Design-Space Exploration
Map Generation

Chris Vesters
chris.vesters@student.ua.ac.be

Abstract

Design space exploration is a technique that will generate useful instances and makes the normal trial and error process obsolete. This saves a lot of time and increases possibilities of technology by finding new approaches. In this report we shall use this technique to automatically generate realistic road networks.

Keywords: Design-Space Exploration, Map Generation

1. Introduction

The essence of design-space exploration is generating the best solution or design. The process of trial and error takes time and costs money, this can be reduced or even eliminated by using design-space exploration.

The goal of this report is to show the strength of the technique, we do this in a little example which shall generate realistic map. Before starting with the project, we will first go into detail about the problems and solutions that arise when performing design-space exploration. This is covered in section 2 and is based on the work of others. In section 3 we will set out the goal and specifications of the example we will work out. Section 4 will implement a basic version of the example based on L-systems and in section 5 the complete example will be worked out based on graph transformations. work out the example to demonstrate the use of design space exploration. In section 6 we conclude on what is realised with this project and the possible extensions, improvements and applications.

2. Related Work

Design Space Exploration

In the report [3] the technique of design space exploration is explained. Design space exploration consists of several steps, first of all we need to generate a candidate. Depending on the way this candidate is found, it can be that the candidate is not of the 'type' that we want. Therefore we need to check the generated candidate for feasibility. If the candidate is feasible we need to evaluate it, only by doing so we have a means of 'best'.

There exist several algorithms for the exploration of the state space that can be used to generate a candidate. In the paper they distinguishes three types of algorithms:

- Exhaustive: given a certain instance, this algorithm will try all changes it can apply. The biggest problem of this algorithm is that the state space if often very large, leading to long computations before reaching a result.
- Random: randomly apply a change to the current instance. If we let this algorithm run several times we will get several solutions of which we can take the best. This will lead to solutions that are spread over the design space.
- Hill Climbing: candidates are generated by applying changes to the current instance. A newly created instance is only considered a candidate if it is better than the current instance. This means that we go on until further changes don't improve the current solution. The biggest problem of this is that we may end up in a local maximum.

Map Generation

Riry Pheng described in her reading report [8] some techniques that are used in the gaming industry to generate realistic textures, trees, ... very fast. The reason why these techniques are used is that the players want a realistic world. This can not be achieved if everything looks the same, some differences are required. We could manually create different maps, trees, textures, ... but that implies more work leading to longer development times and higher costs. Because this is not acceptable these techniques are used. The techniques covered in the report are:

- Fractals: a system in which an object consists of little copies of itself. The beauty about this technique is that we have infinite detail and we can determine the level of detail.

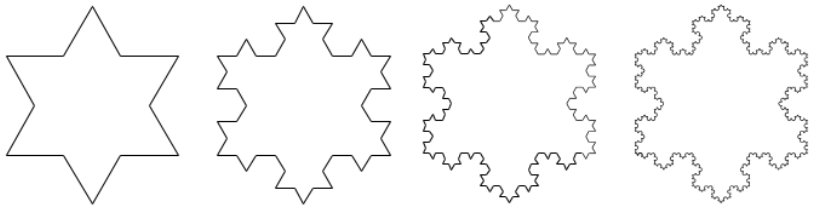


Figure 1: First four iterations of the Koch Snowflake

- L-Systems: a system in which we rewrite a string using a set of rules. Currently used to generate trees and plants.

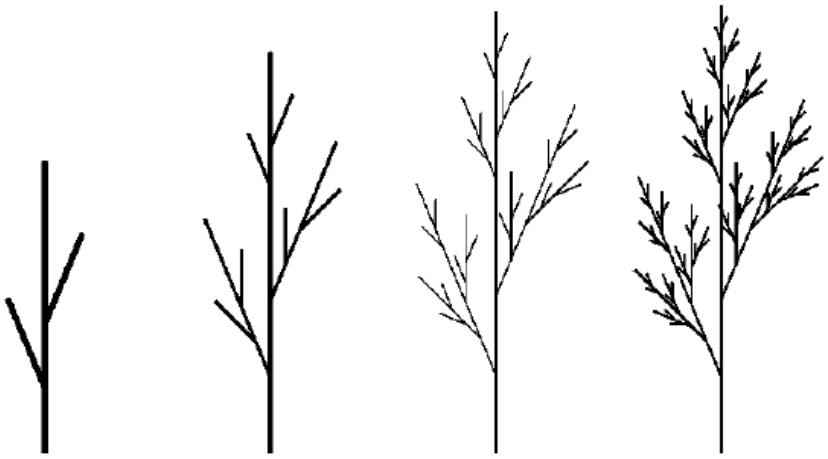


Figure 2: A plant generated based on a l-system

- Perlin Noise: A technique used to create more realistic textures. A random sequence of values is generated and combined into coherent noise by interpolation. By adding several layers of coherent noise we get a realistic texture.

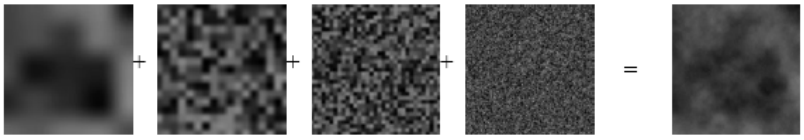


Figure 3: Four layers of noise added together to generate a realistic texture

- Tiling System: A technique used for textures. We have different tiles which can be placed together and even on each other. This is a fast way to create a vast variety of textures.

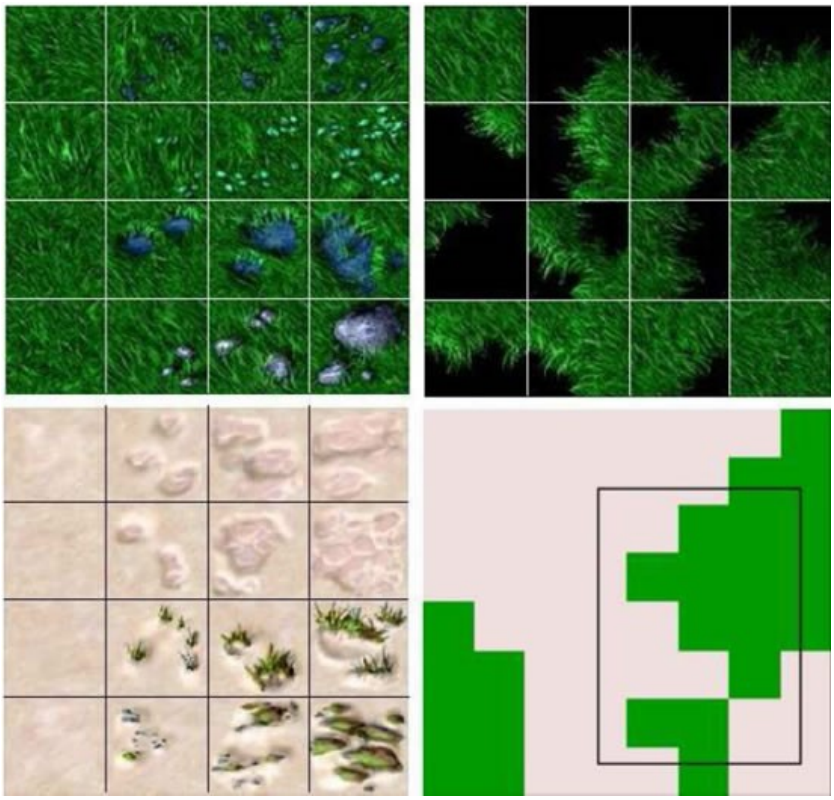
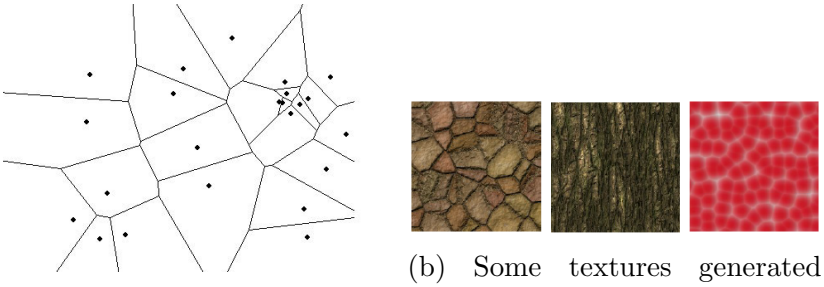


Figure 4: Tiling System

- Voronoi Texture Basis: A technique to generate cellular surfaces by using Voronoi Diagrams.



(a) Voronoi diagram

(b) Some textures generated based on a voronoi diagram

Figure 5: Voronoi Texture Basis

Not all these techniques can be used to generate realistic maps, most of them are meant for creating realistic textures. For this reason only the L-Systems shall be discussed in more detail since this is the technique used further on in this report.

Riry describes in her project report [7] how map generation is achieved. She uses rules to rewrite the map, these rules can be divided into 4 groups which shall be executed randomly:

- Network Growth: this will expand the current map.
- Local Expansion: adding roads to which houses can be connected.
- Local Population: creating houses and thus population.
- Road Fragmentation: breaking down large roads and adding extra specific roads.

An example of a map generated by Riry can be seen in figure 6.

Tough there are some differences between the example presented in this report and that of Riry, but overall we can use her work as a basis and even more as an example of what we can do.

Though it is not used as a basis for this report, the CityGen engine [6] is a very interesting program that is related to this report. The CityGen engine generates both the roads and the buildings by using L-Systems. This way it can create large, complex and diverse cities.

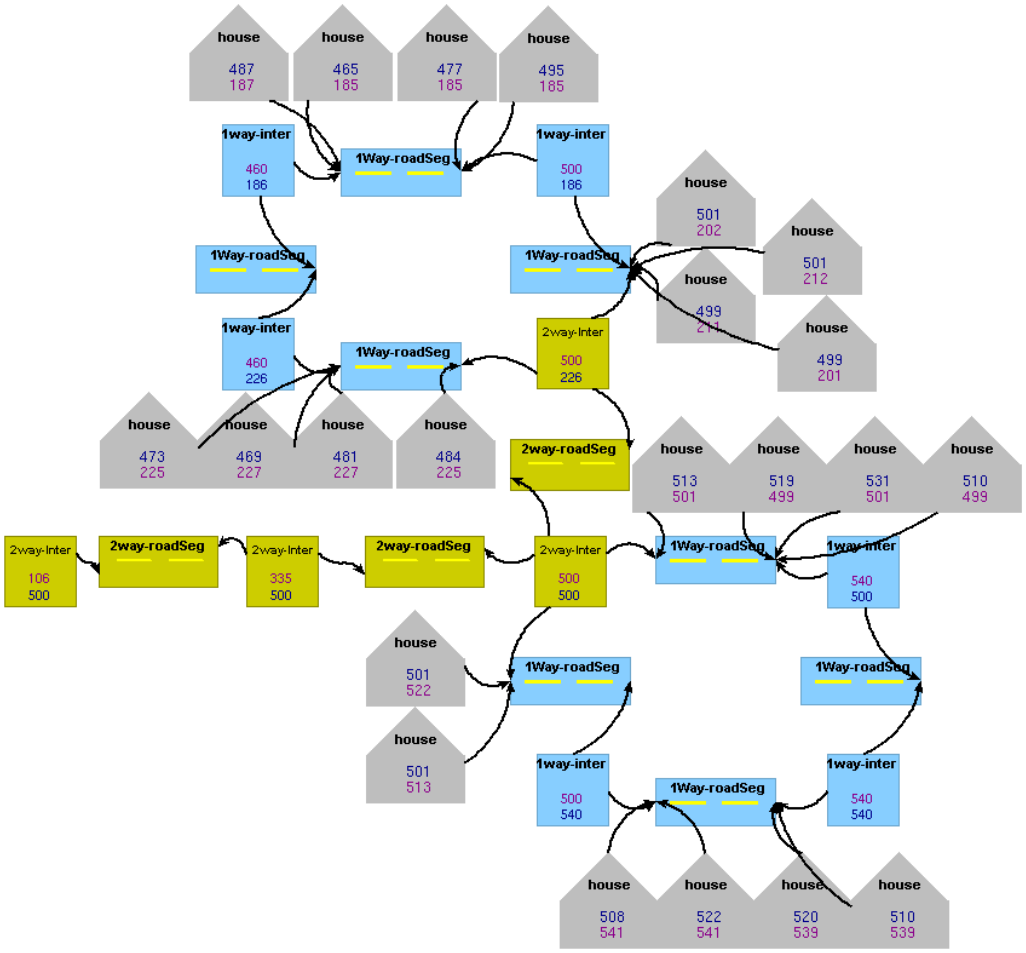


Figure 6: A map generated by Riry

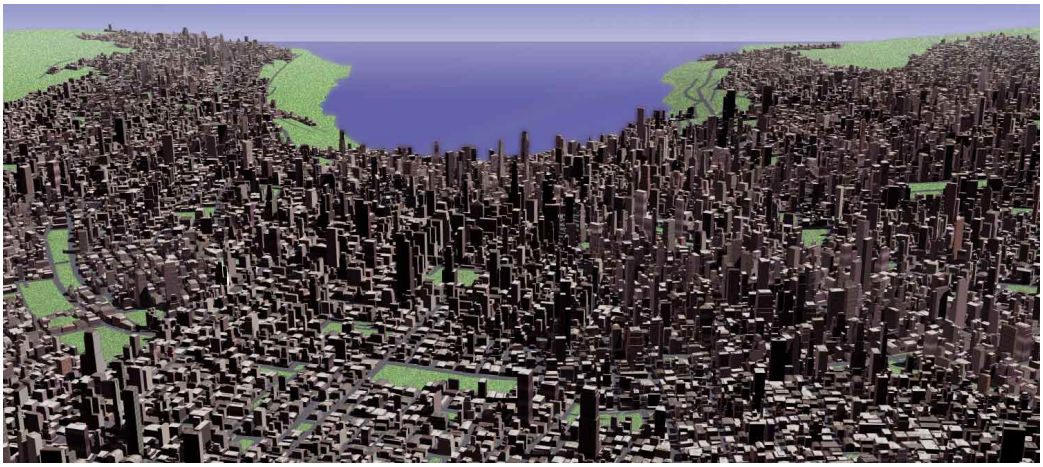


Figure 7: A virtual city consisting of approximately 26000 buildings

3. Project Introduction

The goal of this project is to show the power and usability of design space exploration. We do this by means of a small example, we will generate realistic maps. We shall cover two ways to do this: first we shall quickly show how L-Systems can be used, after which we go over to the more powerful system of graph transformation.

Before we can do anything we need to have some idea of the specifications of the map and the roads we will create. This is important and will contribute to the metrics to evaluate the map later. Some questions that need to be answered:

- Which style of roads will be generated? In this report we will limit us to a Manhattan map, which means that all roads are straight and only change in one of the two (x,y) direction.
- What is the width of the road? The width of the lane would normally depend on the type of road and allowed speed (for safety reasons). We will use a standard width of 3.5m for a lane [11]. This means that a road which consists of 2 lanes is 7m wide. Note that also the one-way road is this wide, the only difference between two-way and one-way road is that on a one-way road all cars travel in the same direction.

- What is the minimal length of a road? The minimal length of a road is set to the length of a car, the length of a car of course depends on the car itself. For simplicity the dimensions of one car as chosen, assuming that everyone driving on the map will have that car. The car we shall use is a Mercedes CLS Coupé. The dimensions of such a car can be seen in figure 8.

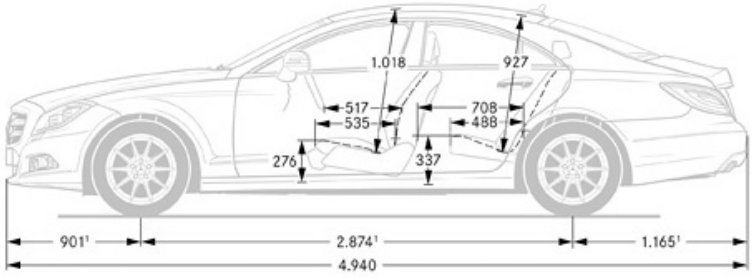


Figure 8: The dimensions of a Mercedes CLS Coupé

From this it follows that a road should be at least 5m long.

- What are the dimensions of an intersection? Because an intersection can have roads coming in from all four directions (North, East, South and West), an intersection must be 7m $\times$ 7m.

4. L-Systems

An L-System is a system in which we use rules to replace letters to create longer strings. It can be written as a tuple G = (V, ω, P) which consists of:

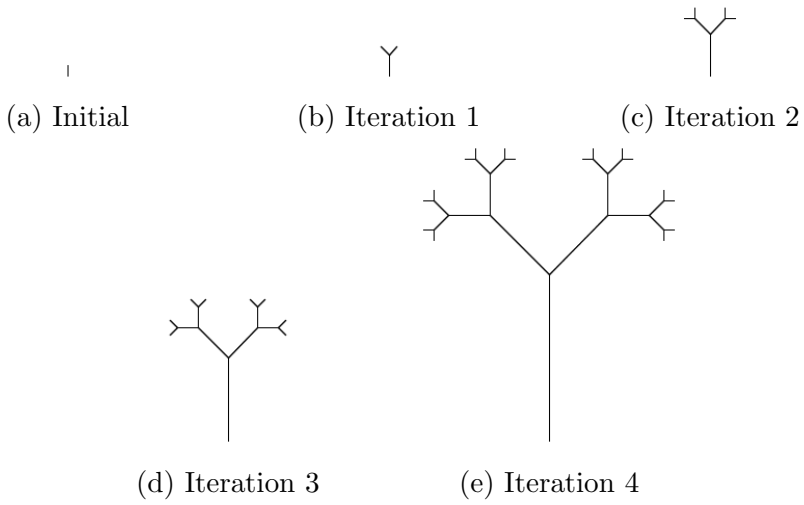
- An alphabet V
- An initial string ω
- A set of rules P

An example of a simple L-System which produces a simple plant.
V = 0, 1, [,], +, -
ω = 0
P = {(1 $\rightarrow$ 11), (0 $\rightarrow$ 1[-0]+0)}

The meaning of this is:

- 0: a line segment ending in a leaf
- 1: a line segment
- [: push the current state on the stack
-]: pop an element from the stack and return to it
- +: turn right 45°
- -: turn left 45°

This L-System generates the following results:



(a) Initial (b) Iteration 1 (c) Iteration 2

(d) Iteration 3 (e) Iteration 4

Figure 9: Iterative growth of L-System

L-Systems are used in games to create different plants and trees of the same family, which means they have to be similar but not exactly the same. This could be done manually but this would take to much time. To achieve

difference in multiple runs of an L-System we have to have some sort of randomness. This is achieved by assigning chances to each rule, this form of L-Systems are called Stochastic L-Systems.

If L-Systems can be used to generate trees and plants, they should also be used to generate maps. This can be seen in a very simple way, let us look at the previous generated plant, more precisely figure 9e. We can now interpret this as a map where the lines are roads. At the end and beginning of every line an intersection is created.

The biggest problem with this approach is that we can't differ between types of roads, which means we have to create a string with a larger alphabet.

$$V = \left\{ \begin{array}{ll} M: & \text{an entity which will let the map grow with a certain size} \\ I: & \text{an intersection} \\ T: & \text{a two-way road of a certain length} \\ O: & \text{a one-way road of a certain length} \\ [: & \text{store the current state} \\ ]: & \text{restore the previous state} \\ +: & \text{increase the angle with } 90^\circ \\ -: & \text{decrease the angle with } 90^\circ \end{array} \right.$$

$$\omega = I\left[+M\left(n\right)\right]\left[-M\left(n\right)\right]\left[++M\left(n\right)\right]M\left(n\right)$$

$$P = \left\{ \begin{array}{lcl} P_1: & M\left(n\right) & \rightarrow T\left(x\right)I\left[+M\left(\frac{n-x}{3}\right)\right]\left[-M\left(\frac{n-x}{3}\right)\right]M\left(\frac{n-x}{3}\right) \\ P_2: & M(0) & \rightarrow \epsilon \\ P_3: & T(x) & \rightarrow O(x) \end{array} \right\}$$

Note that in the rule P_1 x is a random generated number between 5 and n. The explicit mentioning of intersections is not required, but is done for clarity.

The maps generated with this L-System are very basic and certain constraints were omitted. There is for instance no randomness and thus it will generate the same map every time. This can be overcome by using Stochastic L-Systems.

This L-System is just meant to show the possibilities and limitations of L-Systems. One of the biggest limitations is that the generated map is not easy to analyse. If we wish to analyse them we first have to convert the string into another structure which allows easier analysis, probably a graph. So instead of always converting the string into a graph, it is more efficient to

expand the graph and save all the conversion work.

5. Graph Transformation

We will use a graph representation for the road network and rule transformations to develop it. To achieve this multiple libraries and programs are used:

- Atom3 [2]: used to describe the rules in. We use an adapted version [10] which supports the Himesis compiler.
- T-core [9]: the tool to find matches and rewrite the graph.
- iGraph [5]: T-core is developed on top of iGraph, but it is also used to simplify the graphs.
- NetworkX [4]: used to analyse the network.
- PyDevs [1]: python DEVS simulator, used to simulate the generated road networks.

Meta Model

The meta model to specify the road network is a very basic one (see figure 10). It consists of 3 classes which don't need any further explanation, but there are some few remarks:

- The Road class can be eliminated since it only contains duplicated information. The length of the road can be calculated based on its two bounding intersections. In the eliminated version there would be self-loops at the Intersections. The reason the Road class is present is to ensure a basic version which can be easily modified and extended. Some extensions will be discussed at the end of this report.
- The MapInformation class specifies the size of the map for which we generate a road network and the amount of resources we may use. The limitation is necessary to prevent creating maps that fill the entire map.
- There is no distinction between a one-way road and a two-way road, this distinction is made by the edges coming in and out. A two-way road has two incoming and two outgoing edges, a one-way road has only one incoming and one outgoing edge.

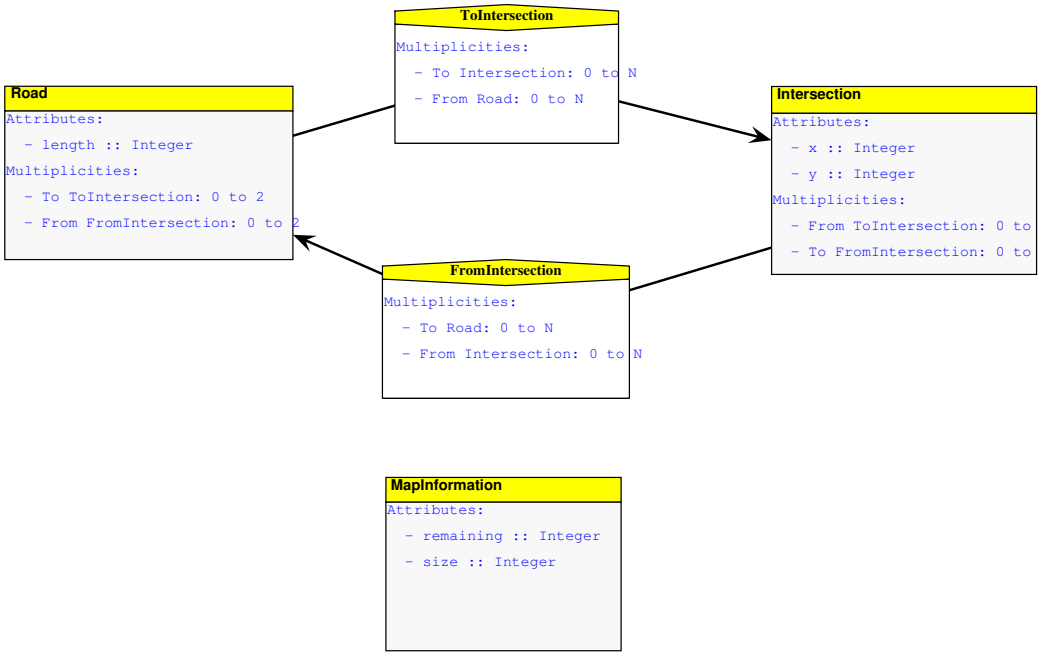


Figure 10: The meta model of the road network.

The meta model is kept simple on purpose, this way it become easier to adapt and expand.

Rules

There are seven rules to modify the map: ExpandNorth, ExpandEast, ExpandSouth, ExpandWest, GrowRoad, ConnectIntersections and OneWay. Each of these rules will now be discussed in more detail.

ExpandNorth, ExpandEast, ExpandSouth and ExpandWest are all alike. The only difference is the direction in which the road is created. These rules will create a new road in the specified direction with a length of 5m and a new intersection (at the end) from an existing intersection. The visual representation of this rule can be seen in figure 11. Because there is no visual difference in the rules, only one is presented. The differences are encapsulated in the checks, which shall not be discussed here. We do shortly mention the reason for the NACs:

- There doesn't exist an intersection that lies in the way.

- There doesn't exist a road which lies in the way (this road will be orthogonal to the newly created road because roads that are parallel will also have colliding intersections).
- There may not be a road starting in the intersection that goes in the same direction. This extra check is required because if the existing road is large enough the new road won't collide with the intersection.

We note that the direction of the roads are split, this is to cover one-way roads as well. This pattern will repeat itself in the next rules as well.

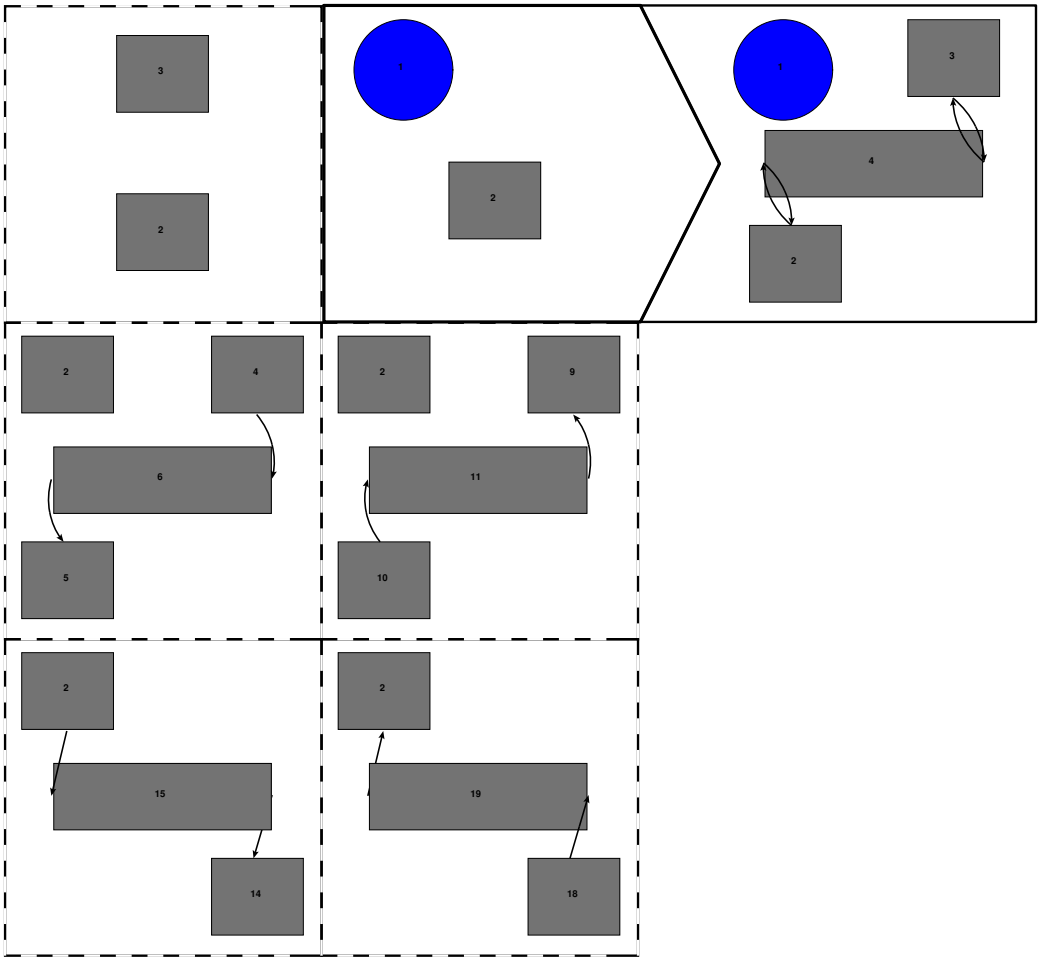


Figure 11: The Expand rule.

The GrowRoad rule will extend an existing road and move an intersection by 1m. The visual representation of this rule can be seen in figure 12. The NACs check for several situations:

1. There isn't an intersection in the way.
2. There isn't a road in the way.
3. The intersection we are moving isn't connected to another road. This is required to preserve a Manhattan road network.

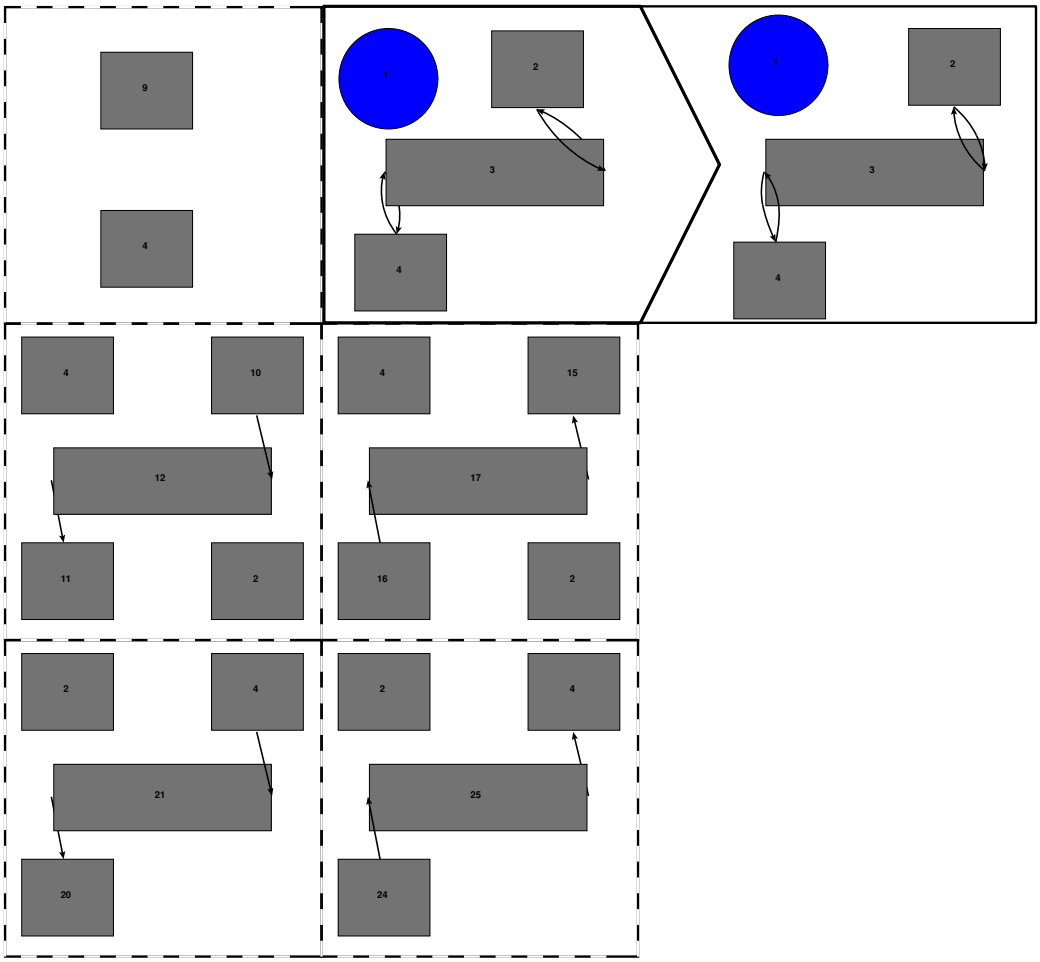


Figure 12: The GrowRoad rule.

The ConnectIntersections rule will connect two intersections with a road. If we would only extend the network from a staring intersection, the entire

network would be a tree. This means that there is only one route from one point to another. The visual representation of this rule can be seen in figure 13. The NACs check that there isn't an intersection or another road lying between the intersections.

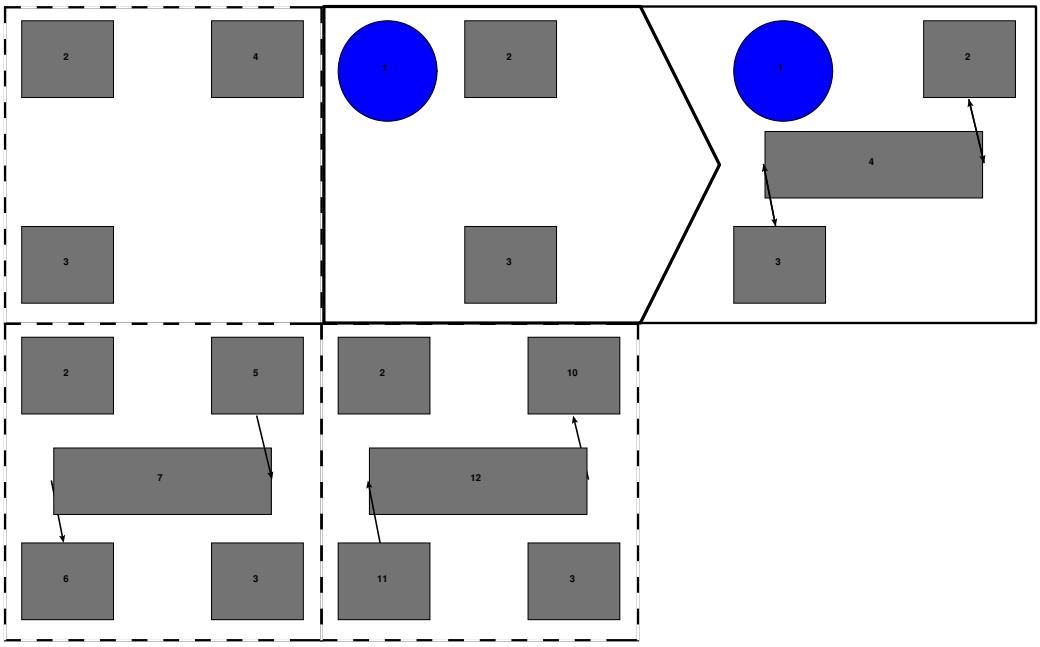


Figure 13: The ConnectIntersections rule.

The OneWay rule will convert a two-way road into a one-way road. This is a very simple rule, which can be seen in figure 14. We only need to specify one-way roads in one direction, because there is no constraint on the intersections they can be switched which results in a one-way road in the other direction.

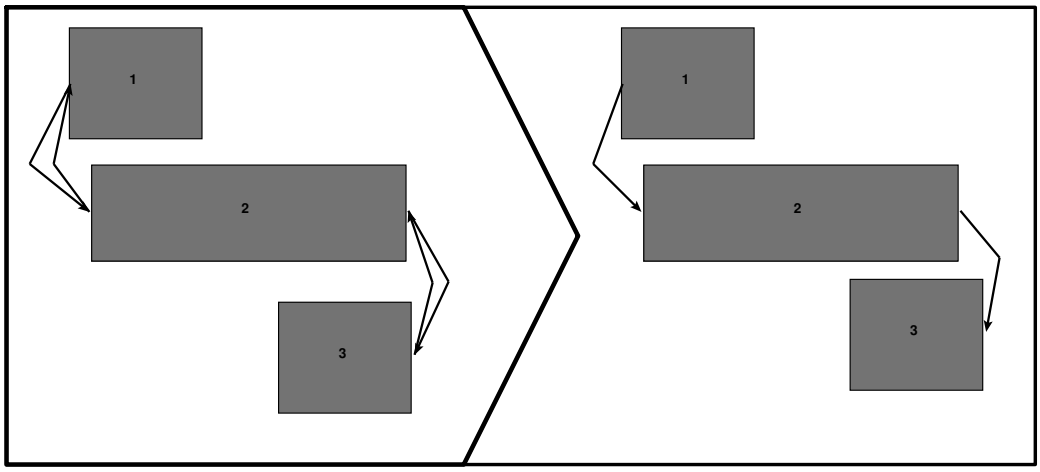


Figure 14: The OneWay rule.

Feasibility And Evaluation

Candidates can now be generated by applying one of these rules to a map. Generating candidates is only the first step of design space exploration. Feasibility is the next. The rules are created in such a way that mostly all of the candidates are feasible. It is for example not possible that roads intersect without an intersection, the rules check prevent this. The only thing that can happen is that we changed a two-way road into a one-way road which now results in an intersection that can't be reached any more. We therefore check that the graph is one strongly connected component, only if this is the case the generated instance is considered a candidate.

The final step to be able to begin design space exploration is evaluation, for this we need some kind of metric to express how good a certain map is. As in many cases there are multiple aspects which should be realized:

- The distance over the road from point A to B should be as close as possible to the Manhattan distance. Because we use Manhattan roads, this is the best we can do. We also want to minimize the amount of intersections we pass by, certainly if there are more than 2 incoming roads, because this means that we will have to slow down or even stop.

- The road network should cover as much of the map as possible.

- The road network should be evenly distributed over the map. This may not always be wanted, just imagine a city center were all roads come together. In this case it is acceptable that there are more roads in the center if the traffic requires this.

- The amount of routes that lead to a certain point. It would be great to be able to distribute the traffic that goes to one point over multiple routes. Not only is this good to reduce traffic jams, but also to have an alternative route if a certain road is closed.

The best way to verify these aspects is to simulate the traffic that uses the roads. This is however not possible because there are no residents, only if there existed some sort of houses (car generator) and offices, stores, etc (car collectors) this become possible. Because these are not provided in the example we can't use this. We therefore have to fall back on static evaluation of the map.

The metric used to evaluate the maps are based on two criteria:

- The longest path from one intersection to another: we want this length to be as close as possible to two times the size of the map.

- The average minimal cut of the graph: for every node we calculate the minimal cut to every other node. From this we take the average for that node. We do this for every node and calculate the overall average. This will create multiple routes to a node.

The first criteria will generate a map with a best longest path, it doesn't care about reachability. The second one on the other hand does, though it does not state that the distance should be close to the optimal Manhattan distance. This metric is not perfect but it is sufficient for this example.

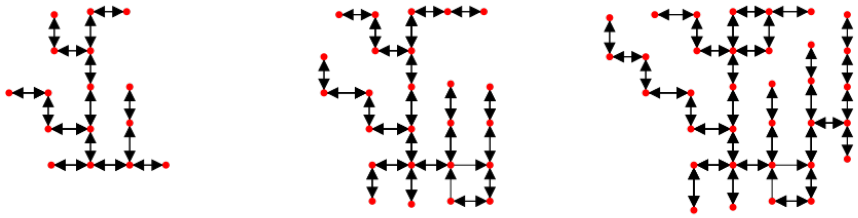
Running The Example

Now that we have all the different steps, we just need to put them together. This is done by choosing the wanted state space exploration algorithm. Since it is just a small experiment we want to have some good results in a reasonable time. None of the three algorithms mentioned before can guarantee this, so we use a self-designed algorithm.

The algorithm will generate all children of a candidate. Each child will be evaluated and checked for feasibility. If the child is not feasible or the score is not greater than it's parent's score, the child is rejected. If the child passes the evaluation it is added as a candidate. The next candidate to be extended is the one with the highest score. This will result in some sort of Depth First Search algorithm since the children will always score better than the parent, and thus also better than the parent's siblings. When a candidate has no children which are feasible and better, we have reached a local maximum. This modified version allows us to generate decent road networks without having to examine the entire graph. Though there is no guarantee that the reached maximum is the best one to be discovered, to find the best local maximum we need to examine the entire graph. This algorithm does not guarantee that the optimal solution will be found due to the elimination of children.

All is done now, and it is time to let the example run and generate some road networks. Shown below are a couple of road networks that are iteratively generated with the algorithm mentioned above. The time needed to search the entire state space is to long, which means that the program was aborted before it could find the best local maximum. The program was even aborted before it could find a first local maximum, yet the results are acceptable and sufficient as an example.

(a) The initial map (b) Iteration 1 (c) Iteration 10



(d) Iteration 50 (e) Iteration 100 (f) Iteration 150

Export To PyDEVS

To prove that the generated candidates can be used for simulation an exporter to PyDEVS is provided. The currently generated road networks can't immediately be simulated because of the absence of cars. They way the PyDEVS file works is simple. Cars travel from intersection to intersection until they reach their endpoint. There can only be one car at a certain intersection, this is a naive way to prevent crashes. A car will adjust it's speed to the maximum speed it's allowed on the road immediately after entering it. Some time before leaving the road the car will slow down to 30km/h, when this speed has been reached a query will be send to the intersection to ask for permission to enter. If the intersection is free, a reply will be sent indicating the car can enter immediately. At that time the intersection becomes reserved. If the intersection is reserved, the car must wait until further notice. If the intersection is occupied, but not reserved yet it will reply with the time left before the car leaves and the intersection becomes reserved.

The simulator is in many aspects a bit simple, and can certainly be improved. It does show that we can simulate traffic for the generated roads.

6. Conclusion And Future Work

Though the experiment used in this report is very small scaled, we can nevertheless conclude from the results that design space exploration is a good technique to explore a state space and find useful instances.

Though the small example can indicate the usefulness of design space exploration, itself it's not that useful. There are many extensions that can be added, First of all adding houses, stores, offices, etc will allow the road network to be simulated. This will give us more metrics leading to better and more realistic road networks. Other extensions include among speed limits, amount of lanes on a road and traffic lights. This will all contribute to more realistic maps. These can only be used in cases where we can start from nothing such as games.

Therefore another possibility for the future is to fill the initial map with houses, stores, offices, etc and find the best road network to connect all these buildings. This could be used to completely redesign existing cities and reduce traffic problems.

References

[1] Bolduc, J.-S., Vangheluwe, H., Van Tendeloo, Y., 2000 - 2013.
URL `http://msdl.cs.mcgill.ca/projects/projects/DEVS/`

[2] de Lara, J. e. a., 2002 - 2013.
URL `http://msdl.cs.mcgill.ca/projects/projects/AToM3/`

[3] Denil, J., Han, G., Persson, M., Liu, X., Zeng, H., Vangheluwe, H., 2013. Model-driven engineering approaches to design space exploration. Tech. rep.

[4] Hagberg, A., Schult, D., Swart, P., 2004 - 2013.
URL `http://networkx.github.com/`

[5] igraph Project, T., 2006 - 2013.
URL `http://igraph.sourceforge.net/`

[6] Parish, Y. I. H., Müller, P., 2001. Procedural modeling of cities.

[7] Pheng, R., 2008. Procedural modeling for city map generation - final report.

[8] Pheng, R., 2008. Procedural modeling for city map generation - reading report.

[9] Syriani, E., Hans, V., 2009 - 2013.
URL http://syriani.cs.ua.edu/Research.aspx

[10] Syriani, E. e. a., 2009 - 2013.
URL http://syriani.cs.ua.edu/Research.aspx

[11] Wikipedia, .
URL http://en.wikipedia.org/wiki/Lane