In practice, creating acceptably coherent and engaging content using PCG requires time and expertise, with no guarantee of success (Schreier 2017). To help reduce the amount of human input required for PCG, some turn to procedural content generation via machine learning (PCGML) (Summerville et al. 2017), which trains ML models on existing content to generate new content. However, content generation using PCGML currently requires deep domain expertise in ML and existing data.

To address the expertise problem, PCGML mixed-initiative systems have been created. These systems allow users to interact with and apply PCGML without ML expertise (Guzdial et al. 2019; Schrum et al. 2020). However, they still require a fairly high degree of effort to use effectively. For example, prior approaches require users to label existing data (Guzdial et al. 2018), to learn the impacts of altering certain parameters (Schrum et al. 2020) or to implicitly train an ML agent (Guzdial et al. 2019). These requirements serve as barriers to more widespread adoption of mixed-initiative systems (Lai, Latham, and Leymarie 2020).

One approach to level design begins with first sketching out a level. For example, the original levels of Super Mario Bros. were sketched out on graph paper (Murphy 2015). Inspired by this, we developed a PCGML mixed-initiative system called Image-to-Level that translates an input image into a game level.[1] This system allows for a simple interaction with a PCGML model, rather than having to learn to use a more iterative approach. One could anticipate that a direct conversion of an image to a game level might not accurately reflect the aesthetics and mechanical requirements of the game. As such, we employ a two step process, first converting an image into a rough level and then "repairing" the level to make changes so that the style of the level is more similar to the original game's levels. In this paper we use the running example of Super Mario Bros. as it is a well-understood PCG research domain, however the system can also produce Lode Runner levels.

Related Works

We overview related work on three major categories: PCGML, co-creativity in PCGML, and computer vision as it intersects with PCG. We include the last of these as our system can be thought of as a type of image processing.

PCGML

Procedural content generation via machine learning (PCGML) describes the use of machine learning in the creation of video game content (Summerville et al. 2017). PCGML attempts to generate novel game content based on existing examples. We focus on PCGML approaches to generate game levels.

Our system employs a convolutional neural network (CNN), Markov chain, and autoencoder. Markov chains have been frequently applied to Super Mario Bros. and a number of other games (Snodgrass and Ontanón 2016), we specifically draw on Markov Chains due to their ability to capture local coherency. Autoencoders and CNNs have also seen a good deal of use in Super Mario Bros. (Guzdial et al. 2018; Sarkar, Yang, and Cooper 2020), we draw on these neural networks as they tend to perform well with image-like data. All of these prior approaches generate levels from input random noise, instead we make use of images.

One common method in PCGML approaches is to include an explicit repair step due to the noisy nature of ML-based generation. Jain et al. (Jain et al. 2016) used a denoising autoencoder to repair unplayable level sections and to classify levels into different styles. Mott et al. (Mott, Nandi, and

[1]The tool is available at http://ImageToLevel.com

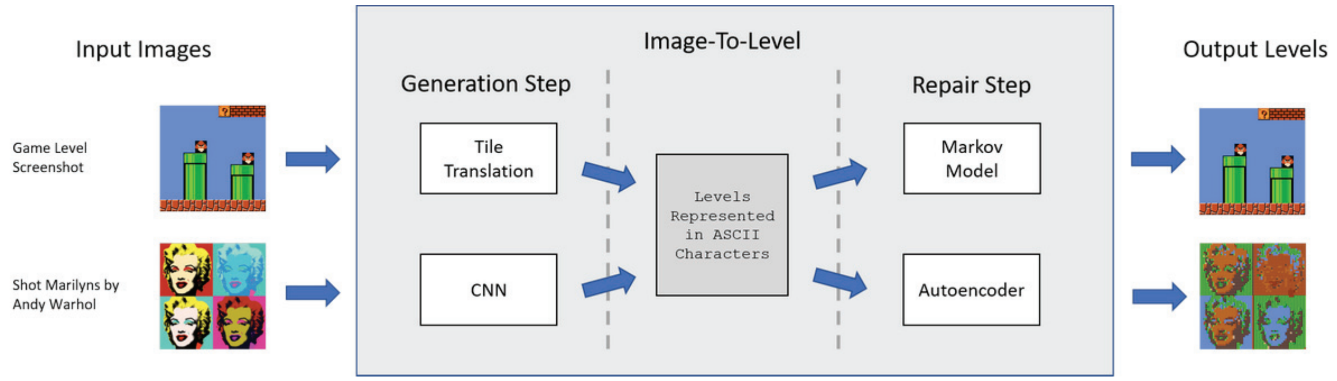

Figure 1: The Image-to-Level System at a Glance.

Zeller 2019) used a long-short term memory recurrent neural network (LSTM) to repair locally incoherent sections of generated levels. Shu et al. (Shu et al. 2020) developed a method for repairing errors in pipe tile placement in Super Mario Bros. with an evolutionary algorithm. Our work includes a repair step, with two options: an autoencoder and a Markov chain. Our repair step differs from this existing work as it repairs the results of translating an image into a level instead of repairing a level generated from scratch.

The closest prior PCGML work to ours is Rebouças et al.'s model for generating game sprites from sketches (Rebouças Serpa and Formico Rodrigues 2019). They employed a Generative Adversarial Neural Network (GAN) to turn sketches into the animation states of game entities. However, this can be understood as a refinement of the input, whereas our approach requires a translation from an image representation (pixels) into a level representation (tiles).

Co-Creativity

Co-creative systems are ones where humans and AI agents work together, they are also referred to as mixed-initiative systems (Liapis, Smith, and Shaker 2016). A variety of co-creativity approaches exist but most of these approaches rely upon search-based or constructive PCG methods (Smith, Whitehead, and Mateas 2010; Baldwin et al. 2017).

There have been some examples of PCGML co-creative systems. Guzdial et al. (Guzdial et al. 2019) employed a Deep Reinforcement Learning agent to collaborate with a user to produce a Super Mario Bros. level, acting as an equal partner. Schrum et al. (Schrum et al. 2020) employed a GAN trained on Super Mario Bros. and allowed users to explore the GAN's learned latent space. Machado et al. (Machado et al. 2019) employed an AI agent with learned probabilities to suggest different game components to add to a game. All of these systems involve a prolonged series of interactions with a user. In comparison, Image-to-Level accepts user input in the form of an image, but it is a much more lightweight system, requiring only a single interaction. We chose co-creativity over other similar terms (e.g. mixed initiative and tool vs. designer (agent)) because of the large contribution of the AI system to the final level and how it can be used by a human designer in an iterative fashion.

Vision in Games

Computer vision represents the field of research into and techniques for automated image processing (Forsyth and Ponce 2002). Image-to-Level can be considered a specialized example of image processing. To the best of our knowledge it represents the first instance of employing AI to convert an image directly into a game level.

Computer vision does not frequently intersect directly with level generation, but there have been some instances of prior work. Guzdial et al. (Guzdial et al. 2017) used CNNs for visual procedural content generation, recolouring an existing level. However, their work does not directly impact level structure. Snodgrass (Snodgrass 2018) outlined an approach to automatically identify tiles from images of Super Mario Bros. levels, converting these images to a tile-representation. This work differs from our own as it generates a unique tileset for a given image, whereas we are attempting to convert an image into a general tileset representing a shared representation for all levels of a game. Closest to our work, Stephenson et al. (Stephenson et al. 2019) developed a system to convert sketched images of structures made up of rectangular blocks into analogous structures in Angry Birds. While these structures are important portions of Angry Birds levels, they do not include enemies or the terrain found in complete levels. Image-to-Level produces all of these tile types and thus creates full game levels.

There is more research on how computer vision can impact textures, which are used for many 3D games. Champandard (Champandard 2016) demonstrated how low detail segmented image layouts can be used to generate new images that match the artistic style extracted from another trained image, a process also known as Image Style Transfer (Gatys, Ecker, and Bethge 2016). Image Inpainting refers to the programmatic repair of images, used to fill in image gaps (Guillemot and Le Meur 2013). This is similar to our methods that take advantage of known statistical properties of existing game levels to repair existing images. The difference is that our levels do not have missing gaps, but rather have areas that do not satisfy the local statistical tile properties present in existing game levels. Isola et al. (Isola et al. 2017) developed the pix2pix system which uses a GAN to perform image-to-image translation. This differs from our work as the output of Image-to-Level are tile-based level representations, rather than images. To apply pix2pix to generate game content, a method for converting images to a game level representation would still be required.

System Overview

Image-To-Level has a two step process in which we generate an initial level from a source image (Generation) and then repair that level (Repair) to produce a final level as demonstrated in Figure **??**. The intuition is that an image converted directly into a level would likely not include the mechanical and aesthetic constraints of the original game. Thus, we solve this problem with the addition of a Repair step. The Repair step alters the generated level structure to better fit the local patterns from the original game levels.

For each step we implemented two approaches. In the Generation step we have an approach we call Tile Translation and a Convolutional Neural Network (CNN) to convert an image into a game level in a tile representation. By tile we indicate one of a limited set of possible game components that could occur at each position of a level (e.g. a ground, an enemy, etc.). A Markov chain and an autoencoder are the two approaches we implemented for the Repair step.

We chose to employ multiple approaches as we believe that the primary value of this system is in the two steps of Generation and Repair. By implementing two approaches for each step, we can assess the value of these steps more generally than if we only implemented a single, potentially faulty approach for each. In addition, by building the system to allow for multiple approaches for either step the system becomes extendable to allow for other approaches to be implemented in the future. We expect different approaches to work better for different inputs and for different target games.

Our system learns from Super Mario Bros. and Lode Runner game levels and can generate repaired levels for both games from an input image. We use Super Mario Bros. as a running example in this paper as it is well understood in PCGML (Summerville et al. 2017) and hugely popular (Roach 2020). We avoid further discussion of Lode Runner due to lack of space. We use data from the Video Game Level Corpus (VGLC) (Summerville et al. 2016). The level data from the VGLC represents a simplification of the original Super Mario Bros. levels. For example, only one type of mobile enemy, the goomba, is represented within each level.

Generation

The initial step in our system is the generation of a level in a tile representation from an image input. This represents a major difference between our approach and most PCG methods, which use some random noise as input. We chose to focus on converting images into levels that look as close as possible to the source image; this allows a designer to use the source image as the starting point of level creation. We implemented two major approaches for this step: Tile Translation and a CNN.

Tile Translation Our desired output for the Generation task is a level in a tile representation. Our first approach is what we call Tile Translation (TT). At a high level, the TT approach slices a source image into multiple square cells and matches each with the most similar tile from the target game. This process is shown in Figure **??**. The input image, in this case the Shot Marilyns, is converted into square cells as can be seen in the bottom left. For each cell we extract a normalized histogram H of red, green, and blue colors (RGB). Each cell is matched to the closest tile from the target game according to some distance function. We visualize this process in the bottom right, with the output after each cell has been matched at the top right. Notably, we use two different distance functions for matching image cells to their closest game tiles. We employed two distance functions as we anticipated that different types of images and games might be better suited to different distance functions.

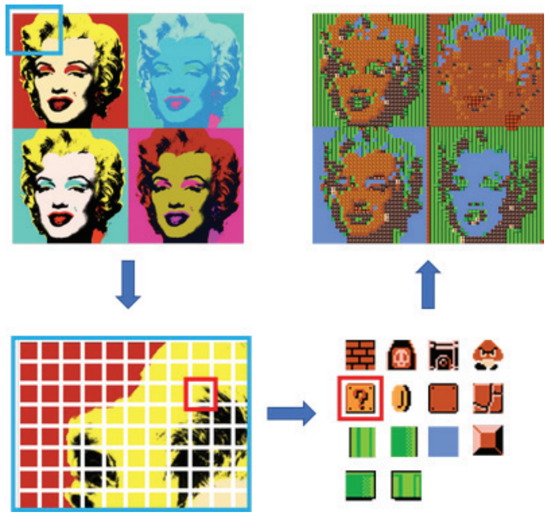
Figure 2: Tile Translation

We chose to compare image cells with game sprites using color because the approach is straightforward and intuitive. A deep red cell should be matched with the closest deep red tile from the game.

Our first distance function for comparing color between an image cell and game tile is the Bhattacharyya distance formula (Guorong, Peiqi, and Minhui 1996). The Bhattacharyya distance is meant to measure the overlap between probability distributions, which are similar in representation to the color histograms described above. Given that we directly compare the histograms, we refer to this distance function as Histogram. We theorized that an even simpler distance function could result in better tile matching. Thus for our second distance function we directly compare the average color for each cell and tile in Red, Green, Blue (RGB) space. We refer to this as the Average distance function.

Convolutional Neural Network The Convolutional Neural Network (CNN) takes an ML approach to converting images into game levels. Our TT approach is dependent on having a separate set of tile images for comparison, but the CNN can be trained using only images of a game level and the game in the tile representation. Theoretically, it can also

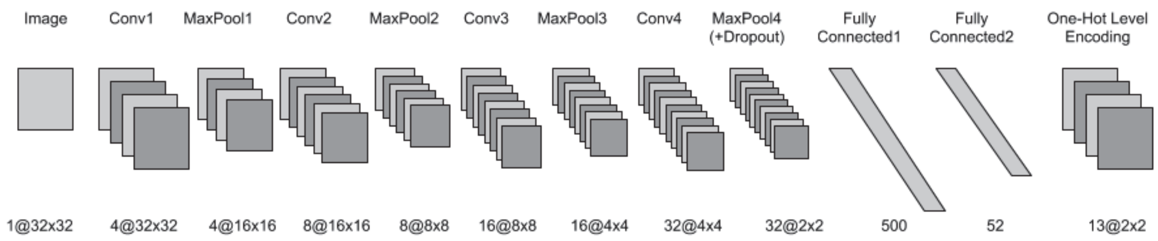

Figure 3: Network architecture for the CNN.

handle decorative elements more robustly, avoiding converting them into tiles when desired. We note that the VGLC representation that we use ignores decorative elements.

A CNN is typically trained on square images, while Super Mario Bros. levels are rectangular in shape. To address this, we slice each Super Mario Bros. level into a set of smaller images, each with a width equivalent to two tiles (16 pixels/tile, thus 32x32). These images constitute our inputs to train our CNN. For the training output, we use the equivalent 2x2 matrix of tiles. This lead to a training set of size 30,874. We experimented with 3x3, 4x4, all the way up to 8x8 but found that 2x2 lead to the best performing model. We use a one-hot encoding for the representation of each tile in a generated chunk. Our Super Mario Bros. data includes 13 different tile types (e.g. sky, ground, enemy) resulting in a one-hot encoding of size 13 and 2x2x13 as the output shape. We train our CNN using the architectures shown in Figure **??**, which is composed of four convolutional layers, with the parameters given above, followed by a dropout, fully-connected, and reshape layer, all using ReLU activation We arrived at this architecture through iterative experimentation on our training set. We trained this CNN for 50 epochs with a batch size of 16, using the adam optimizer and the mean square error loss function. Once trained, we iterate through an input image in 32x32 pixel slices until the CNN has fully converted the entire source image into a game level.

Repair

An image converted directly into a level in a tile representation may not accurately reflect the design of the existing levels. This is especially a concern when the input image is more abstract or is representative of something different from a game level (e.g. the Shot Marilyns in Figure **??**). Therefore, we are using this additional step to rearrange the tiles in a way to accommodate for different title combinations to make the output from the Generate step closely resemble the original game. For this step we drew on two approaches, an autoencoder and a Markov chain.

Autoencoder An autoencoder is a deep neural network model that learns to map input to itself, essentially learning a space of valid inputs, which has shown success in repairing level content (Jain et al. 2016). To create an autoencoder for repairing video game levels, we first need to gather training data. To tackle this problem, we split our levels in the tile-representation into 8x8 windows. We chose this size because it represents the smallest chunk of a level that still captures local structure, which we determined empirically. We again used the one-hot encoding used for the CNN Generate approach, meaning our input and output had a 8x8x13 shape. This lead to a training set of size 19,726 from the Super Mario Bros. levels from the VGLC.

Our autoencoder architecture is informed by the work of (Guzdial et al. 2018). Figure **??** depicts our architecture. We use a convolutional autoencoder which uses the 8x8x13 tensors. The encoder is made up of two convolutional layers with Tanh activations. The output of the second layer then passes through a fully connected embedding layer. This embedding then passes through the decoder, made up of two deconvolutional layers mirroring those of the encoder. We observed the best results when using a ReLU activation on the output layer. The loss function used was the mean squared error and the model was optimized with adam. We trained the model for 40 epochs with a batch size of 64.

At inference time, a level passed to the autoencoder must first be split into 8x8 chunks. These chunks are passed through our autoencoder, which then maps the chunk to the closest legal chunk in its learned latent space. Chunks repaired by the autoencoder are concatenated in the same fashion as they were created using a sliding window method. The result is a one-hot encoding of a full level, which can then be converted to a level in our tile-based representation by replacing each vector with its corresponding tile. Training the autoencoder requires fixed input dimensions, but levels produced by the generation step may not have consistent height and width. In this way, the autoencoder can repair levels regardless of the level dimensions.

Markov Chain Even by converting the images into 8x8 slices, the Autoencoder still has a relatively small training set. One method that does not require as large of a training set is a Markov Chain (MC) (Snodgrass and Ontanón 2016). A MC learns a mapping between a tile and its surroundings, what is the probability of a tile being placed in a position given the surrounding tiles. To train the MC, we used the Super Mario Bros. levels. For each tile (center), we can consider it to be surrounded by eight other tiles as seen in Figure **??**. A MC requires keys and values, for our purposes we used the center as the value and a key made from the tile values of the North and West tiles in that order. We wanted to use as few tiles as possible to gain the most information and found these two tiles gave the best results. The model builds a representation of local structure in the form of a probability distribution by associating the key with the

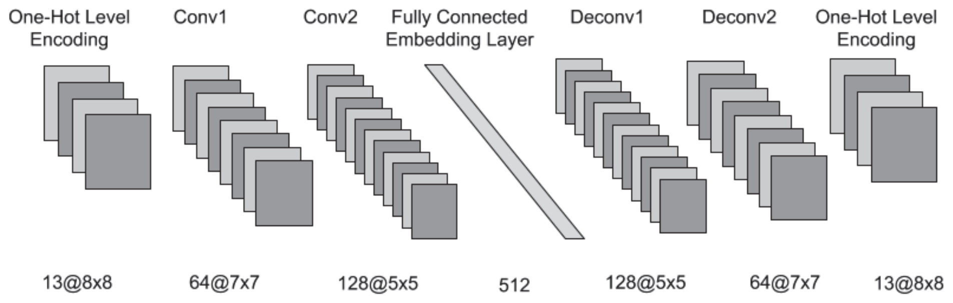

Figure 4: Network architecture for the autoencoder.

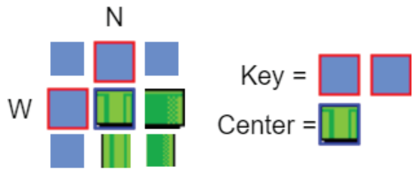

Figure 5: Tiles used in building MC model

center tile (label) in the existing game levels. For example, if half the time when north and west were both the blue sky tile as in Figure **??** we saw the top left of a pipe, then the top left of a pipe would have a probability of 50% for this key. This is a smaller and therefore less descriptive key than other Markov PCGML approaches (Snodgrass and Ontañón 2016), but we still found success with it since this application is focused on repairing content instead of generating it from scratch.

Given a generated level, the MC iterates over each tile and gathers the surrounding north and west tiles to form the key. It then checks the model to see if the key exists or not. If the key exists then it queries to see if there is any probability for the center tile to occur with this key, if so then nothing is changed. If the key exists but there are no instances of the center tile in the model then we sample a valid center tile from the model randomly to ensure variability in the output.

Evaluation

The primary metric for success for our Image-to-Level system is its ability to translate from images to game levels. This can be subjective and we plan to do a human subject study to follow this work. In the meantime, we want to quantitatively assess the translation of an image into a level using ground truth, and we chose to focus on two sets of images with known tile-based level representations. The first set, known as the *Training* set, was composed of sections of each of the original Super Mario Bros. levels used for training our CNN, autoencoder, and Markov chain. The second set, the *Test* set, was comprised of Super Mario Bros.: the Lost World levels from the VGLC, which were not used in the training of the system. As not all levels in Super Mario Bros.: The Lost Levels are the same height, we selected only those that contained exactly 14 tile rows for consistency. This resulted in the use of 15 levels in the Training set and 8 in the Test set. Since all levels are not the same width we cropped all levels from the 50th column up to the 90th column of tiles in the level. For the evaluation, we calculated the percentage of tiles that were correctly converted back to the true level tile after the image passed through Image-to-Level. This was done for each of the possible Generation and Repair approaches. In this way we can evaluate all variations of the system on its primary task where we have a gold standard answer.

Results

Table 1 shows the results from running the two level sets though Image-to-Level and calculating the percentage of correctly translated tiles, averaged over all levels. The "No Repair" column indicates the average tile accuracy prior to the Repair step. The results indicate that the Tile Translation (TT) approach was able to perfectly reproduce Super Mario Bros. and Super Mario Bros.: The Lost Levels levels from their level images. This follows from the fact that two games are nearly identical in terms of their tiles, only differing in decorative elements (which are ignored by the VGLC) and the ground tile. The results also indicate that the relatively simple "Average" distance function performed as well as the much more complex "Histogram" distance function. However, given that this tile translation task was straightforward, we don't take it as conclusive evidence that one should always stick with Average.

The CNN performed worse than TT at the Generation step. However, it performed equivalently on both the training and testing tasks. This indicates its ability to perform well, with only 5% error in cases similar to the original Super Mario Bros. However, we anticipate the CNN may struggle with input that varies significantly from its training set.

Both the autoencoder and Markov Chain repair methods slightly degraded (around 1%) the accuracy from the Generation step, with the latter performing slightly better. This indicates that overall both approaches performed well as a repair functions, correctly identifying that almost no repair had to be done. The Markov chain created more changes to the levels on average, leading to a small improvement in the accuracy of the CNN levels both in the training and test cases. While small, this indicates that the Markov chain is capable of improving levels to better reflect Super Mario Bros. structure. The Markov Chain's slight degrading for the

Category	Generation Step	No Repair	Autoencoder	Markov Chain
Training	CNN	94.4524	94.1786	94.7143
	TT: Histogram	100	98.9881	99.7738
	TT: Average	100	98.9881	99.7857
Test	CNN	96.1384	96.0491	96.3839
	TT: Histogram	100	99.1964	99.3973
	TT: Average	100	99.1964	99.0402

Table 1: The average percentage of tiles matching the true tiles in the training set versus the test set. TT refers to the Tile Translation method, followed by the underlying distance function used for the generation step.

TT methods is due to the fact that we selected the second 40 tiles of each level to enforce uniformity. Because of this, the beginning of the level did not match any beginning of any level in the Markov chain's training data. Therefore, we surmise that the Markov chain is overall the stronger method when there are greater repairs to make.

These results are positive, but don't reflect the subjective performance of our system with images that are not already representative of Mario levels. This is where we expect our Repair step to improve the generated levels, and we address this here through a qualitative example. To avoid bias in our choice of input, we chose the first for-public-use result when searching for "landscape" on Google images. We chose the word "landscape" as it seemed likely to describe the kind of scenic photo we imagined a user might use as input to our system. We scaled the image to match the 40x14 game tile resolution used in our quantitative results. The result of passing the image through Image-to-Level can be seen in Figure 6. We selected one particular image which was generated using the TT method with the Average distance function and repaired it using the MC approach. After Generation, the output level is composed of almost all "Goomba" enemies, which would immediately fall off the screen if the level was loaded into a Mario engine. The repair method changed most of these Goomba tiles to block tiles, reduced the number of coins, fixed a cannon tile, and added ground to the base of the level. Overall, we believe that after repair the resulting level image better demonstrates Mario constrained while strongly resembling the original image.

Limitations and Future Work

Our results demonstrate that Image-to-Level can be successful at transforming images into levels. One significant benefit of Image-to-Level is the modularity of the Generation and Repair steps. We saw the utility of these variations in our quantitative results. We plan to explore other Generation and Repair methods such as (Shu et al. 2020) in future work.

We have not yet run a human subject study to determine the system's utility as a design tool. We have begun the process of extending our system to specifically handle turning sketches into levels, based on the inspiration of the historical design of Super Mario Bros. levels on graph paper. We plan to investigate this further with additional improvements to our user interface. Potential avenues include real-time translations of sketches into immediately playable levels. We anticipate, due to the simplicity of the system, that it could also be used for outreach or as part of a public installation.

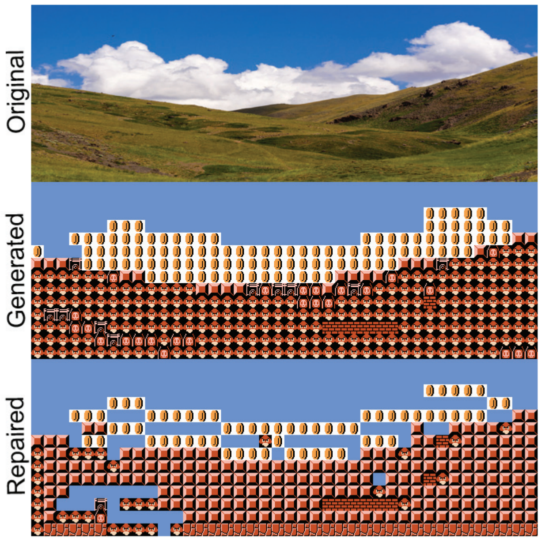

Figure 6: A Randomly selected image passed through Image-to-Level, using the Average generation and Markov chain repair methods. More examples are available at http://imagetolevel.com

The system currently provides no guarantee that we can output good levels. Undesired patterns can still exist after repair: for example, there are enemies trapped inside the ground in our repaired version of Figure 6. Further, there is no guarantee that either of our currently supported games (Super Mario Bros. and Lode Runner) will be good fits for an input image. Extending this towards other non-platforming games, such as Pacman, Brick Breaker, or Mario Kart would increase the expressivity of the system. Further, since these games have a resolution closer to a standard "portrait" resolution, it would allow users to create levels from images like selfies.

One regular issue with PCGML broadly is a lack of training data. If we can extend this approach to more games, we could allow PCGML researchers to translate images into tile-based levels that can then serve as training data. To further benefit the PCGML community, one immediate goal is to make our system open-source and extendable so that other researchers can add additional PCGML techniques.

Conclusions

We present a system that takes images as inputs and outputs game levels using PCGML techniques. Our Generation and Repair approach has yielded positive results, both quantitatively in recreating game levels from screenshots and qualitatively in producing a novel level from a photograph.

Acknowledgements

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Alberta Machine Intelligence Institute (AMII) in supporting this research.

References

Baldwin, A.; Dahlskog, S.; Font, J. M.; and Holmberg, J. 2017. Towards pattern-based mixed-initiative dungeon generation. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, FDG '17. New York, NY, USA: Association for Computing Machinery.

Champandard, A. J. 2016. Semantic style transfer and turning two-bit doodles into fine artworks. *arXiv preprint arXiv:1603.01768*.

Forsyth, D. A., and Ponce, J. 2002. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference.

Gatys, L. A.; Ecker, A. S.; and Bethge, M. 2016. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2414–2423.

Guillemot, C., and Le Meur, O. 2013. Image inpainting: Overview and recent advances. *IEEE signal processing magazine* 31(1):127–144.

Guorong, X.; Peiqi, C.; and Minhui, W. 1996. Bhattacharyya distance feature selection. In *Proceedings of 13th International Conference on Pattern Recognition*, volume 2, 195–199. IEEE.

Guzdial, M.; Long, D.; Cassion, C.; and Das, A. 2017. Visual procedural content generation with an artificial abstract artist. In *Proceedings of ICCC Computational Creativity and Games Workshop*.

Guzdial, M.; Reno, J.; Chen, J.; Smith, G.; and Riedl, M. 2018. Explainable PCGML via game design patterns. *CoRR* abs/1809.09419.

Guzdial, M.; Liao, N.; Chen, J.; Chen, S.-Y.; Shah, S.; Shah, V.; Reno, J.; Smith, G.; and Riedl, M. O. 2019. Friend, collaborator, student, manager. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*.

Isola, P.; Zhu, J.; Zhou, T.; and Efros, A. A. 2017. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5967–5976.

Jain, R.; Isaksen, A.; Holmgård, C.; and Togelius, J. 2016. Autoencoder for level generation, repair, and recognition. In *ICCC Workshop on Computational Creativity and Games*.

Lai, G.; Latham, W.; and Leymarie, F. F. 2020. Towards friendly mixed initiative procedural content generation: Three pillars of industry. *arXiv* arXiv–2005.

Liapis, A.; Smith, G.; and Shaker, N. 2016. Mixed-initiative content creation. In *Procedural content generation in games*. Springer. 195–214.

Machado, T.; Gopstein, D.; Nov, O.; Wang, A.; Nealen, A.; and Togelius, J. 2019. Evaluation of a recommender system for assisting novice game designers.

Mott, J.; Nandi, S.; and Zeller, L. 2019. Controllable and coherent level generation: A two-pronged approach. In *6th Experimental AI in Games Workshop at AIIDE 2019*.

Murphy, M. 2015. The original super mario game was designed on graph paper.

Rebouças Serpa, Y., and Formico Rodrigues, M. A. 2019. Towards machine-learning assisted asset generation for games: A study on pixel art sprite sheets. In *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 182–191.

Roach, J. 2020. The bestselling games of all time.

Sarkar, A.; Yang, Z.; and Cooper, S. 2020. Controllable level blending between games using variational autoencoders. *arXiv preprint arXiv:2002.11869*.

Schreier, J. 2017. The story behind mass effect: Andromeda's troubled five-year development.

Schrum, J.; Gutierrez, J.; Volz, V.; Liu, J.; Lucas, S.; and Risi, S. 2020. Interactive evolution and exploration within latent level-design space of generative adversarial networks. *arXiv preprint arXiv:2004.00151*.

Shu, T.; Wang, Z.; Liu, J.; and Yao, X. 2020. A novel cnet-assisted evolutionary level repairer and its applications to super mario bros. *ArXiv* abs/2005.06148.

Smith, G.; Whitehead, J.; and Mateas, M. 2010. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, 209–216.

Snodgrass, S., and Ontanón, S. 2016. Learning to generate video game maps using markov models. *IEEE transactions on computational intelligence and AI in games* 9(4):410–422.

Snodgrass, S. 2018. Towards automatic extraction of tile types from level images. In *AIIDE Workshops*.

Stephenson, M. J. B.; Renz, J.; Ge, X.; and Zhang, P. 2019. Generating stable, building block structures from sketches. *IEEE Transactions on Games* 1–1.

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

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2017. Procedural content generation via machine learning (PCGML). *CoRR* abs/1702.00539.