Generating Pixel Art Character Sprites using GANs

Flávio Coutinho*† and Luiz Chaimowicz*
*Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
{flavioro, chaimo}@dcc.ufmg.br
†Departamento de Computação, Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, Brazil
fegemo@cefetmg.br

2022 21st Brazilian Symposium on Computer Games and Digital Entertainment (SBGames) | 978-1-6654-6156-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/SBGAMES56371.2022.9961120

***Abstract*—Iterating on creating pixel art character sprite sheets is essential to the game development process. However, it can take a lot of effort until the final versions containing different poses and animation clips are achieved. This paper reports an investigation on the use of conditional generative adversarial networks to aid the designers in creating such sprite sheets. We propose an architecture based on Pix2Pix to generate images of characters facing a target side (e.g., right) given sprites of them in a source pose (e.g., front). Experiments with small pixel art datasets yielded promising results, producing models with varying degrees of generalization, sometimes capable of generating images very close to the ground truth. We analyzed the results through visual inspection and quantitatively with FID.**

***Index Terms*—generative adversarial networks, pixel art, image-to-image translation, procedural content generation**

I. Introduction

Procedural content generation (PCG) can be described as the *"algorithmic creation of game content with limited or indirect user input"* (p. 6) [1]. It can be used either online in games or offline during development. In the former case, it is usually employed to generate game levels to enable richer replay value, as one playthrough can be very different from the other. In the latter case, the users are the game developers: the level designers receiving aid to generate content or designers when creating art assets.

In the case of online PCG, a common criticism is that such systems usually create repetitive content, making it less memorable to players [2]. To some extent, PCG tries to replace human designers in their activity by developing programs that create content, and programmers do not necessarily have good game and level design skills. However, when offline methods are embedded in game design tools, for example, PCG can *"augment the creativity of individual human creators"* (p. 3) [3] instead of trying to replace them. As an illustration, some PCG systems let human creators and algorithms work together to produce game content. The automation of only part of the content creation pipeline is usually designated as mixed-initiative PCG, when both the algorithm and the human creator can interact with the content being generated in a co-creation process [4], [5].

In this work, we present an architecture for generating pixel art character sprites in a target pose given its image in a source direction: for example, our approach can create an image of a character facing to the right from another one of it facing front. Such a model can aid designers during their creation process when integrated with a PCG tool.

The proposed architecture is based on Pix2Pix [6]. We trained and experimented with models using different datasets. The generated images were evaluated through visual inspection and by calculating the Fréchet Inception Distance [7] between the model's output and the ground truth for a quantitative assessment.

The four datasets were small and had a number of training examples varying from 184 images to 776 in the largest one. As a result, in a dataset in which characters are modularly built by assembling different parts of the body, the model generated images with high perceptual quality, with almost no divergence from the ground truth. In the other datasets, the results had different quality levels, with some sprites suffering from high-frequency color noise. On average, the characters' shapes were more similar to the target images than their colors. Overall, the generated shape and the colors still resembled the ground truth, even for some lesser quality results.

II. Related Work

Serpa and Rodrigues [8] tackled the generation of two types of pixel art sprites of game characters from line art sketches: (a) a grayscale sprite encoding shading information and (b) a colored sprite with body-part segmentation. The grayscale sprite can have 6 shades of gray and encodes how the image responds to lighting. On the other hand, the colored sprite segments the character in different body parts (head, face, hair, limbs, trunk, etc.) by assigning one of 42 colors to each region.

The authors adapted the Pix2Pix architecture [6] to generate two sprites from the same input. It comprises a U-net generator and a patch-based discriminator. The modifications include: 1. the U-net generator has two decoders instead of a single, each one responsible for producing each sprite (gray and colored); 2. the use of an L_2 loss term in the generator's loss function instead of the L_1 used in original Pix2Pix; and 3. replacement of the leaky ReLU activations by ELU functions in the layers used for downsampling in the generator.

In their experiments, the authors trained the model separately for the sprites of two characters (85 and 530 training examples). The generated grayscale images were very close to the ground truth in both characters. For the color sprites (segmentation of body parts), the model trained for the character with larger region segments (it had large arms and torso)

This work was partially supported by CAPES, CNPq and Fapemig.

generated better results for input sprites with similar poses in the training set. However, for unseen poses, the generated images contained a lot of high-frequency noise.

Jiang and Sweetser [9] proposed a generative model for automatic coloring of game sprites, turning single-channel images into colored ones. It is also based on the Pix2Pix architecture [6] and processes images using the YUV representation rather than in the RGB color space.

Experiments showed that the model trained with YUV inputs yielded better-colored images than when represented with RGB. Such improvement is attributed to the fact that when working in YUV, the network had to learn only the U and V channels, using Y as a redundancy of the condition.

Gonzalez et al. [10] approached the problem of translating (Pokémon) pixel art characters from a source domain (Pokémon type, e.g., fire) into a version of the same sprite but in a target domain (e.g., grass). They used a Variational Autoencoder (VAE) with convolutional layers.

With a small dataset composed of 974 images, the authors decided to experiment with pre-training the model with the Anime Face Dataset. The reconstruction of the original images yielded blurry results with shapes faintly resembling the original but with noisy colors.

The domain transfer (type swap) task had more shape degradation, usually with high-frequency noise and dangling pixels outside the intended character shape. Color-wise, using transfer learning provided images with colors that resembled the ones from the target domain (i.e., Pokémon type).

Hong et al. [11] proposed a system that takes two sprites, one representing a character's shape and color and the other comprising a target pose on which to draw the first. The authors used a multiple discriminator GAN (MDGAN).

The generator takes as input an image with the target pose information (bone graph) and another with shape and color, and it uses separate encoders for each. Its goal is to generate a sprite of the character with the shape and color of the second input image but in the pose represented in the first. There are two discriminators, the first being responsible for determining whether two images share the same color and shape, and the second extracting the pose of a character sprite.

The authors compared their approach with an adapted version of Pix2Pix, among other architectures, and MDGAN had the best outcome. However, albeit successful in the proposed experiments, the system requires a bone graph dataset matching characters' positions in the shape and color dataset. For that reason, the authors artificially tailored datasets with non-pixel art images, and using such an approach would make it difficult with real pixel art character sprites. In addition, all characters are required to have the exact same shape, varying only its texture. And that greatly restricts the artistic variability.

Our work also adapts Pix2Pix [9], [12] to generate pixel art character sprites. But differently from the related work, we want to translate characters drawn on a source to a target side. Regarding the particular task being tackled, MDGAN [11] is the most similar work. Still, it imposes unnecessary restrictions to the problem. Regarding the image representation, we consider RGBA channels versus RGB in [8], [11], YUV in [9], and HSV in [10]. We also study the impact of the alpha channel in one of our experiments. Our main contribution is the proposition of an architecture that can translate character sprite images from one side into another with differing levels of generalization capacity, depending on the dataset size and variance, exempting from an artificial bone-graph dataset and not restricting character shapes.

III. Architecture Overview

Similar to some related works [8], [9], our architecture is based on Pix2Pix. Each model we train can translate images of a dataset from a source into a target pose. Next, we describe our generator and discriminator networks. As presented in Fig. 1, the input to the generator is a 64×64×4 image with RGBA channels in the [-1, 1] domain.

A. Generator

The generator is a U-net with 64×64×4 input and output. The layers in the first half downscale the image to 1×1, and those in the second upscale the data to its original resolution.

Each downsampling step is composed of a convolution (4×4 kernels), instance normalization (except for the first one), and leaky ReLU layers, and it halves the resolution. Considering the input has 64×64 pixels, there are 6 downsampling blocks.

The decoder is a reflection analogy of the encoder, with upscaling steps instead. Each block comprises a transpose convolution (4×4 kernels), instance normalization, an optional dropout (50%), and ReLU activation layers. The first three blocks have dropout regularization, and the number of filters is the same as in the encoder but in reverse order. The last upsampling block has a $\tanh$ activation, so it outputs pixel intensities in the range of [-1, 1]. To allow the network to learn custom downsampling/upsampling mechanisms, it changes the resolution only through fractionally/strided convolutions [13]. There are skip connections from the output of the i^{th} encoder layer to the $n^{th} - i$ decoder one, with $n = 6$, to preserve spatial information.

The loss function for the generator is similar to the one in [6], but uses a non-saturating adversarial part. We also use the L_1 distance between the real and generated images, with the hyperparameter λ set to 100. The generator's loss is:

$$\mathcal{L}_G = -\mathbb{E}_x[\log D(G(x))] + \lambda \mathbb{E}_{x,y}[||y - G(x)||_1] \quad (1)$$

where x represents images in source pose, and y in the target; and $||\cdot||_1$ is the L_1 distance (absolute value of the differences of the pixels from generated to target images).

B. Discriminator

Our discriminator is a PatchGAN [6]: a conditional classifier that takes a source image (e.g., character facing front) as a condition, and either a real or a generated sprite in the target direction (e.g., facing right), splits it into same-size square patches, and discriminates each region as being real or fake.

The rationale behind splitting the output in patches is to enable local discrimination instead of providing a single global

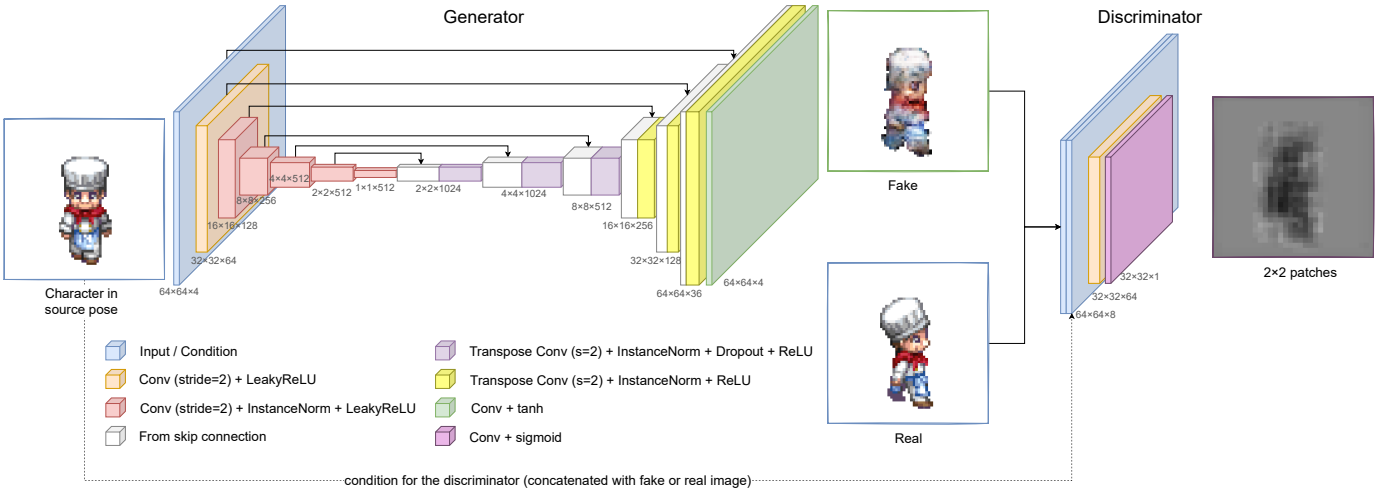

Fig. 1. Generator (left) and discriminator (right) network architecture.

value of a sprite being either real or fake. Doing so allows penalizing more parts of the images that require more work from the generator. While the generator's L_1 loss term steers the generation towards the target images (but leads to blurry results), the patch discrimination works as a texture loss.

We experimented with different patch sizes: 2×2, 5×5, 11×11, and 64×64 (single patch) and Fig. 2 shows a comparison. The patch size of 2×2 achieved the closest result to the ground truth. The other images presented some color and shape noise, with the models using larger patches yielding worse results. In particular, the model with a single patch suffered from dangling pixels outside the sprite shape.

We attribute the better results with smaller patches to the discriminator being penalized by misclassifying 2×2 regions individually. As pixel art sprites typically have low resolution, each pixel carries a lot of information and should consider mostly its local vicinity. In such a setting, there are very small regions of low frequency (same color or just a slight variation). Hence, smaller patches evaluate not only the texture of the area but also the shape edges. Furthermore, it may be due to that double responsibility that the images present high color variation even in parts that should have a single or a few colors. For such reasons, we chose to use 2×2 patches. Ultimately, the resulting shape and colors in all patch sizes resemble the ground truth to some degree.

Regarding its layers, the discriminator is composed of the same downsampling blocks used by the generator (Fig. 1). Moreover, its loss function is the same as for conditional GANs [14], which can be calculated as a binary cross-entropy between the real images it discriminated as fake and the fake discriminated as real. The only change is that because the discriminator's output is not a single number per image but one for each patch, it is first reduced to the mean value.

Fig. 2. Outputs of models with varying patch sizes for the discriminator.

IV. Experiments

We conducted different experiments to evaluate the model architecture. First, we investigated how well the model performed on the test data on the individual datasets. Then, the results were analyzed through visual inspection and quantitatively using the Fréchet Inception Distance (FID) [7].

During the analysis, we also investigated the generalization capacity of the models under different situations: (a) changing the colors of either one part or (b) the majority of a sprite, and (c) slightly changing the character pose.

Last, we evaluated whether the model provided better quality results when considering the alpha channel of the images or using only the RGB components.

Fig. 3. Sample images from the dataset showing 4 different sizes/art styles (columns) in 4 directions (rows).

A. Datasets

We assembled datasets of pixel art character sprites from different sources. They contemplate characters in four directions – front, right, back, and left – and comprises four different sizes and art styles (Fig. 3).

The four image sources had different character sizes, so the smaller ones were transparency-padded to the largest size, which was 64×64. Also, we created an alpha channel with the character shape for the images that lacked one. Table I presents the sizes and number of images per character pose. Some sources had 3 frames for each character side, as they were extracted from sprite sheets with walking animations.

TABLE I
DESCRIPTION OF THE DATASETS.

SOURCE	SIZE	EXAMPLES	PER POSE
TINY HERO	64×64×4	912	1
RPG MAKER 2000	32×24×3	216	3
RPG MAKER XP	48×32×4	294	3
RPG MAKER VX ACE	32×32×4	408	3

The Tiny Hero sprites were modularly created by assembling previously-drawn body pieces to form each character. For this reason, it is usual for different characters to present the same or similar shapes, but with a different color or a different combination of parts.

B. Training Procedure

Training used an 85% split. The model received images in batches of 1 sample. The generator's and discriminator's weights were optimized with Adam (momentum of $\beta_1 = 0.5$ and $\beta_2 = 0.999$) using a fixed learning rate of 0.0002. Instead of running the optimization for some epochs, as the batch contained a single image, the process was split into training steps. All of the models trained for 40,000 steps on a hardware with an NVidia GTX 1050 Ti GPU and it took about 01h30m

V. RESULTS

We now present the generated images in different settings, analyze the results through visual inspection, and then quantitatively compare them using FID[1].

A. Individual Datasets

As each dataset had a different number of training examples and the training steps were fixed, each model trained for a different number of epochs (see Table II).

In all experiments, the models were trained to translate a front-facing character into the pose facing right. Fig. 4 and 6 show results of the models trained with the individual datasets. All examples were hand-picked from the test data and organized vertically with better results at the top.

All training images could be reproduced with high fidelity by the models with all datasets, which is also indicated by their low FID values. Next, we analyze the models considering only images from the test set.

TABLE II
TRAINING SIZE, EPOCHS AND FID PER DATASET (40K STEPS).

DATASET	TRAIN SIZE	EPOCHS	FID	
			TRAIN	TEST
TINY HERO	776	≈52	0.092	0.115
RPG MAKER 2000	184	≈217	0.091	2.306
RPG MAKER XP	250	160	0.264	9.493
RPG MAKER VX ACE	347	≈115	0.263	5.495

1) Tiny Hero sprites: The images generated for Tiny Hero (Fig. 4, left) were perceptually identical to the ground truth, which is corroborated by the FID score of 0.115. Although it looks rather impressive at first, such a nice result could only be achieved due to how the Tiny Hero dataset was built: the characters were created by assembling body parts and accessories. So although the model had not seen the test images nor their ground truth, it learned how to translate each body part while training. This test does not show that the proposed model can generalize the translation but instead that it can memorize segments of the sprites it sees during training.

2) RPG Maker 2000: The first row of Fig. 4 (right) shows a child girl translated with only minor color differences to the ground truth. Although that sprite was not on the training set, there was another one with the exact shape but different hair and dress colors. In this case, the model could understand their shapes and translate only the colors of parts of a sprite. Something similar happened to the granny and gramps in the

[1]Note on FID: the distances should be calculated with thousands of images [7]. However, the metric still applies even with our tiny datasets.

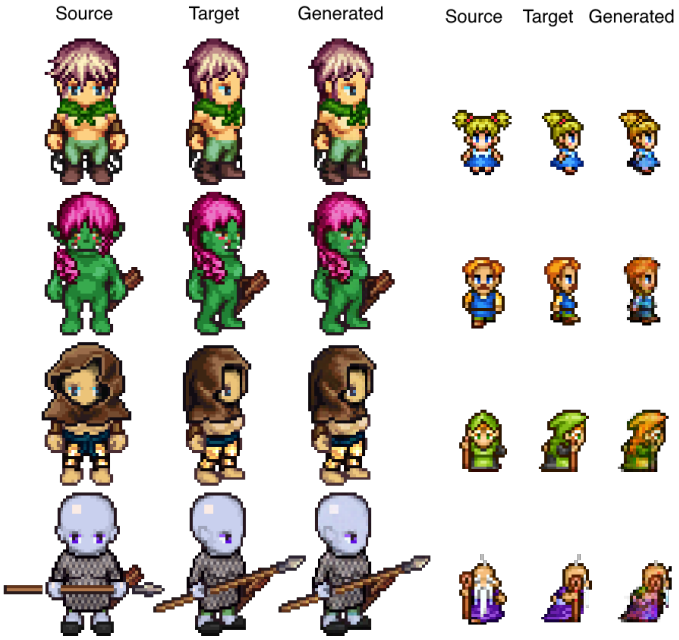

Fig. 4. Test examples from *Tiny Hero* (left) and *RPG Maker 2000* (right).

third and fourth rows – the model learned a sprite with the same shape but wearing an orange dress. In this case, it could not fully translate the colors, but it preserved the shape for the old lady and partially preserved it for the graybeard man.

The man in the second row of Fig. 4 was an interesting result suggesting a more profound generalization, as its translation uses information of a sprite with similar, but not equal shape, seen during training, but using different colors and slightly different shape. Fig. 5 shows the test image input, ground truth, and generated image, with the closest match from the training set to the side for comparison. We can note that the front-facing characters look similar, with differences in colors, the size and position of the hair, and the design of the mustache. The generated image contains some noise, and the edges inside the shape are not crisp, but it did not reuse the same hair and mustache shape as the other character.

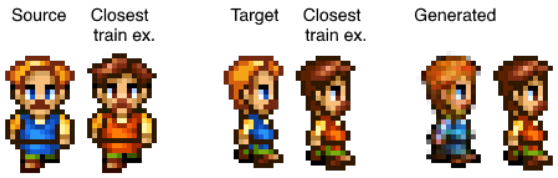

Fig. 5. Front and right view of a test sprite (blue clothes) with the closest example (orange) in the training set from *RPG Maker 2000*.

The FID value for the generated images in the dataset was 2.306 (Table II) and, after Tiny Hero, it was the smallest value, indicating the generated images were close to the ground truth.

3) RPG Maker XP: The model trained with the RPG Maker XP sprites had the highest FID score (9.493), indicating lower translation quality. Fig. 6 (left) shows examples of the best quality translations (top) to the worst ones (bottom). Such sprites are larger than those from RPG Maker 2000 (32×24 vs. 48×32) and not modular as in Tiny Hero. Furthermore, there are no sprites with the same shape with just some color variation, making it a suitable lower bound for the translation quality of the proposed model.

The image generated in the first row of Fig. 6 (left) had higher quality than the others. That happened because this dataset has 3 images per character pose, which are frames of a walking cycle. In this case, the frame of the character standing still stayed in the training set, with the other two as part of the test. Although the generated image contains some noise, it is possible to see the faint color of the ribbon in his head, and the overall shape is correct. This suggests that small changes in the pose are at least partially generalizable.

Each character in the middle rows (dealer girl and farmboy) has distinct features – a bunny tiara and a straw hat. Unfortunately, the model could not correctly translate such features' shapes, but we can notice that some traits, such as the most prominent colors and prominences, exist.

Last, the generated image for the clown in the fourth row has a lot of high-frequency noise. Again, the colors appear in meaningful positions, but there is very little information to separate them visually.

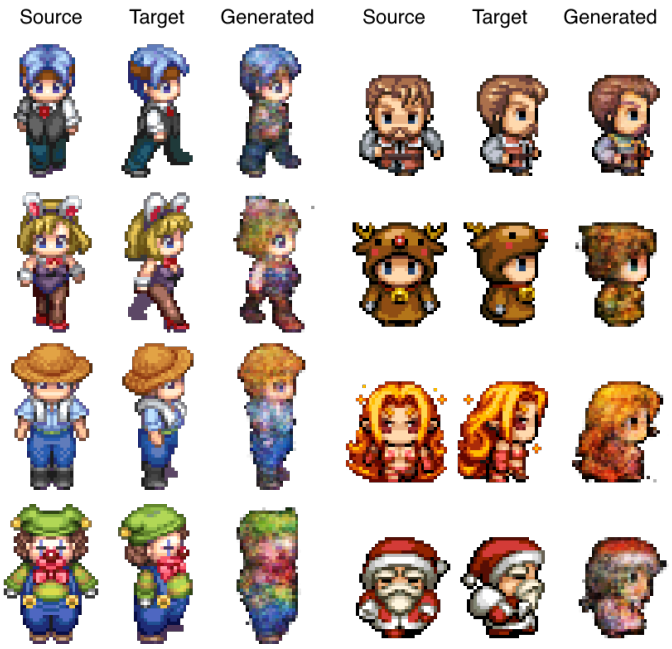

Fig. 6. Test examples from *RPG Maker XP* (left) and *RPGM VX Ace* (right).

4) RPG Maker VX Ace: The first row of Fig. 6 (right) presents the best result, and the character is a color variation of another sprite seen during training. Except for the color of the left hand and highlights in his hair, there is very little perceptual difference with the ground truth.

The characters in the middle rows have distinctive features, such as in the RPG Maker XP test. They differ from the ground truth, but the sprites have the overall colors and an approximate

shape. There was high-frequency color noise and blurry edges inside the shape. In the fourth row, Santa Claus was generated implausibly and, even worse, without its sack of gifts.

The comparison of the FID between the model output and the ground truth images was 5.495 – a value between RPG Maker 2000 and RPG Maker XP.

B. Alpha Channel

We experimented with training the model to receive, process, and output images with the RGBA components and with only RGB. In character sprites, the images typically have either an alpha channel or a background key color which is ignored by the game engine when rendering.

Although the alpha channel is mainly used as a boolean mask (so is the case in all the datasets used in this work), and a color key would also fulfill the same purpose, we found that our GAN models do use the redundant information in the channel, allowing it to generate shapes with higher fidelity to the ground truth and avoiding dangling pixels outside the expected shape region. Fig. 7 shows a comparison of images generated by models using RGBA and with RGB only.

Fig. 7. Comparison of model outputs when training with RGBA vs. RGB.

VI. Final Remarks

This work proposed an architecture based on Pix2Pix to create models that can translate images of character sprites from a source to a target pose (e.g., front to right-facing). We studied which patch sizes for the discriminator yielded results with better quality and also found that using the information in the alpha channel improves the quality of the shapes.

We showed through experiments with different datasets that the model can translate the color of partial parts of the characters and its entirety – although in some situations, the color transfer was not entirely correct. The model also indicated some generalization capacity to leverage the information of a character in a pose (front-facing, standing still) to satisfactorily translate a slightly different pose (front-facing, one foot forward).

Still, in the absence of more examples, the translations, in some cases, had low quality. Overall, when the model does not generalize well, the generated images suffer from having high-frequency color noise and very faint inner edges, resulting in unusable images.

For future work, one can experiment using the multi-domain architecture of StarGAN [15], as it would allow a single model to learn the mappings among all available poses instead of having one model for each. Another further investigation is to increase the model's generalizability by experimenting with different loss functions, network architectures, and image representations.

References

[1] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, "What is procedural content generation? Mario on the borderline," in *ACM International Conference Proceeding Series*. New York, New York, USA: ACM Press, 2011, pp. 1–6. [Online]. Available: http://portal.acm.org/citation.cfm?doid=2000919.2000922

[2] D. Karavolos, A. Bouwer, and R. Bidarra, "Mixed-Initiative Design of Game Levels: Integrating Mission and Space into Level Generation," in *Proceedings of the 10th International Conference on the Foundations of Digital Games*, 2015. [Online]. Available: http://unity3d.com/

[3] J. Togelius, N. Shaker, and M. J. Nelson, "Introduction," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016, ch. 1.

[4] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: Reactive planning and constraint solving for mixed-initiative level design," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, 9 2011, pp. 201–215. [Online]. Available: http://ieeexplore.ieee.org/document/5887401/

[5] A. Liapis, G. Smith, and N. Shaker, "Mixed-initiative content creation," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016, ch. 11.

[6] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2017-Janua. IEEE, 7 2017, pp. 5967–5976. [Online]. Available: http://ieeexplore.ieee.org/document/8100115/

[7] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 6627–6638, 6 2017. [Online]. Available: https://arxiv.org/abs/1706.08500v6

[8] Y. R. Serpa and M. A. F. Rodrigues, "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)*, vol. 2019-Octob. Rio de Janeiro: IEEE, 10 2019, pp. 182–191. [Online]. Available: https://ieeexplore.ieee.org/document/8924853/

[9] Z. Jiang and P. Sweetser, "GAN-Assisted YUV Pixel Art Generation," in *Australasian Joint Conference on Artificial Intelligence*, 2021, pp. 1–12.

[10] A. Gonzalez, M. Guzdial, and F. Ramos, "Generating Gameplay-Relevant Art Assets with Transfer Learning," in *Proceedings of the AIIDE Workshop on Experimental AI in Games*, 10 2020, pp. 1–7. [Online]. Available: http://arxiv.org/abs/2010.01681

[11] S. Hong, S. Kim, and S. Kang, "Game sprite generator using a multi discriminator GAN," *KSII Transactions on Internet and Information Systems*, vol. 13, no. 8, pp. 4255–4269, 2019. [Online]. Available: http://itiis.org/digital-library/manuscript/2473

[12] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*. Springer, 2016. [Online]. Available: http://pcgbook.com/

[13] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 11 2015. [Online]. Available: http://arxiv.org/abs/1511.06434

[14] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *arXiv preprint arXiv:1411.1784*, 11 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[15] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "StarGAN: Unified Generative Adversarial Networks for Multi-domain Image-to-Image Translation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 6 2018, pp. 8789–8797. [Online]. Available: https://ieeexplore.ieee.org/document/8579014/