Sprite Generation

CS 534 - Fall 2018 -Term Project Proposal

Instructor: Mohit Gupta - University of Wisconsin Madison

Date: October 31, 2018

Authors:

- Curt Henrichs - cdhenrichs@wisc.edu - 9070390126
- Sayem Wani - swani@wisc.edu - 9080301972
- Saheen Feroz - sferoz@wisc.edu - 9080316251

Website: https://uwmadison-cs534-term-project-f2018-cdh.github.io/

Problem

Developing visual content for games, even simplified 2D games, proves to be a hurdle for indie studios and solo game developers who have neither the time nor capital to develop captivating environments, engaging character designs, and remarkable particle effects. So these developers turn to procedural algorithms to assist them in this endeavour. Current, general-use algorithms either iterate over the equivalent of building blocks to construct sprites that lack the detail of a hand-crafted image or explore a crafted algorithm using a pseudo-random seed [1]. Alternatively a developer can develop a deep neural network, such as a generative-adversarial network, to produce novel sprites for their game [2]. However, a neural network is not a panacea; it requires a large dataset of images. Additionally, the developer must rely on the network converging to a optimal weight space while simultaneously not overfitting.

Solution Proposed

Our solution takes significant inspiration from Lewis Horsley's and Diego Perez-Liebana's 2017 paper titled *Building an Automatic Sprite Generator with Deep Convolutional Generative Adversarial Networks* [2]. The rational to use a GAN over hand-crafting would be to generate sprites without much human intervention after model training. Note further refinement and animation would still need to be done to tailor these sprites for the specific game. As a future step the sprite pipeline may include animation generation [5]. Our motivation partially drawn from personal experience where the problem with making games comes down to doing the art and partially inspired from literature [1].

One valid critique of the GAN method is the sprite's art style. That is the sprite will be a combination of features from the dataset the network was trained on and thus may not have consistent artistic qualities. Our solution will handle this in two parts. First, the dataset will be curated with a rough theme in mind, something analogous to a game studio's genre. Second, themes will be taught into an autoencoder as it can be thought of as a pre-trained style domain that learned the visual metaphor for sprite generation in its decoder.

Model Architecture

Our solution will first train an autoencoder, then flip the encoder, decoder using the hidden central feature structure as both input and output, see Figure 1. After this we will hold the autoencoder's weights constant while training a generator and discriminator. Meaning the GAN only needs to learn what features to select and discriminate on instead of having to learn the entire space. Note a possible limitation of our solution is a potential for loss in novelty with the worst case being mode collapse [6].

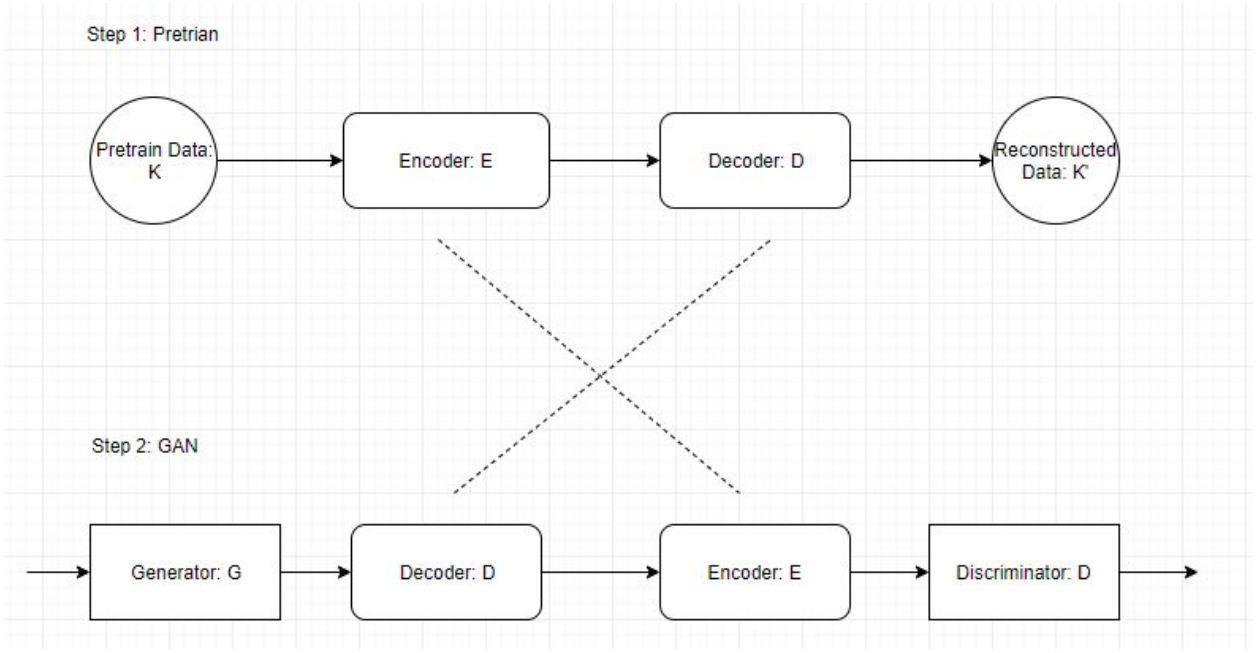

Figure 1: High-level network design with autoencoder.

Drawing on Lewis Horsley's and Diego Perez-Liebana's paper [2], we will use a similar DCGAN [4] model including their use of strided convolution in lieu of maxpooling and fractional strided convolution instead of upsampling and we will implement batch normalization. Additionally, we are making the following changes,

1. Our network will generate images of 32x32 pixel images.
2. We will adjust the hyperparameters such as learning rate and network structure to get best possible results from our dataset.

3. We will perform pretraining of the generator model through autoencoders. Our network will use successive decoders from result back trained through the normal unsupervised process. This should populate the weight with a better chance of generalizing from input. A potential drawback will be lack of novelty in output images if the autoencoder overfits during training.
4. After obtaining results for autoencoder we will develop a conditional GAN. This means that there are labeled inputs into generator and discriminator to provide more control than just a random input vector [3]. Providing this supplemental data provides the network with input structure which should lead to better image generation. The challenge will be in constructing a dataset with granular labels.

Comparison Architecture

For comparison we will develop a standard DCGAN that closely follow Horsley's and Perez-Liebana's architecture. without the autoencoder and labeling.This network will still have hyperparameters adjusted to produce best possible output given the limitation on time.

Datasets

Regarding the dataset, we will be collecting sprites from open source resources, specifically OpenGameArt.org. This should provide enough data for an initial system. If necessary we can perform simple manipulations such as changing the color palette and single-pixel manipulations. Finally, if we need more data than what can be collected online and with time permitting, we will employ neural style transfer on our current sprite. Our team recognizes the largest risk to the success of the project is collecting enough quality data for our network to generalize. Our models will be trained on constrained topics as subsets of items, entities, and environment. While this will reduce our training set it will also reduce the variance thereby reducing cases to learn.

All sprite data will be filtered to only include sprites of 32x32 pixels, some sprites may be cropped or padded (note sprites that when manipulated, lose the original meaning such as terrain, will be discarded). Sprites themselves may either originate from a sprite-sheet where they are extracted or may already be provided individually. The sprite is then normalized to a four channel image (RGBA) before being manually classified in a hierarchy. In the future a classification neural network could be developed for sorting but that is outside the scope of our project. Finally, we package the sorted image data as serialized Numpy arrays along with labels generated from the hierarchical structure due to sorting.

Development and Training

We are using Keras to develop both our architecture and the comparison architecture. Keras was chosen due to the ease of use in leveraging Tensorflow and that it is a Python library. For each subset of images, the class of interest, we will train a new variant of the network. Regarding training we chose Google Colab as it provides free GPU access for a Jupyter notebook.

Performance Evaluation

In order to gauge the performance of our network we need some way to codify the subjective quality of a sprite. We are presenting the following subjective sprite quality metric,

1. Collect a random sample of 50 sprites
2. For each human rater,
    a. Determine if sprite is qualitatively acceptable, considering
        i. Coherence
        ii. Satisfaction
        iii. Usable
    b. Count number of acceptable sprites
3. Take average of raters to produce final quality metric

This metric can then be used to gauge the relative difference between the comparison architecture and our solution.

Current Progress

In the following sections we will cover our development process starting with the GAN created to generate handwritten digits, to a GAN to generate environment sprites, and finally, a GAN that generates weapon sprites. Note that the GAN created for this milestone is an inprogress version of our comparison architecture. In addition to what was shown, we also developed an autoencoder on the MNIST dataset to learn Keras. We plan on working off that implementation for our architecture.

GAN - MNIST Handwritten Digits

As a proof of concept while collecting our sprite dataset we developed a GAN to generate handwritten digits. Figure 2 displays several digits generated from the trained network. These results were generated after training the network for 20 epochs using the full MNIST dataset.

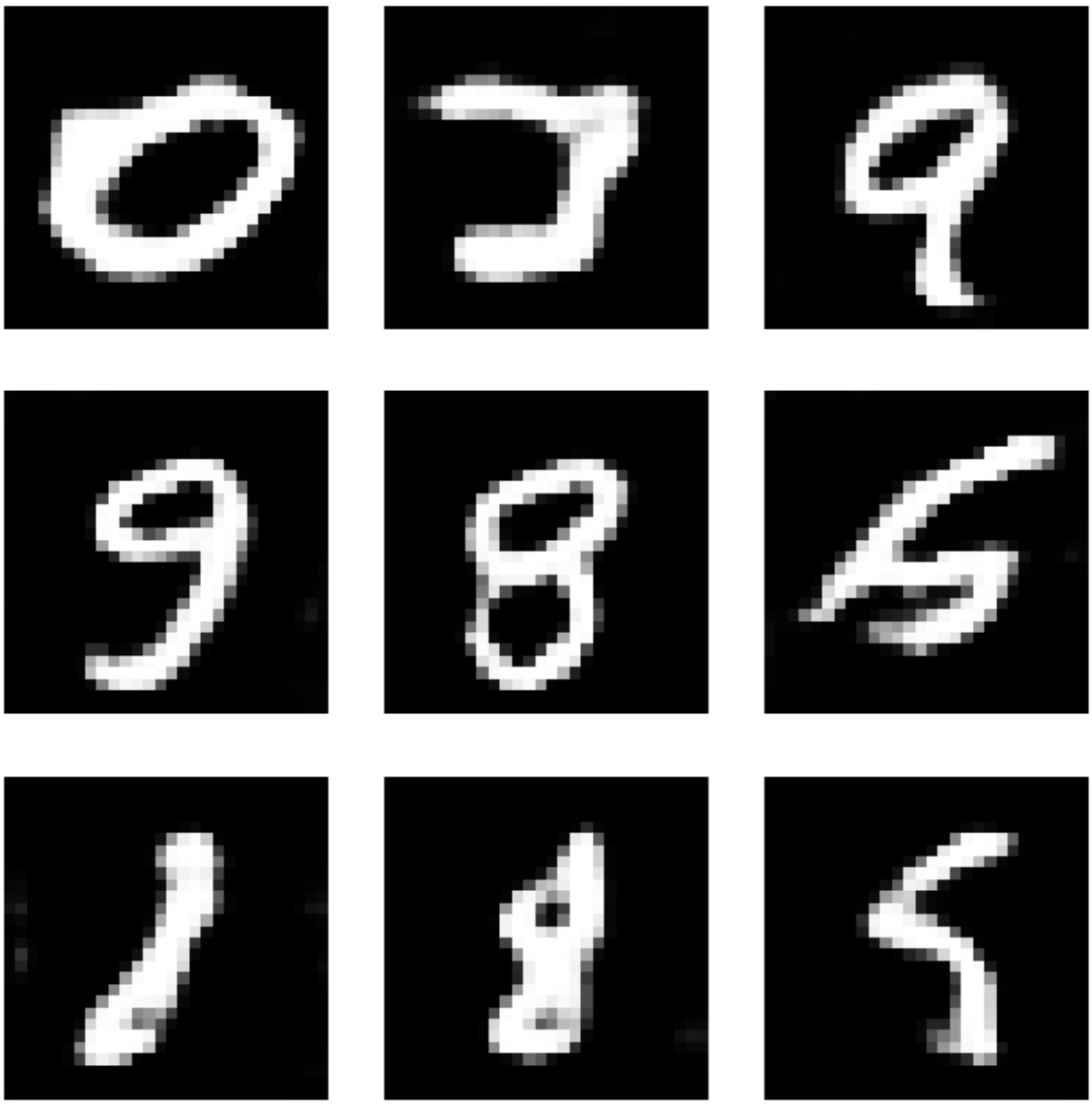

Figure 2: MNIST handwritten digits generated by GAN.

GAN - Environmental Sprites

Using an exploratory GAN based off the model architecture developed for the MNIST problem, we trained a network that generated terrain tiles, Figure 3. Our network was trained for approximately 2000 epochs to achieve these results. Most interesting sprites are the various block patterns that emerge as a feature. Note that some of the artifacts are exaggerated due to the sprites being scaled, however there is still much work to be done in generating valid sprites.

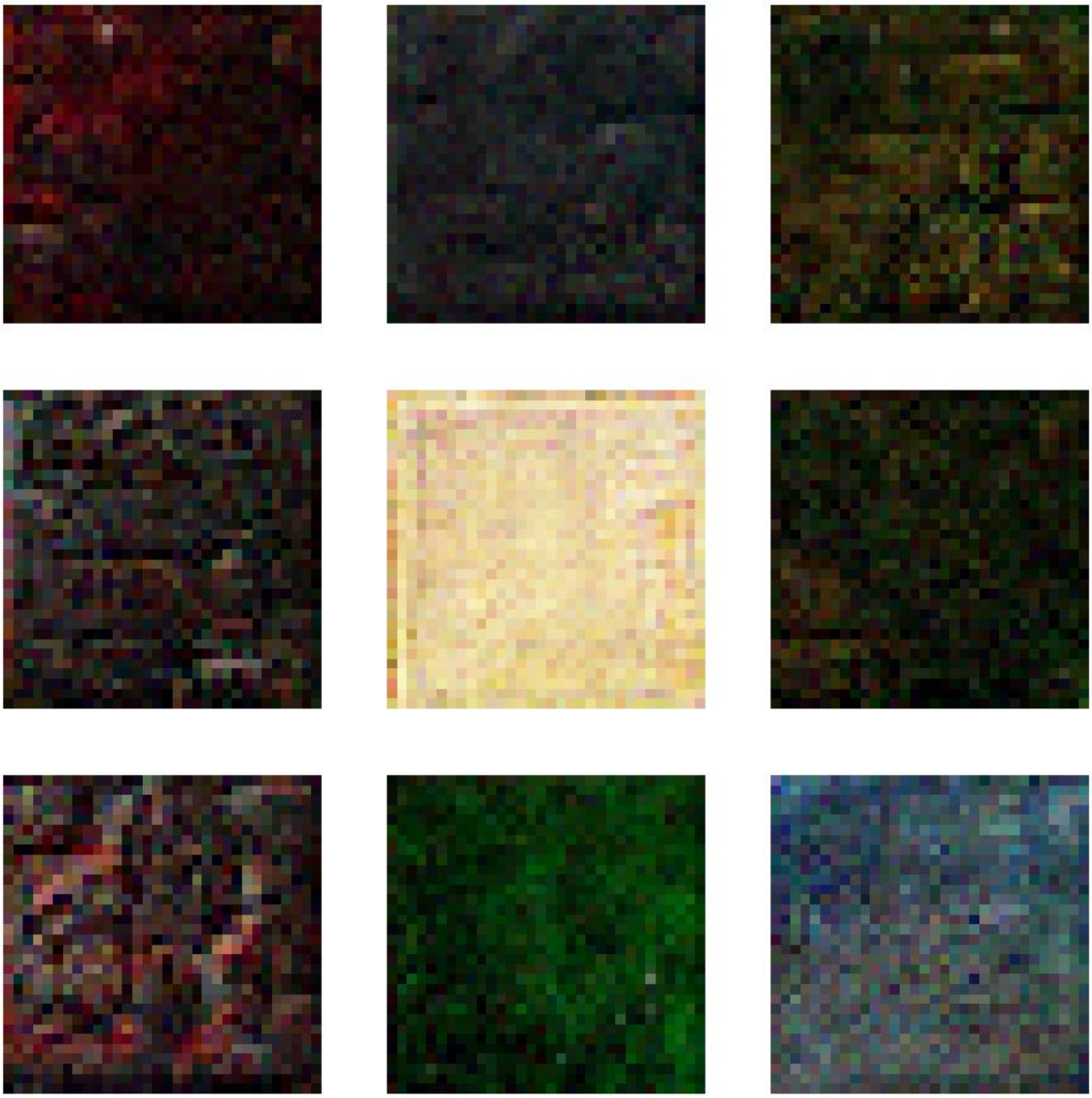

Figure 3: Environment sprites generated with the network created. Note that some of the blurring artifacts are generated due to scaling the 32x32 pixels.

GAN - Items Sprites

Using a similar network structure to the model that generated environmental sprites, we trained a GAN that could generate items, specifically our largest subset was various weapons like swords, staffs, maces, etc. Figure 4, shows what our network learned at approximately 500 epochs. A thousand epochs later, the results are shown in Figure 5. In both cases the network has difficulty learning the relevant details but has a general understanding of shape and shadow.

Figure 4: Weapon sprites generated with GAN. Note results are poor but understandable.

Figure 5: Results generated after further training of model. Improvement shown but quality in terms of color but it is still low.

Further modification to the DCGAN along with training the network for 3000 epochs resulted in a model that seemed to produce some valid results. Figure 6, shows a sample of these generated images, note that the network did not generalize well, instead learning only the exemplars in the dataset. For example the sword generated in the center is almost identical to one training sprite. We plan to address this issue by increasing our dataset and thus variance.

Figure 6: Potential overfitting of the limited weapons dataset after modifying network structure. Note that this set was randomly selected, the duplicates appeared by chance.

Website

Our website is hosted on github.io which provides free static hosting. The site is generated using Jekyll with the Grayscale bootstrap Jekyll theme. Currently the textual content is stubbed out. However we are using it to serve the compressed dataset for our online network training.

Next Steps

We will continue to collect and label sprite from online resources. We believe that the combination of more data in addition to developing on our DCGAN will improve on the results shown. Regarding our network training, we will continue to develop and tune the comparison network to provide its best possible results. Additionally, we will train the architecture on new subsets of our data such as entities. Finally, we will implement our conditional DCGAN architecture with autoencoder on the same datasets as was used for the comparison network.

Revised Implementation Plan

A revised time table is provided below. This revision is due to taking longer to learn Keras, develop GAN, and produce initial results. With current understanding and available tools, future work should be completed more smoothly.

Date	**Goal**
Sept. 23	Website started, Project infrastructure started.
Sept. 26	Proposal submitted. Website is updated.
~Oct. 20	Keras tutorials complete, basic mockup of DCGAN using MNIST handwritten digit dataset.
~Oct. 28	Initial dataset complete without image manipulation. Sample image set recorded for networks. Comparison network model implemented results recorded.
Oct. 29	Team decides to pursue data labeling and collecting more data from online sources. Team decides to attempt conditional GAN after autoencoder step. Team works on Mid-Term Report.
Oct. 31	Mid-Term Report submitted. Website is updated.
Nov. 11	Improvements to DCGAN model such as autoencoder, sample image set recorded. Dataset is extended.
Nov. 18	Conditional GAN is implemented on the autoencoder GAN model. Results recorded.
Nov. 25	Last minute changes are made. Final analysis is performed. Presentation write up started. Website results write up started.

Nov. 30	Website updated with all content. Presentation is complete.
Dec. 12	Final Presentations. (4:30-4:35)
Dec. 12	Project Website Submitted.

Table 1: Important dates for the project, note that the dates before Oct. 29th are approximate.

References

1. Hendrickx M., Meijer S., Velden J. V. D., Losup A. (2013) *Procedural Content Generation for Games: A Survey*. ACM Transactions on Multimedia Computing, Communications, and Applications. v.9, n.1, p.1-22. Online Access: https://dl.acm.org/citation.cfm?id=2422957
2. Horsley L., Perez-Liebana D. (2017) *Building an Automatic Sprite Generator with Deep Convolutional Generative Adversarial Networks*. IEEE Conference on Computational Intelligence and Games, p.134-141. Online Access: https://ieeexplore.ieee.org/document/8080426/?part=1
3. Mirza M., Osindero S. (2014) *Conditional Generative Adversarial Nets*. CoRR. Online Access: https://arxiv.org/pdf/1411.1784.pdf
4. Radford A., Metz L., Chintala S. (2015) Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. CoRR. Online Access: https://arxiv.org/pdf/1511.06434.pdf
5. Reed S. E., Zhang Y., Zhang Y., Lee H. (2015) *Deep Visual Analogy-Making*. Advances in Neural Information Processing Systems 28. P.1252-1260. Online Access: http://papers.nips.cc/paper/5845-deep-visual-analogy-making.pdf
6. Thanh-Tung H., Tran T., Venkatesh S. (2018) *On catastrophic forgetting and mode collapse in Generative Adversarial Networks*. ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models. Online Access: https://arxiv.org/pdf/1807.04015.pdf

Partner Contribution Breakdown

Curt Henrichs

- Wrote rough draft of mid-term report
- Edited and reviewed rough draft of mid-term report
- Collected and processed current sprite dataset
- Modified MNIST GAN to produce environment and item sprites
- Setup initial version of website using prebuilt theme

Sayem Wani

- Edited and reviewed rough draft of mid-term report
- Developed MNIST GAN
- Reviewed literature on training GANs

Saheen Feroz

- Edited and reviewed rough draft of mid-term report
- Developed MNIST GAN
- Reviewed literature on training GANs