Real time map generation using Deep Reinforcement Learning

-Overview-

Related Work

1. This approach [1] involves using a genetic algorithm for generating and evolving balanced maps (i.e., maps that give the same advantages / disadvantages for all players) for real-time strategy (abbr. RTS) games.

2. This approach [2] involves using “particle swarm optimization” (abbr. PSO) for generating balanced maps, fulfilling at the same time a given subset of requirements and Steepest ascent hill climbing with random restart (abbr. SAHC), a greedy metaheuristic less prone to getting stuck into local optimum than the classic hill climbing method

3. This approach [3] uses Markov Models in order to generate video games maps. The authors are trying to come with a generation method that is widely applicable, not just for a specific game. They are trying to achieve these using “Multi-dimensional Markov Chains” (abbr. MdMC), “Hierarchical Multi-dimensional Markov Chains” (abbr. HMdMC) and “Markov Random Fields” (abbr. MRF).

4. This approach [4] uses Tree Search algorithms such as Depth-First Search (abbr. DFS), Breadth-First Search (abbr. BFS), Greedy Best First Search (abbr. GBFS) and Monte Carlo Tree Search (abbr. MCTS) in order to generate 2D game maps.

5. This approach [5] makes use of Conditional Generative Adversarial Networks (abbr. cGAN, extension of simple GAN), Convolutional Neural Networks (abbr. CNN) and classical procedural methods.

6. The approach [6] consists of two Reinforcement Learning (abbr. RL) agents: the generator and the solver, that are dependent on each other. The generator creates an environment which is then tested by the solver. The generator receives feedback (rewards and observations) from the solver plus some auxiliary content which is set by the developers

Our approach

1. Dataset processing

We will make use of semantic representation of the CITY-OSM dataset. The dataset consists of 1671 aerial images, weighting almost 45GB, of Berlin, Chicago, Paris, Potsdam, Tokyo and Zurich cities. The raw images were segmented into a much 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]), not being of direct interest for our task. A series of computer vision transformations were applied in order to obtain a suitable dataset:

- Resizing
  The raw images are very big (i.e., 2611 x 2453 resolution => ~10 MB per image) and also, they don't have the same size. For this we will resize them to a convenient size. A resolution of 512 x 512 was chosen as it is small enough to work with but high enough to keep the important aspects of the image.

- Road extraction
  We will keep only the pixels that lay within a manually set color range (thresholding), obtaining a binary image where the road is marked with black pixels while the white pixels mark the background (non-interest area).

- Closing
  We will apply one iteration of closing morphological transform using a 3 x 3 rectangular kernel in order to get rid of the isolated pixels.

- Erosion
  We will apply two iterations of erosion morphological transform using a 3 x 3 rectangular kernel in order to fill in eventual white pixels from the road so the road is complete and well filled withing its edges.

- Skeletonize

  We will create a skeleton of the roads network where the width of the roads is almost equal. This is achieved using scikit image's skeletonize method. The method is an implementation of Zhang's algorithm [7] used for image content thinning.

- Binary dilation

  The final step consists of applying a binary dilation morphological transformation with a selim disk kernel, ensuring that the roads obtained at the previous steps won't have any discontinuities at pixel level.

2. Extracting relevant data

   We will transform the image into a graph as an abstraction to our task.

   - Nodes

     Determine the intersections of the roads network.

     Determine the road ends of the roads network.

     Determine the road turns of the roads network.

   - Edges

     Draw edges between determined nodes using the BFS algorithm on the image

3. Lindenmayer Systems (L-Systems)

   We define a randomized L-System in the following manner in order to generate roads networks:

   - After a straight road, we will generate a turn, an intersection or a road end
   - After an angle we force a new straight road (so we can observer how the road turns)
   - After an intersection, for each of its branches, we may generate either a straight road or a turn
   - After a road end we don't generate anything further

   The values of the elements are randomly chosen based on statistical data extracted regarding turning angles, segment lengths, number of appearances of each element..etc.

4. Reinforcement Learning

We train a Reinforcement Learning agent with the task of generating a graph that resembles a road network. We will use a Double Deep Q-Network (abbr. DDQN) powered agent, implementation of the "tensorflow agents" framework along with Adam optimizer and element wise squared loss as the loss function. the agent will have to choose at each step between creating a straight road segment of a certain length, creating an angle of a certain number of degrees, creating an intersection with a certain number of branches and creating a road end.

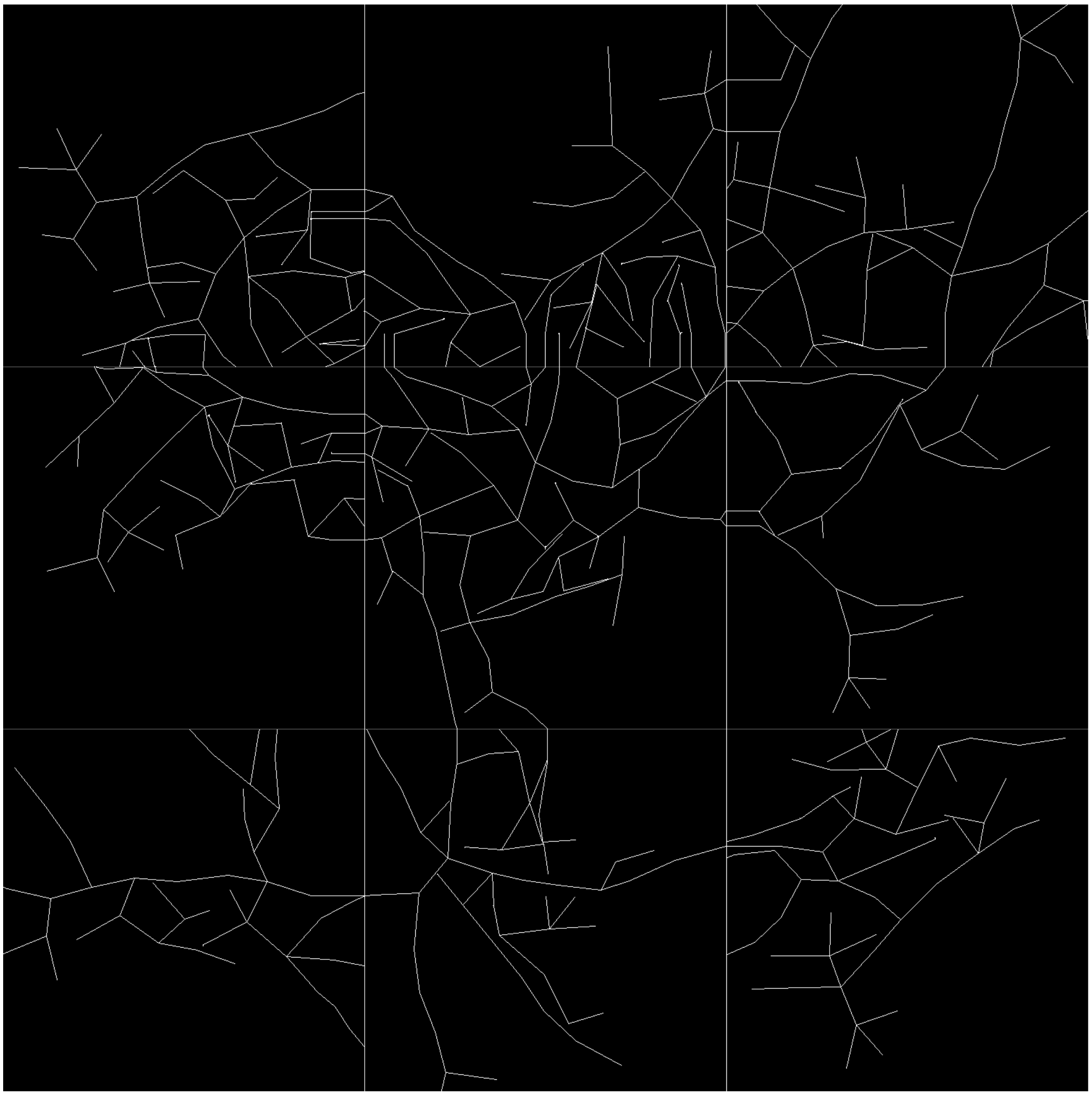
We interpreted our problem as an optimization task where we want to generate as many road segments as possible, without self-intersections and without getting out of the work space. For simplicity, we will consider fixed length segments. We separate the events into three kinds, this way defining the reward function:

- Positive events: these are wanted events, give a positive reward
- Weak negative events: these are not really wanted but they don't break anything, don't give any reward / neutral reward = 0
- Negative events: these events are process breakers, give hard penalty and stop execution

To prevent a continuous series of the same action (that yield positive or neutral reward) we also added a condition of frequency: if an action appears more than a fixed number of times in the past 20 actions, give a hard penalty to discourage abusing a certain action.

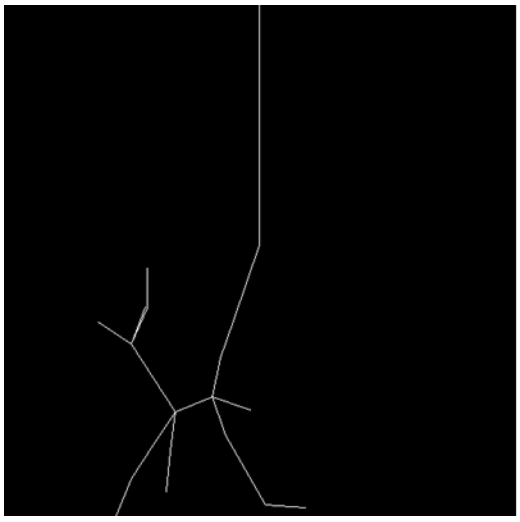
Results

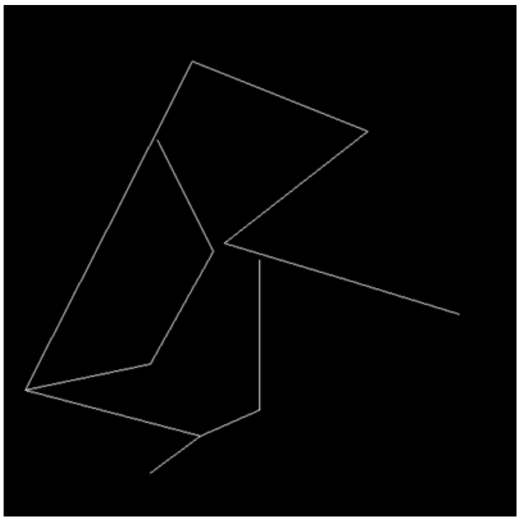
1. Lindenmayer Systems



The generated map may be realistic considering real life maps of cities.

2. Reinforcement Learning





The generated samples are very basic but the agent shows potential into generating more complex results.

References:

[1] Lara-Cabrera, Raul & Cotta, Carlos & Fernández-Leiva, Antonio. (2012). Procedural Map Generation for a RTS Game. 13th International Conference on Intelligent Games and Simulation, GAME-ON 2012.

[2] de Araújo L.J.P., Grichshenko A., Pinheiro R.L., Saraiva R.D., Gimaeva S. (2020) "Map Generation and Balance in the Terra Mystica Board Game Using Particle Swarm and Local Search". In: Tan Y., Shi Y., Tuba M. (eds) Advances in Swarm Intelligence. ICSI 2020

[3] S. Snodgrass and S. Ontañón, "Learning to Generate Video Game Maps Using Markov Models," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 9, no. 4, pp. 410-422, Dec. 2017

[4] Bhaumik, Debosmita & Khalifa, Ahmed & Green, Michael & Togelius, Julian. (2019). "Tree Search vs Optimization Approaches for Map Generation".

[5] Ping K., Dingli L. (2020) "Conditional Convolutional Generative Adversarial Networks Based Interactive Procedural Game Map Generation." In: Arai K., Kapoor S., Bhatia R. (eds) Advances in Information and Communication. FICC 2020.

[6] Linus Gisslén, Andy Eakins, Camilo Gordillo, Joakim Bergdahl, & Konrad Tollmar. (2021). "Adversarial Reinforcement Learning for Procedural Content Generation".

[7] T. Y. Zhang and C. Y. Suen. 1984. A fast parallel algorithm for thinning digital patterns. Commun. ACM 27, 3 (March 1984)