Continuous Procedural Network of Roads Generation using L-Systems and Reinforcement Learning

Ciprian Paduraru, Miruna Paduraru and Stefan Iordache
University of Bucharest, Romania

Keywords: Networks, Roads, Deep Learning, Simulation Software, Video Games, L-systems, Reinforcement Learning.

Abstract: Procedural content generation methods are nowadays used in areas such as games, simulations or the movie industry to generate large amounts of data with lower development costs. Our work attempts to fill a gap in this area by focusing on methods capable of generating content representing network of roads, taking into account real-world patterns or user-defined input structures. At the low- level of our generative processes, we use L-systems and Reinforcement Learning based solutions that are employed to generate tiles of road structures in environments that are partitioned as 2D grids. As the evaluation section shows, these methods are suitable for runtime demanding applications since the computational cost is not significant.

1 INTRODUCTION

The environments used by computer games, simulation applications, and movies have grown significantly in recent years. Nowadays, most users demand large open-space worlds with lots of content. Procedural content generation is currently used to generate textures (Dong et al., 2020), animations (Danny Alberto et al., 2019), and virtual worlds (Freiknecht and Effelsberg, 2017). Open-world games such as Minecraft [1], NoMansSky,[2], the Far Cry series, or movies such as The Mandalorian [3] use procedural content generation to increase the size of environments and reduce production costs.

Our work addresses the problem of procedural generation of road networks that bear a resemblance to real cities. Our motivation stems firstly from the set of applications that could use such a system, such as simulators for self-driving cars, driving simulation environments, and computer games. Second, after a literature review, we find that there is a gap in this area. The contribution of our current work can be summarized from two points of view:

- From the authors' knowledge, this is the first work that builds generative models for networks of streets that mimic real-world cities. Our methods are based on statistical learning of features, which are then sampled by L-Systems (Curry, 2000) or training Reinforcement Learning (RL) agents to generate the networks.
- Provide an augmented dataset and operations where the road network pictures from different real cities are converted into a graph format suitable for further research in the community. We believe that the methods and code used for processing can be easily adapted to other inputs such as maps from Open Street Maps [4] or Google Maps.

The contributions are also made open source at https://github.com/AGAPIA/ProceduralContentGeneration.

The rest of this paper is organized as follows. Section 2 describes previous work on procedural content generation, focusing on map generation. Section 3 presents the processing methods for converting top-down pictures of cities into augmented graph structures. Our methods and procedural content generation proccesses are described in Section 4. The evaluation of our work is considered in Section 5. The final section contains a conclusion and plans for future work.

[1] https://www.minecraft.net/en-us

[2] https://www.nomanssky.com/

[3] https://www.theverge.com/2020/2/20/21145671/mandalorian-sets-stagecraft-epic-games-ilm-fortnite-baby-yoda-digital

[4] https://www.openstreetmap.org/

Paduraru, C., Paduraru, M. and Iordache, S.
Continuous Procedural Network of Roads Generation using L-Systems and Reinforcement Learning.
DOI: 10.5220/0011268300003266
In *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022)*, pages 425-432
ISBN: 978-989-758-588-3; ISSN: 2184-2833

2 RELATED WORK

Important content of the current top video games use the so-called 'Procedural Content Generation' (PCG) to improve gameplay and make the game more interesting and entertaining for players (Liu et al., 2020). We will analyze some methods that have been used so far in content generation, focusing primarily on map generation. Dynamic map generation for video games has been studied in many papers in the literature. In this section we will try to describe the state of the art in this field and compare it with our current work.

L-Systems where used in (Parish and Müller, 2001) to create procedural cities. Compared to our method, their method is purely generative and receives as inputs goals and constraints and does not learn a model from real data topologies. Tensor fields were also used in (Chen et al., 2008), but with the same drawbacks, since it is a purely generative method based on constraints. In (Lara-Cabrera et al., 2012), the authors use a genetic algorithm to generate and evolve balanced maps (i.e., maps that provide equal advantages and disadvantages to all players) for real-time strategy (RTS) games. Their method generates content offline using a generate-and-test schema. The algorithm runs over a set number of generations with a tunable population size, where each individual maps 84 different parameters. The selection process is done in a tournament way and uses a mutation rate of $\sim 70\%$ and a crossover rate of 75%. To calculate an individual's score, a genotype-to-phenotype transformation was used. Evaluating how balanced a generated map is, it is a costly process where two bots play the game until the end and evaluate the balance coefficient. In (de Araújo et al., 2020), Particle Swarm Optimizations (PSO) techniques are used to generate balanced maps that simultaneously satisfy a set of soft user-specified requirements. Their method assigns a particle to each tile of the map (which is organized in a grid structure similar to our solution) and penalizes each tile for deviating from the user-suggested requirements. However, their generative process uses random heuristics to some sense. Applying such methods to structures with constrained geometry, such as 2D graphs in our case, would lead to poor results and lose the expressiveness of a real-life example.

In (Snodgrass and Ontañón, 2017), a more statistical approach to content generation is presented that uses multidimensional Markov Chains (MdMC) to generate tiles of simple 2D levels in games. The MdMC is trained using existing data and then sampled to generate more content at runtime. Although its approach is interesting for our purpose, it is limited to generating game levels where each tile could define a piece in the world, which could not lead to learning the structure of smooth roads in a city's road network. The same limitation is found in (Ping and Dingli, 2020), which uses Conditional Generative Adversarial Networks (cGAN) to generate 2D or 3D maps based on a user-specified sketch. In addition, this method needs the generation of a sketch of the map first, and it is difficult to adapt it to a constantly moving visualization perspective inside the environment.

An approach that uses reinforcement learning is described in (Gisslén et al., 2021). Their method consists of two RL agents: the generator and the solver, which are interdependent. The generator creates an environment, which is then tested by the solver. The generator receives feedback (rewards and observations) from the solver, as well as some additional content specified by the developers. In this way, the generator learns to create different types of environments of varying types, difficulty, and complexity. After the generator creates a new environment, it challenges the problem solver to produce the best possible result. As a result, the problem solver becomes more robust and is more likely to solve new, unprecedented tasks and reduce the amount of hard coding required to solve the task. However, we note that the method is not suitable for our purposes because the computational cost of generating new tiles with connected roads would not be suitable for runtime generation.

Given our goal of learning graphs of networks and then training a model that creates similar looking graphs, an approach such as NetGAN (Bojchevski et al., 2018) that uses Generative Adversarial Networks (GANs) could also be used. However, the problem with NetGan is that it only supports undirected graphs and cannot generate continuous networks as we need in a procedurally generated road network. Another approach is Misc-GAN (Zhou et al., 2019), where graphs are fed into a GAN architecture that tries to extract a distribution for the given graph dataset at different levels of granularity. The network learns this distribution and then attempts to apply it to the patterns being generated. Like the previous approach, this method is not suitable for generating continuous road networks, only individual samples, which is not of great use for our goal. The work presented in GraphGAN (Wang et al., 2017) and VGAE (Kipf and Welling, 2016) is designed for datasets where the user defines the set of nodes and the model can then fill in the edges. However, our goal is to generate both nodes and edges, which means that the approach they use is not suitable for our requirements. We try to overcome these limitations with our proposed solutions based on reinforcement learning.

3 DATASET AND PROCESSING ALGORITHMS

To model a generative method that learns from real data, we will use the semantic representation of the CITY-OSM dataset (Li et al., 2019). The dataset consists of 1671 aerial images of the cities of Berlin, Chicago, Paris, Potsdam, Tokyo, and Zurich, which together comprise nearly 45 GB. The raw images were segmented into a simpler representation consisting of roads (marked with the color blue RGB(0, 0, 255)), buildings (marked with the color red RGB(255, 0, 0)), and the rest of the background (marked with the color white RGB(0, 0, 0)). Note that similar input content can be added by computer vision methods directly from Open Street Map or Google Maps. Therefore, we believe that the methods described below are also suitable for customized and new input maps.

3.1 Detecting Road Nodes and Edges

We apply a heuristic to detect the **nodes** (vertices) of the graph G describing the roads in the scene. The heuristic was built as needed on top of the ideas defined in (Mena and Malpica, 2005) to fix some issues with previous approaches and our dataset. We assume that the nodes should be located at the elements of interest: intersections, road ends, and turning points. For this purpose, we take each white point P in the given input image (representing a road) and draw a centered circle in P with radius R (empirically, the value was set to 5 pixels). If the number of intersecting points (from different road segments) is at least 3, we mark P as an intersection point (Figure 1a, 1b). If only a single point is intersected, it means that P is an endpoint of the road (Figure 1c, 1d). To detect nodes that are turning points, we require that a pair of distinct nodes in the set of circle intersections form a turning point of less than TA degrees (empirically, we chose this value to 90 degrees), Figure 1e. Finally, we discard points in G that are physically closer than TD pixels in the input image (chosen as 30 pixels). The pseudocode of these operations is shown in Listing 1, while a visualization of a complete example is shown in Figure 2.

To detect the edges of the graph, a Breadth-First Search (BFS) is used starting from each node and following the white pixels in the image. Since the road is larger than one pixel, it is possible to go beyond a node when searching for connections. To avoid this, we define a zone of influence R for a node so that the search for a connection node stops when it is in this area.

At this point, the graph constructed in G has the network of nodes and road links between them. However, in order to learn efficiently from the data, we needed some other features that are stored internally:

- The distances between nodes, $D(V_i, V_j)$.
- The angles of pairs of nodes representing turning points, $Angle(V_i, V_j)$.
- How many branches each intersection node has, $Inter(V_i)$.

4 METHODS

Having a dataset of examples in a graph format like the one described in Section 3, the goal is to continuously generate similar graphs that adhere to the given patterns and some constraints. In a computer game or simulation environment, the environment is assumed to be rendered at a given location (X, Y). We assume that the environment can be split into a grid structure (Figure 3), the rendering effects appear on the current tile (by default of size 1km in our environment) and the other 8 tiles in the neighborhood, and each time the visibility is moved to another tile, a new network of roads must be generated for the new visible tiles, while the older tiles that are no longer visible can be destroyed.

As for the generation process for a single tile, intuitively, if one wants to generate a road network similar to the streets of Manhattan (New York), the streets should be created primarily in a grid pattern, rather than the streets of Paris, which have other patterns. This intuition gave us the idea that the sample graphs in the database could be grouped into clusters representing different levels of granularity (e.g., cities, neighbourhoods, etc.). A generative model could then learn the patterns of each cluster and later be used to create similar looking road networks. Formally, we divide the database into NK clusters. In the continuation of this section, we describe the clustering and the two proposed methods for learning generative models for road networks.

4.1 Clusterization Process

To statistically store the information from each graph G, and considering that the size of a single physical example is 512×512 pixels, we create the following measures in order:

(a) Intersection example. (b) Intersection example. (c) Road end point. (d) Road end point. (e) Turn point nodes.

Figure 1: Examples of applying the heuristic to find different types of nodes in the roads graph.

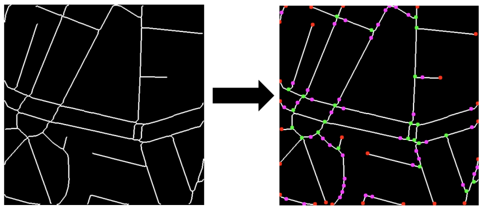

Figure 2: An example of node detection in a processed image. The nodes at intersections are marked with green color, the nodes at road ends are marked with red color and the nodes at road turning points are marked with magenta color.

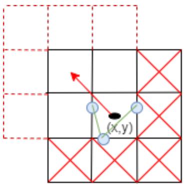

Figure 3: The environment is divided into rectangular tiles of equal size. If you change the position of the visualization from one tile to another, as indicated by the red arrow, new tiles - the dotted red ones - must be generated, while unneeded tiles - the ones crossed in red - can be deleted to save computational resources. The green lines represent a possible road network within the current tile. The blue points located at the intersection between the roads within the tile and the boundary represent the boundary set $\mathcal{B}$ of the current tile.

1. Group the distances between the nodes into 10 equal bins that are between $10 - 512 \times 512 \times \sqrt{2}$. We further denote this as the set G_D, i.e. $G_D[k]$ indicates how many distances exist in the group k in the graph.
2. Group angles representing turning points in 10 equal bins between 0-180. We refer to this as the set G_A, i.e. $G_A[k]$ indicates how many turning points with an angle in group k exist in the graph.
3. Aggregate the different numbers of branches between 1 and 10 and group them in the set G_I, i.e. $G_I[k]$ indicates how many intersections with k branches exist in the given graph.
4. Count the number of intersection nodes, G_{NI}, the total number of end roads, G_{NE}, the number of nodes that are turning points, G_{NA}, and the number of nodes that connect only straight nodes, G_{ND}. The sum of all interesting nodes in G is then $G_T = G_{NA} + G_{NE} + G_{NI} + G_{ND}$

The next step is to use the KMeans (Hartigan and Wong, 1979) algorithm to group the graphs into K (user-specified) groups. In practice, we have concluded that it is best to apply clustering to graphs that are from the same city. Attempting to cluster Chicago and Paris, for example, would result in an interpolation of sparse input data that ultimately does not represent either city. Instead, it is better for results to apply the clustering mechanism to each city individually and then consider different clusters as parts of the city. The use of standard statistical methods such as deviations computations are required in practice to ensure that the data being aggregated and clustered have the desired input structure. As a similarity metric between two graphs G_1 and G_2, we used the absolute difference between each pair of features described in the four metric categories above. The statistics for the graphs that are part of a cluster K are then aggregated by summing the values for $G_T, G_{NA}, G_{ND}, G_{NE}, G_D, G_A, G_I$ (we continue to denote the aggregations per cluster by K_{type} instead of G_{type}). The following measures and operations are additionally added for each cluster K to be used later in the generation process:

- Probability of a turning point: $P_{KA} = \frac{K_{NA}}{K_T}$.
- Probability of an intersection point: $P_{KI} = \frac{K_{NI}}{K_T}$.
- Probability of an endpoint: $P_{KE} = \frac{K_{NE}}{K_T}$.
- Probability of a straight road continuation: $P_{KD} = \frac{K_{ND}}{K_T}$.
- A distribution probability for each of the sets K_D, K_I, K_A, representing the distribution of values for

Listing 1: Pseudocode for adding the road network nodes inside the graph G.

```
G = {}
for each white pixel P(x,y):
Draw a circle of radius R in centered in P
S = set of points (from distinct segments) intersected by the circle
if |S| >= 3: # intersection case
Set P as intersection node
G = G U {P}
elif |S| == 1: # road endpoint case
Set P as road endpoint
G = G U {P}

# detect turning points
for each distinct pair P1, P2 of points in S:
if angle((P1-P), (P2-P)) < TA:
Set P1 and P2 as turning points
G = G U {P}

Eliminate nodes from G closer than TD distance from each other
```

the set of straight road distances between nodes, the number of intersections, and the turning angles. A Gaussian mixture model (Reynolds, 2009) with a number of 3 components (empirically chosen after plotting values from the dataset) is used and trained using Expectation-Maximization algorithm (Heskes et al., 2004). We further denote a sampling process from these as operations: $SampleDist(K)$, $SampleNumBranches(K)$, and $SampleTurnAngle(K)$, respectively.

4.2 Generating a Network of Roads for a Single Tile

At each time the position of the visualization is moved between tiles, new tiles must be generated as shown in Figure 3. The proposed process of generating a road network for a single tile at position $(\mathcal{X},\mathcal{Y})$ involve the following steps:

1. If the boundary set of points of the tile, $\mathcal{B} \neq \emptyset$ (see below for more details on its construction), then choose a starting position $(x,y) \in \mathcal{B}$. Otherwise, start from a random position (x,y) inside the tile.
2. Set as initial road draw direction the vector from (x,y) to the center of the tile.
3. Select (manually or randomly) a cluster K to serve as the representative data sample for the tile.
4. Apply one of the methods defined in Section 4.3 or Section 4.4.

4.3 Generative Model using Lindenmayer Systems

Parametric and likelihood-based variants of Lindermayer Systems (L-Systems) are used for the first generative model that can generate road networks. For better understanding of this section, we briefly describe how L-systems are used in our specification. More details for the interested reader about these systems and variants can be found in (Rozenberg and Salomaa, 1980), (Curry, 2000). The L-systems used in our case consist of a start symbol, a set of rules, and a set of end conditions. Each rule can optionally specify a probability for its application, and each symbol (module) that is modified can have a set of parameters. A concrete example: $A\,(x,y) : P \rightarrow A\,(h,z)$ means that the rule is applied to a symbol (module) A with initial parameters x and y, has a probability P of being applied, and when applied, changes the values of the two parameters to h and z, respectively.

In the following, we give the rules for building an L-system using the statistical data defined in Section 4.1. The system is capable of drawing a complete road network within a tile starting from given starting points.

- Start symbol: $\mathcal{X}$.
- Parameters: $\mathcal{X}\,(x,y,dir)$, where (x,y) is the current 2D coordinate in the tile to continue drawing the road network. The initial position can be chosen randomly or can start from the boundary of a neighbouring tile (the set $\mathcal{B}$ defined below in the stop condition). The current drawing direction is

parameterized by d, which is a 2D vector. The number of operations on the same path is stored internally in a variable *age* (not shown in the rules below for simplicity), which is mainly used to stop the process when possible infinite loops are detected.

- Rules: Draw a sample from a uniform random variable (0-1), then decide which item to generate next based on the probabilities and using a roulette wheel, in the following order:
  - Draw a straight connection from a previous point and keep the same direction:
    $$\mathcal{X}(x,y,d) : P_{KD} \rightarrow \mathcal{X}((x,y) \dotplus d \times r, d)$$
    $$, where\, r = SampleDist(K)$$
  - Change the current direction with a new sampled direction:
    $$\mathcal{X}(x,y,d) : P_{KA} \rightarrow \mathcal{X}(x,y,newDir)$$
    $$, where\, newDir = SampleTurnAngle(K)$$
  - Create a road end:
    $$\mathcal{X}(x,y,d) : P_{KE} \rightarrow \mathcal{E}(x,y)$$
    $$, where\, newDir = SampleTurnAngle(K)$$
  - At an end of road symbol we stop the process:
    $$\mathcal{E}(x,y) \rightarrow \emptyset$$
  - Create an intersection with a sampled number of branches and split the drawing process into several parallel directions:
    $$\mathcal{X}(x,y,d) : P_{KI} \rightarrow \mathcal{X}(x,y,nd_1)|...|\mathcal{X}(x,y,nd_{nB})$$
    $$, where\, nB = SampleNumBranches(K)$$
    $$, and\, nd_i = SampleTurnAngle(K)\, with\, i \in \overline{1 \ldots nB}$$
- Condition for a hard stop: the coordinate parameters (x,y) are outside the specified tile. In this case, the algorithm stores the intersection point between the previous coordinate set and (x,y) as a boundary point and adds it to a group of points $\mathcal{B}$.

At each step defined above, the new position (x,y) is added as a new node to the generated graph, with the appropriate label and with an edge connecting the previously generated point. To ensure the quality of the results, we do not allow self-intersection during generation, i.e., we rerun the sample if the new position would lead to self-intersection.

4.4 Model using Reinforcement Learning

For this task, we have currently chosen the Double Q-Learning (DDQN) method (Hasselt, 2010).

The set of *actions* that the agent can perform at each step is similar to our previous approach with Lindenmayer Systems. At each step, the agent must choose between creating a straight road segment of a certain length, changing the drawing angle by a certain number of degrees, creating an intersection with a certain number of branches, or creating an end of road. The actions are thus encoded in this order with indices between $0-3$. The specific values to be applied are sampled from the learned data distribution as before.

The *state* of the environment seen by the agent at each step is defined by a state composed of 10 input parameters: the number of nodes of each type, the length of the last segment, the direction of the current drawing direction, the current and previous drawing positions, the branches of the last intersection, and the last action performed. These inputs are passed through two fully connected neural layers with 32 and 8 neurons, respectively for further processed.

We do not explicitly describe the details of the *reward* function, but instead explain the rules used to achieve good RL agent results with the algorithm used:

- We encourage segments of average value in the detriment of very long or short segments.
- We gradually penalize very close angles (e.g., less than 30 degrees).
- We penalize new segments if they go out of bounds or self-intersect the existing road.
- We encourage road ends a little less than other elements, so that the road does not end so quickly.
- We encourage the creation of a new segment after a change in direction.
- We encourage the creation of a direction change after a new intersection.
- We penalize the creation of a similar element in a row.

Future work will also investigate whether the policy gradient class algorithms (Sutton and Barto, 2018) could provide better results in certain cases. In addition, further research needs to evaluate whether adding the image drawn at each time point can be efficiently used as input by convolutional layers. The results of our current evaluation are inconclusive in these cases.

5 EVALUATION

As mentioned in the Section 2, there was no similar work in terms of requirements, i.e., mimicking the real world or given user input with real-time usage) that could be compared to our presented work. Instead, in the following text, we make an evaluation of the proposed generative methods from different perspectives.

5.1 Qualitative Results Comparison

It is difficult to visually compare the results between the L-systems and the RL-based systems. However, from our observation, the RL-based method has a different degree of sampling for stochastic variables in theory, which translates into a more diverse road network in practice. However, the advantage of using L-systems, is that the user can constrain and control the generation process in deeper details, which could be an important decision in some fields, such as game development. To test whether the patterns of a given city can be reproduced and the models indeed generate similar content, we selected a subset of two cities, Paris (city center) and Chicago, from the original dataset and applied our method to test whether the inference leads to similar results. This overfitting test showed that both low-level methods can produce similar content to the original data without significant visual differences, while still showing the original roads patterns. However, for both methods, we constrained the environment and methods to produce self-overlapping results. Without this hard constraint, we obtain results similar to those in Figure 4.

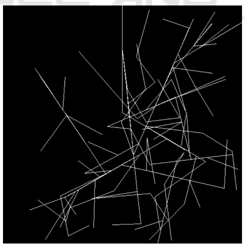

Figure 4: A typical result that might occur if you do not use the self-intersecting hard constraint.

Figure 5 shows a final result for creating a 3×3 grid structure from a portion of the Paris city map.

Note also that our generation process can work with inputs learned from user-defined datasets, regardless of the patterns they exhibit. Following the tile selection cluster method we introduced in Section 4.1, the user can switch patterns for individual clusters at runtime. In a computer game, for example, this can be used to switch between different parts of the worlds or to create custom mini-games. One limitation of our method is that we do not learn anything about the elevation of the ground at this point. This has been left as future work.

Figure 5: Network of roads on a 3×3 tiled environment generated using our methods based on a some specific part of Paris city map.

5.2 Runtime Performance

To find out if the method is suitable for real-time evaluation, we compared the generation of a different number of tiles at once (1 to 9) with both methods on an Intel Core i7 11700k CPU. Since nowadays parallelism is also heavily used in games or simulation engines, we evaluated both sequential and parallel computing time considering 8 worker threads, each assigned to a different CPU core (i.e., a parallel task for each tile). The results shown in Table 1 indicate that the method is suitable for real-time use, even when the visualization perspective is heavily modified, i.e. when many tiles need to be generated at once. Using L-Systems as the base method provides a slight advantage, at the cost of a possible lower variety of generated environments as noted above.

6 CONCLUSION AND FUTURE WORK

This paper presented first methods for generating a continuous road network in large environments that mimics the patterns of real-world cities. The evaluation section has shown that our proposed methods are generally suitable for use at runtime in applications such as computer games or simulation environments. Second, we presented a set of scripts that can transform a dataset of top-down pictures of real-world cities into graph augumented structures that the community can experiment with further, and made them available as open source. We plan to continue our work on procedural content generation, particularly in creating models capable of generating content around these roads. Possible examples could include vegetation, buildings, etc. We are also looking forward to expanding the augmented data structures used and in-

Table 1: Comparative runtime evaluation (milliseconds) between methods using reinforcement learning, L-Systems, both sequential and parallel with 8 worker threads.

Number of tiles	1	2	3	4	5	6	7	8	9
Sequential RL	2.03	4.43	7.88	10.4	12.81	15.79	18.21	21.41	23.42
Parallel RL	2.14	3.61	6.33	8.21	9.89	11.83	12.98	13.99	14.61
Sequential LS	1.76	3.98	7.72	10.29	10.63	13.73	18.02	20.33	22.48
Parallel LS	1.99	2.92	5.12	7.71	8.60	11.59	12.33	13.29	14.46

clude terrain elevation. In terms of methodology, we plan to explore other methods for generating similar or combined content.

REFERENCES

Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. (2018). Netgan: Generating graphs via random walks.

Chen, G., Esch, G., Wonka, P., Mueller, P., and Zhang, E. (2008). Interactive procedural street modeling. *ACM Trans. Graph.*, 27(3):Article 103: 1–10.

Curry, R. (2000). On the evolution of parametric l-systems.

Danny Alberto, E. C., Luo, X., Navarro Newball, A. A., Zúñiga, C., and Lozano-Garzón, C. (2019). Realistic behavior of virtual citizens through procedural animation. In *2019 International Conference on Virtual Reality and Visualization (ICVRV)*, pages 243–247.

de Araújo, L. J. P., Grichshenko, A., Pinheiro, R. L., Saraiva, R. D., and Gimaeva, S. (2020). Map generation and balance in the terra mystica board game using particle swarm and local search. In Tan, Y., Shi, Y., and Tuba, M., editors, *Advances in Swarm Intelligence*, pages 163–175, Cham. Springer International Publishing.

Dong, J., Liu, J., Yao, K., Chantler, M., Qi, L., Yu, H., and Jian, M. (2020). Survey of procedural methods for two-dimensional texture generation. *Sensors*, 20(4).

Freiknecht, J. and Effelsberg, W. (2017). A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*, 1(4).

Gisslén, L., Eakins, A., Gordillo, C., Bergdahl, J., and Tollmar, K. (2021). Adversarial reinforcement learning for procedural content generation.

Hartigan, J. A. and Wong, M. A. (1979). A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1):100–108.

Hasselt, H. (2010). Double q-learning. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

Heskes, T., Zoeter, O., and Wiegerinck, W. (2004). Approximate expectation maximization. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press.

Kipf, T. N. and Welling, M. (2016). Variational graph auto-encoders.

Lara-Cabrera, R., Cotta, C., and Fernández-Leiva, A. (2012). Procedural map generation for a rts game.

Li, Z., Wegner, J. D., and Lucchi, A. (2019). Topological map extraction from overhead images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Liu, J., Snodgrass, S., Khalifa, A., Risi, S., Yannakakis, G. N., and Togelius, J. (2020). Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1):19–37.

Mena, J. and Malpica, J. (2005). An automatic method for road extraction in rural and semi-urban areas starting from high resolution satellite imagery. *Pattern Recognition Letters*, 26(9):1201–1220.

Parish, Y. I. H. and Müller, P. (2001). Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 301–308, New York, NY, USA. Association for Computing Machinery.

Ping, K. and Dingli, L. (2020). Conditional convolutional generative adversarial networks based interactive procedural game map generation. In Arai, K., Kapoor, S., and Bhatia, R., editors, *Advances in Information and Communication*, pages 400–419, Cham. Springer International Publishing.

Reynolds, D. (2009). *Gaussian Mixture Models*, pages 659–663. Springer US, Boston, MA.

Rozenberg, G. and Salomaa, A. (1980). *Mathematical Theory of L Systems*. Academic Press, Inc., USA.

Snodgrass, S. and Ontañón, S. (2017). Learning to generate video game maps using markov models. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(4):410–422.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., Xie, X., and Guo, M. (2017). Graphgan: Graph representation learning with generative adversarial nets.

Zhou, D., Zheng, L., Xu, J., and He, J. (2019). Misc-gan: A multi-scale generative model for graphs. *Frontiers in Big Data*, 2:3.