Experimental AI in Games: Papers from the AIIDE Workshop
AAAI Technical Report WS-16-22

Predicting Resource Locations in Game Maps Using Deep Convolutional Neural Networks

Scott Lee, Aaron Isaksen, Christoffer Holmgård, Julian Togelius
NYU Game Innovation Lab, New York University, Brooklyn, NY 11201
sl3998@nyu.edu, aisaksen@appabove.com, christoffer@holmgard.org, julian@togelius.com

Abstract

We describe an application of neural networks to predict the placements of resources in StarCraft II maps. Networks are trained on existing maps taken from databases of maps actively used in online competitions and tested on unseen maps with resources (minerals and vespene gas) removed. This method is potentially useful for AI-assisted game design tools, allowing the suggestion of resource and base placements consonant with implicit StarCraft II design principles for fully or partially sketched heightmaps. By varying the thresholds for the placement of resources, more or fewer resources can be created consistently with the pattern of a single map. We further propose that these networks can be used to help understand the design principles of StarCraft II maps, and by extension other, similar types of game content.

Introduction

Procedural Content Generation (PCG) has proven a useful tool for game designers and developers as well as a rich and rewarding research topic for researchers in artificial intelligence and games (Shaker, Togelius, and Nelson 2015). While PCG applications and research has often focused on methods for producing game content with no or limited human input, there has been a recent focus on developing systems that can assist designers with game content creation, by providing suggestions, refinement, or feedback or otherwise collaborating with designers to augment their creativity.

Of course, research and practice in PCG (and other game AI fields) do not exist in a limbo, but are in a dialogue with game development and the wider field of artificial intelligence. In particular, new developments in artificial intelligence can often be appropriated for use in a game context, sometimes in unexpected roles and with unexpected results. In the past five years or so, a main trend in artificial intelligence has undoubtedly been the resurgence of neural network research and applications, after algorithmic advances and better hardware, as well as the availability of large data sets, have improved performance of such networks drastically. What is now referred to as *deep learning* is essentially the construction of neural networks with multiple layers for prediction, classification or unsupervised learning in data sets which typically exhibit very high dimensionality. Within the last few years, we have seen deep networks perform transformations and predictions from raw high-dimensional image data, something that was previously thought to be out of the league of machine learning (LeCun, Bengio, and Hinton 2015).

Could deep learning and high-dimensional data be used for PCG and AI-assisted game design tools? Yes, it would appear that this is a fruitful combination. The basic idea would be to learn recurring patterns from sets of game content artifacts, and use these patterns to generate more content or to assist human designers. This could be done by starting with some "seed" or "prompt" game content and generating the rest of the content by following the patterns learned from previous game content. While such a process could be run autonomously, it would probably be even more valuable within an AI-assisted game design tool, responding to the user's input with suggestions or evaluations based on the learned patterns. We will refer to this general approach as *data-driven procedural content generation*[1].

In this paper, we describe a data-driven PCG approach to partially generating *StarCraft II* maps. More specifically, we train (deep and shallow) neural nets to predict resource locations for existing Starcraft II maps. We then show that the trained nets combined with simple image processing techniques can, in some cases, accurately predict these locations on unseen maps, thereby capturing a persistent pattern in StarCraft II map design. We envision that these predictions can work as suggestions in an AI-assisted map creation tool, where the designer can be presented with suggestions for placing various map features based on incomplete designs at all stages of the map creation process.

Background

Real-time strategy games have featured frequently in game AI research, and within this genre the most common game is indisputably Blizzard's *StarCraft* (Blizzard Entertainment 1998). For several years, a series of competitions on playing StarCraft has been associated with CIG and AIIDE conferences, and numerous papers have been published on StarCraft-playing agents or some aspects of such agents (Churchill 2016; Ontanón et al. 2013; Churchill et

[1]It is also tempting to call it "game design by autocomplete".

al. 2016). But StarCraft has been used as a testbed not only for game-playing agents, but also for map generation. Togelius et al. used search-based methods for generating balanced StarCraft maps (Togelius et al. 2010; 2013). While search-based methods are capable of optimizing overall map properties, they are computationally expensive. Faster, constructive methods for StarCraft map generation have been proposed by (Uriarte and Ontanón 2013). Other methods have been applied to generate maps for other real-time strategy games, including Answer Set Programming to *Warzone 2100* (Brain and Schanda 2009) and a hybrid search-based and cellular automata method to *Dune II* (Mahlmann, Togelius, and Yannakakis 2012). None of these map generation methods were trained on existing game content, instead the relevant parameters for e.g. constraints and evaluation functions were hand-crafted.

Several types of AI-assisted game design tools have been proposed, usually based on a mixed-initiative co-creation paradigm, where both the software and the human designer has agency and can suggest or critique the artifact being designed (Yannakakis, Liapis, and Alexopoulos 2014). The *Tanagra* system for designing platform game levels (Smith, Whitehead, and Mateas 2011) and the *Ropossum* system for designing levels for the physics puzzler *Cut the Rope* (Shaker, Shaker, and Togelius 2013b; 2013a) are examples of tools which assist the human with designing levels for specific games. Of particular interest to the current work is *Sentient Sketchbook*, an AI-assisted map design tool for real-time strategy games operating on the level of somewhat abstracted map representations (Liapis, Yannakakis, and Togelius 2013). Using evaluation functions similar to those in (Togelius et al. 2010; 2013) Sentient Sketchbook provides continuous feedback on maps as they are being designed, and also suggests new maps that move the existing map design in various directions. As far as we are aware of, no existing AI-assisted game design tools suggest the placement of individual entities; also, none of them are data-driven.

Overall, little work has been done on data-driven PCG. A notable exception is Super Mario Bros. levels, where several methods have been presented for producing new levels from models learned from sets of existing levels. Dahlskog et al. devised a one-dimensional representation of Mario levels, allowing the use n-grams in the same way that simple method is commonly used for text generation (Dahlskog, Togelius, and Nelson 2014). The n-grams were trained on the levels of the originals Super Mario Bros. game and produced good-looking results but with limited variability. Snodgrass and Ontanon instead use a two-dimensional representation of the same levels, and train Markov Chains instead of n-grams (Snodgrass and Ontanon 2014). A similar approach was taken by (Guzdial and Riedl 2015), though they use geometry information inferred from gameplay videos for training set. Shaker and Abou-Zleikha used non-negative matrix factorization to train a model on content from five other generators to create a generator with a wider expressive range (Shaker and Abou-Zleikha 2014). An example from a different domain is Summerville and Mateas's work on generating maps for Zelda using Bayes Nets and Principal Component Analysis (Summerville and Mateas 2015).

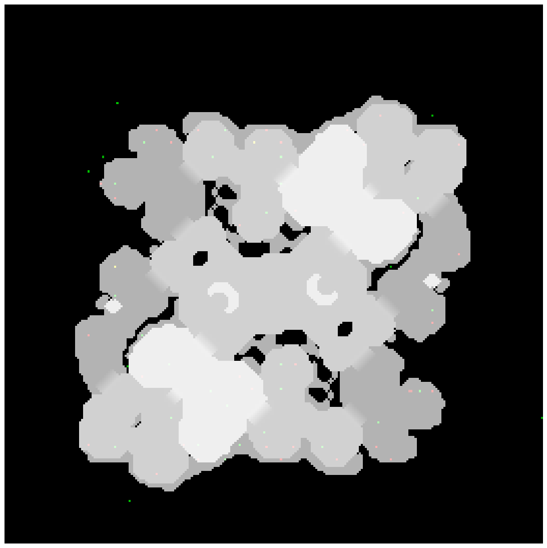
Figure 1: Heightmap generated from a StarCraft II map file.

Perhaps most closely related to the work undertaken here are those approaches that use neural networks. Hoover et al. evolved neural networks to output various types of tiles in Super Mario Bros. levels based on layers of other tile types, e.g. outputting question mark blocks and enemies based on ground and platforms; the training set was, again, the levels of the original Super Mario Bros. game (Hoover, Togelius, and Yannakis). Summerville and Mateas trained recurrent neural networks (Summerville and Mateas 2016) to generate Super Mario Bros. levels with good results and demonstrated the same approach with Zelda levels (Summerville et al.). In other domains, prominently the imitation of painting, progress has been made in transferring the style from a painting to another image using deep networks (Gatys, Ecker, and Bethge 2015) and deep network based systems that allow provide users with auto-completion by filling in details have been proposed and demonstrated (Champandard 2016) and served as inspiration for this work.

In the following section we give a brief introduction to StarCraft II maps and how we use them in this paper.

StarCraft II Maps

A Starcraft II map can be thought of as a grid of tiles, where each tile is a particular type of map content and may or may not contain object(s) and decoration(s). Any location on a Starcraft II map can therefore be expressed as point (X, Y), which would refer to the tile at row X and Column Y. This point can then be described in terms of its type and content. This representation of a map makes it feasible to express properties of a map as a two dimensional coordinate system.

Of particular interest to this project is the height of tiles, as they constrain unit movement, and the distribution of mineral objects across a map. Minerals represent key strategic locations for base expansion, and appear in all competitive maps. Vespene gas is important for the same reasons, and is typically found at the same locations as minerals. We therefore chose to combine minerals and gas into a single category which we will refer to as *resources*. Other objects are not guaranteed to appear in maps, and when they do, are typically very sparse. The work presented in this paper focuses on the gameplay aspects of StarCraft II maps, not the visual components. Therefore we represent maps as tiles that are characterized by their height and whether resources objects are present or not.

For this project, we use maps listing Blizzard as the author. The rationale behind this decision is that Blizzard, being the developers of StarCraft II, have a strong vision for their maps, as well as a motivation to make quality maps. Therefore, we believe treating Blizzard-made maps as a reference set of maps is a safe assumption for the intents and purposes of this project. The experiments conducted in this paper make use of maps created for the StarCraft II (Blizzard Entertainment 2010) game, as well as its expansions Heart of the Swarm (Blizzard Entertainment 2013) and Legacy of the Void (Blizzard Entertainment 2015). A total of 147 individual maps were collected. The map files, which were downloaded through either the Starcraft II game or the included map editor in Blizzard's `MPQ` map data archive format. We decompressed the StarCraft II map files and extracted height and resource data into our own reduced map format. Each of these extracted maps were rotated by 90, 180, and 270 degrees, and these rotated instances were added to our data set. In total, 528 instances were used in the training set and 60 were used in the validation set. Assignments to training and validation sets were chosen randomly.

Method

This section describes how we used neural networks to predict resource placement, and a post-processing step for user control of the amount of resources and grouping.

Because the map data is already represented as a Cartesian coordinate system, it is relatively easy to format the information stored in these files as images. As such, StarCraft II maps lend themselves well to network structures that deal well with image processing. We therefore use a deep convolutional neural network in order to investigate whether the topological features of a map are indicative of certain qualities, followed by some additional image processing for additional user control. The architecture and other parameters of this network were determined through experimentation until a well-performing network was found. In order to characterize this network we compare it to two baselines: a shallow convolutional network, and a simple, fully-connected one-layer network with linear activation. We start by presenting the output of our networks and then compare them to each other to understand the effect of using a deep network and the effect of using convolution.

Neural Network Architecture

The structure for the deep convolutional neural network is as follows. The input takes a 64x64 grayscale image for each game map, where values describe the height of the terrain (0 lowest, and 255 highest). The output is a 64x64 tensor describing the likelihood of placing a resource in this downsampled version of the game map. All convolutional layers use a 1 unit stride, and are followed by a *ReLu* activation function. The baseline shallow convolutional network uses a single convolution layer, and the simple network has a single, fully connected layer with a linear activation function. All network architectures are described in Table 1.

Table 1: Architectures of the three networks

Simple network

Layer	Activation	Size
Input		64x64
Fully connected	Linear	64x64

Shallow convolutional network

Layer	Activation	Size
Input		64x64
Convolution, 16 3x3 filters	ReLu	
Fully connected	Linear	64x64

Deep convolutional network

Layer	Activation	Size
Input		64x64
Convolution, 4 2x2 filters	ReLu	
Convolution, 16 3x3 filters	ReLu	
Convolution, 32 4x4 filters	ReLu	
Convolution, 64 5x5 filters	ReLu	
Fully connected	Linear	64x64

It is important to note that StarCraft II maps are not all the same size. However, because neural networks require consistency in the size of their inputs, the maps must be modified in some way to accommodate the neural network. We chose to center the map and pad the sides with impassable terrain to increase the size of the map image. The output of this neural network is a 2 dimensional tensor indicating potential locations for resources across the map. To optimize memory usage when training, we chose to use input and output image sizes of 64x64 which can later be scaled up to describe full size maps.

The filter sizes we used were found experimentally. Smaller filters resulted in the network being incapable of recognizing larger formations, such as plateaus and ravines, and larger filters resulted in the network being incapable of recognizing smaller features, such as ramps. *ReLu* was also chosen as the result of experimentation, as it was found that other activation functions resulted in noisier outputs.

To calculate loss for the training phase, we use a mean squared error criterion. We subtract the predicted 64x64 network output resource map from the training target resource map (calculated by scaling the original resource map down to 64x64). We solve for the network weights using the stochastic gradient descent solving method. All networks were trained for 200 epochs at a learning rate of 0.1. Empirical evidence showed that any additional training yielded

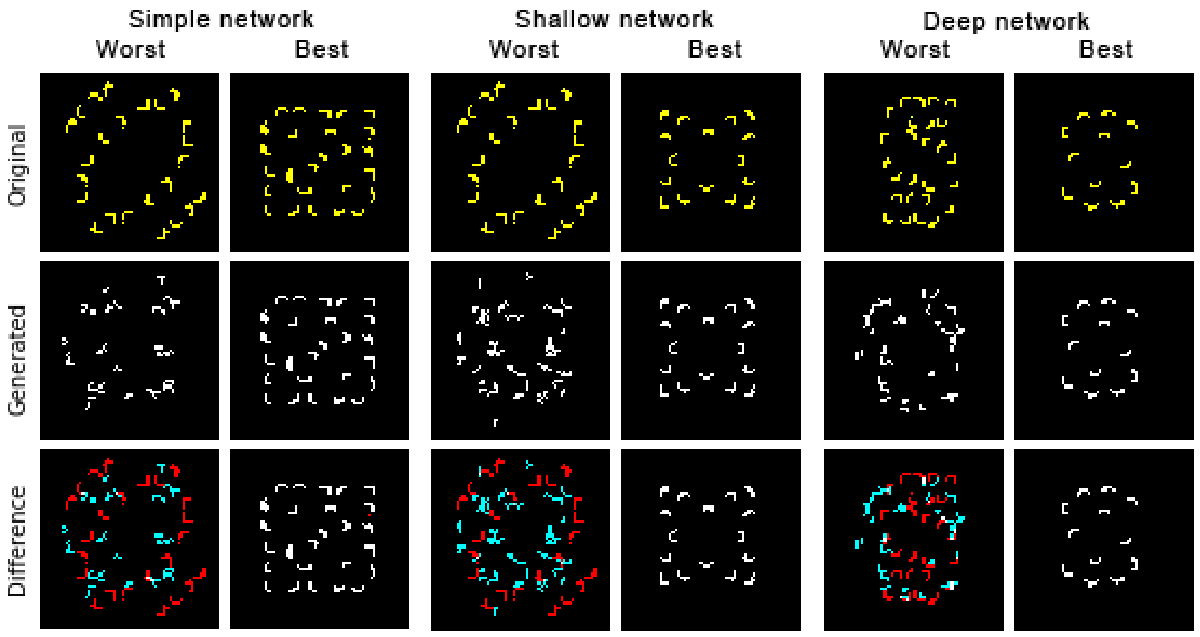

Figure 2: The best and worst examples from each kind of network after 200 epochs, as measured by the Mean Squared Error. In the best cases, all networks reproduce the original unseen validation image. In the worst case some general patterns seem to be recognizable by visual inspection, but especially minerals located in the middle of the map seem difficult for the networks to reproduce.

negligible gains, and was deemed unnecessary. The development in loss over the epochs are displayed in Figure 3 for both training sets and validation sets.

Generating Resource Locations

The outputs of the neural networks are values that are proportional to the likelihood of placing a resource at a particular location on the map. We then apply several additional image processing techniques to finalize the placement of resources. First, we use a threshold value on the output of the neural network to determine if we want to put a resource at a particular place on the map. By raising or lowering the threshold, we can place less or more resources, respectively.

Second, we then require that resources placed on the 64x64 map by the threshold step are also placed in clusters that have at least 3 connected tiles. Resources are not placed by themselves in Starcraft II, they always appear in groups. We look for all connected components on the map, and eliminate those that which only 1 or 2 tiles in the components. Neighbors are determined by their 8-neighborhood so diagonal connectivity is acceptable.

Threshold values can vary between maps, so the user can select a number of resources they would like on the map. An optimizer can then search for the ideal threshold that minimizes the squared difference between the number of resources placed and the target number of resources. This optimization can be done in real-time for the user to add or remove resources. If desired, the user can also target a number of clusters instead of a number of resources – this is done by counting the number of connected components during the small-cluster elimination phase.

Results

After the networks were trained on the training set (n=528), they were evaluated based on their performance on the validation set, a smaller set of maps (n=60) that the networks were not trained on. These outputs were then subjected to a simple 64x64 tile-level comparison with their originals in order to allow us to compare the networks' resource placements to the original resource locations placed by human designers. The tile-level precision of the placements are used to evaluate the performance of the networks, in addition to the mean squared error.

False negatives are of less interest to this project, if we assume that a suggestion system's value is derived more from its specificity than its sensitivity: For a designer using the system as a computer-assisted-design (CAD) tool, false positives would be distracting, as the system would produce unreasonable suggestions. False negatives, on the other hand, are less problematic, as they don't interrupt the designer, but of course mean that the designer is missing out on advice that might have been helpful.

While the outputs shown in Figure 2 seem acceptable, they show a large degree of variation across the validation set, with some maps performing very well and other maps performing poorly. We speculate this may be due, in part, to biases in the training set and in part due to biases in the evaluation method.

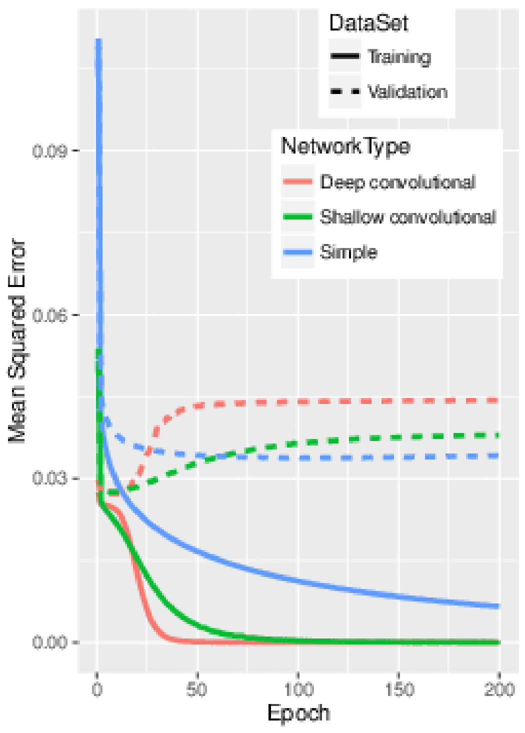

Figure 3: Mean squared error over 200 epochs of training.

Figure 3 shows how the loss of the networks changed over the course of training. The figures show how the simple network slowly converges on fitting to the training set, while the two convolutional networks quickly converge to a complete fit to their respective training sets. This shows that the convolutional networks quickly, around epoch 15, overfit to the training set and plausibly learn each map topology and resource distribution. This overall increases loss for the validation set, something that does not happen in the case of the simple network. The convolutional networks reproduce some maps from the validation set very well, while performing worse on the rest. This is likely due to the low amount of training maps available, but we also speculate that this may indicate that StartCraft II maps exist in stylistic classes, and the convolutional networks learn a class that is overly represented in the training set. Methods for classifying StarCraft II maps would be necessary to explore this hypothesis further, but visual and qualitative inspection can support this idea. Figure 2 shows examples of maps that were generated by each kind of network. On the top row in yellow are the original maps. In the middle row in white are the generated maps. On the bottom row are displayed the differences between the resource placements with true positives (generated content agrees with original) in white and false positives (generated content places resources in empty spaces in the original) in cyan. False negatives (generated content has no resource where the original does) are indicated in red.

In general, it seems that the networks perform better on maps where resources are closer to the edges of the map and worse in cases where minerals are scattered around the center of the map.

In terms of using the convolutional nets for design support tools, they seem potentially useful for the maps where they perform well. Figure 4 shows how applying the combination of the network output and the threshold works for one of the maps for which the deep network performs well. It shows how changing the threshold for different amounts of resources changes the output of the combined network output and post-processing steps. Using this approach demonstrates how a designer could use a trained network to either add or remove resources from a given map, depending on her objective. The original maps given to the network are displayed on the left in yellow and variations on content amounts are displayed increasing from left to right in white. For maps where the method performs well, we believe this could have straight forward applicability for a map designer.

Discussion

The results presented above show that this method clearly has potential for placing resources in real time strategy game maps. Still, the method and the study suffer from a number of weaknesses that should be investigated and refined in future work. The performance of the predictions is, as demonstrated in the figures above, convincing for some maps, but lacking for others. This seems to be due to overfitting to biases in the training set and/or the relationship between the training set and the validation set. Additional data sets could be included, or more randomized training and testing runs could be applied to further investigate this. Because our network outputs are, by nature, probability distributions, often with a certain degree of noise, sensitivity and specificity can be thought of as mutually exclusive. Therefore, it is worthwhile to discuss which of the two is more important to this project, especially when considering its potential value as a game design aide. Specificity may be much more valuable to novice designers, especially those without the knowledge necessary to accurately evaluate suggestions given to them.

These users would have a more difficult time recognizing weak suggestions, and may be overwhelmed by too large an option set. A smaller option set with a stronger guarantee that all of the presented options are of quality would be more useful to this class of less experienced designers. A high sensitivity would be of more value to users who are already capable of recognizing the value of suggestions for themselves. One obvious consequence of limiting an option set is that it severely limits variety and novelty. Because this is a data driven approach to a design aid, the highest confidence suggestions are going to be those consistent with maps that already exist. For designers with a strong degree of domain knowledge, a lower threshold may present options that are possibly viable, but not necessarily consistent with the training set. So long as the threshold is high enough to filter out the noise present in the system, the results have demonstrated that this approach to resource placement suggestion can offer substantial results.

In this system, we used data from just 147 maps (though the number of instances in the data set was quadruple that because of rotations). This is a relatively small training set, in particular given the very large number of parameters for

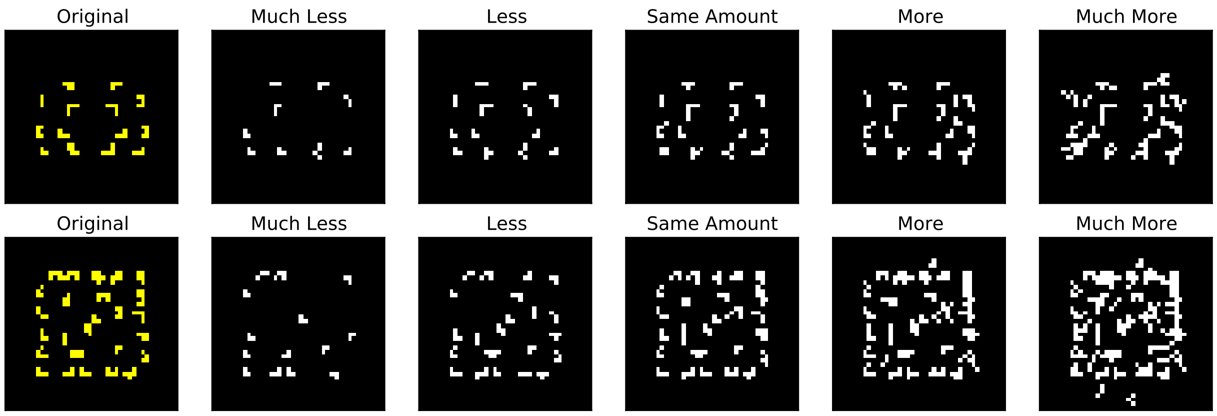

Figure 4: Two examples how changing the desired number of resources changes which parts of the network output are kept, allowing a designer to ask for either less or more content, relative to the original. A designer could use a slider to control this.

the deep network, which could contribute to the apparent overfitting. Most successful cases of deep networks on high-dimensional data employ much larger datasets, typically orders of magnitude larger. It is therefore worth considering dropping quality standards for maps, and including any human-made maps we can scrape from the internet in our dataset in future work.

It is also very much worth investigating combining supervised training seen here with unsupervised methods. In particular, recent work has shown that autoencoders can be used to learn abstract representations of 2D game levels; these representations can then be used to generate new levels adhering to design constraints, to classify levels and to repair levels (Jain et al. 2016). These methods would presumably work well on StarCraft maps as well, and could be used in tandem with supervised training to generate further suggestions in an AI-assisted map design tool. It is also possible that the compressed level representations themselves can offer useful inputs for networks predicting resource placements, or other map features.

Finally, a user study implementing this method into an actual level design tool and collecting feedback from map designers would validate the output of the method and utility for practicing game designers. The need for human validation is especially apparent considering weaknesses in the mean squared error method of evaluation. MSE employs a pixel-level subtraction between outputs, which means that a mineral placement can still be punished if it is even one pixel off. The goal of procedural generation is high quality content, a metric that is extremely difficult to express quantitatively, and we have come to believe that MSE does not adequately represent the quality of a mineral placement.

Conclusion

In this paper, we have demonstrated how deep networks can be used to learn and predict design features of maps for real-time strategy games, specifically StarCraft II. By extracting heightmaps and resource location maps from custom StarCraft II map data files, we constructed a dataset which was used to train a deep convolutional network. In spite of a relatively small data set, this network was capable of predicting resource locations in unseen maps with examples of high precision. Future work that would increase the efficiency of training and the precision of predictions was outlined. Altogether, the results presented here suggest that deep convolutional networks can be trained by example to drive computer assisted design tools for real time strategy game level designers.

It is likely that this approach could be extended to other games such as e.g. arena-based first-person-shooter games, open-world games, or possibly any kind of game where the placement of features of interest and strategic importance are motivated by the topology of levels. Due to the fast output processing capabilities of such networks, it would be feasible to add suggestion capabilities to existing level design tools, enabling a form of on-line autocompletion for level designers or including them in procedural content generation systems.

References

Blizzard Entertainment. 1998. *StarCraft*. Blizzard Entertainment.

Blizzard Entertainment. 2010. *StarCraft II: Wings of Liberty*. Blizzard Entertainment.

Blizzard Entertainment. 2013. *StarCraft II: Heart of the Swarm*. Blizzard Entertainment.

Blizzard Entertainment. 2015. *StarCraft II: Legacy of the Void*. Blizzard Entertainment.

Brain, M., and Schanda, F. 2009. *DIORAMA (Warzone 2100 map tools)*.

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

Churchill, D.; Preuss, M.; Richoux, F.; Synnaeve, G.; Uriarte, A.; Ontanón, S.; and Certickỳ, M. 2016. Starcraft bots and competitions.

Churchill, D. 2016. *Heuristic Search Techniques for Real-Time Strategy Games*. Ph.D. Dissertation, University of Alberta.

Dahlskog, S.; Togelius, J.; and Nelson, M. J. 2014. Linear levels through n-grams. In *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, 200–206. ACM.

Gatys, L. A.; Ecker, A. S.; and Bethge, M. 2015. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

Guzdial, M., and Riedl, M. O. 2015. Toward game level generation from gameplay videos. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*.

Hoover, A. K.; Togelius, J.; and Yannakis, G. N. Composing video game levels with music metaphors through functional scaffolding.

Jain, R.; Isaksen, A.; Holmgård, C.; and Togelius, J. 2016. Autoencoders for Level Generation and Style Identification. In *Second Computational Creativity and Games Workshop*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436–444.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2013. Sentient sketchbook: Computer-aided game level authoring. In *FDG*, 213–220.

Mahlmann, T.; Togelius, J.; and Yannakakis, G. N. 2012. Spicing up map generation. In *Applications of evolutionary computation*. Springer. 224–233.

Ontanón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game ai research and competition in starcraft. *Computational Intelligence and AI in Games, IEEE Transactions on* 5(4):293–311.

Shaker, N., and Abou-Zleikha, M. 2014. Alone we can do so little, together we can do so much: A combinatorial approach for generating game content. *AIIDE* 14:1–1.

Shaker, N.; Shaker, M.; and Togelius, J. 2013a. Evolving playable content for cut the rope through a simulation-based approach. In *Ninth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Shaker, N.; Shaker, M.; and Togelius, J. 2013b. Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. In *AIIDE*.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2015. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

Smith, G.; Whitehead, J.; and Mateas, M. 2011. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *Computational Intelligence and AI in Games, IEEE Transactions on* 3(3):201–215.

Snodgrass, S., and Ontanon, S. 2014. A hierarchical approach to generating maps using markov chains. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Summerville, A. J., and Mateas, M. 2015. Sampling hyrule: Multi-technique probabilistic level generation for action role playing games. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

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

Summerville, A. J.; Behrooz, M.; Mateas, M.; and Jhala, A. The learning of zelda: Data-driven learning of level topology.

Togelius, J.; Preuss, M.; Beume, N.; Wessing, S.; Hagelback, J.; and Yannakakis, G. N. 2010. Multiobjective exploration of the starcraft map space. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, 265–272. IEEE.

Togelius, J.; Preuss, M.; Beume, N.; Wessing, S.; Hagelbäck, J.; Yannakakis, G. N.; and Grappiolo, C. 2013. Controllable procedural map generation via multiobjective evolution. *Genetic Programming and Evolvable Machines* 14(2):245–277.

Uriarte, A., and Ontanón, S. 2013. Psmage: Balanced map generation for starcraft. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 1–8. IEEE.

Yannakakis, G. N.; Liapis, A.; and Alexopoulos, C. 2014. Mixed-initiative co-creativity. In *Proceedings of the 9th Conference on the Foundations of Digital Games*.