Procedural Modeling for City Map Generation - Final Report

Riry Pheng

McGill University, Montreal, Quebec, Canada H3A 2T5

Abstract. Modeling a city poses a number of problems in the computer graphic area. Every urban area has a transportation network that follows population and environmental influences, and often a superimposed pattern plan. To create a virtual city, a roadmap has to be designed and a large number of buildings need to be generated. Although many procedural techniques exist to solve this problem, we now propose a new procedural modeling approach based on meta-modeling formalism, graph-grammar and graph transformations. For the creation of a city street map, rules have been designed to take into consideration the city designers global goals and local constraints.

1 Introduction

Modeling and visualization of man-made systems such as large cities is a great challenge for computer graphics. Cities are very complex to model realistically as they reflect historical, cultural, economic and social changes over time in every aspect in which they are seen. As computing power increases, the modeling and visualization of large-area cities has become feasible. Several procedural techniques have been designed for the quick creation of complex environment, such as fractals, L-systems, Perlin Noise, tiling, and Voronoi texture basis [1].

In this project report, we propose the use of a procedural modeling approach to generate city road maps. We present a meta-modeling formalism called Road-Generator which is capable of modeling a complete city road map using a small set of input data. This report comprises four sections. At first, we discuss the design motivations. We then describe the meta-modeling formalism, followed by the explanation of the transformation rules and the design constraints. Furthermore, some experimental results are presented, followed by a conclusion.

2 Procedural Modeling

Similar to the L-system approach, which uses a formal grammar and a set of rewriting rules, we suggest the use of graph-transformation to achieve the same result. In the example of a Thue-Morse system shown in fig. 1 a set of rewriting rules is defined along with a set of input parameters. Using our approach, a set of graph-grammar rules is created and applied iteratively to an input graph.

$V = \{a, b\}$	ω :	a
$\omega = a$	$n=1$:	ab
$P_1 : a \rightarrow ab$	$n=2$:	$abba$
$P_2 : b \rightarrow ba$	$n=3$:	$abbabaab$

Fig. 1. Thue-Morse system

3 Meta-Model: RoadGenerator

The RoadGenerator meta-model consists of 5 components (see fig. 2):

- RoadSegment1Way, RoadSegment2Way: representing one way and two way streets respectively
- Intersection1Way, Intersection2Way: representing one way and two way intersections respectively
- House: depicts built houses on the map

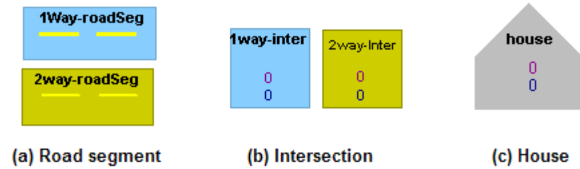

Fig. 2. Components representation

Fig. 3 depicts the connectivity between each components. We can see that only Intersection components and Houses have coordinate attributes. This is due to the fact that RoadSegment length can be easily calculated from the position of those 3 elements. Moreover, having a RoadSegment element provides a mean to visually represent the connectivity between two intersections. Thus, including coordinate attributes to RoadSegments is unnecessary and redundant. Every Intersection can connect to 4 different RoadSegments. However, a RoadSegment can only be connected with 2 Intersections. Although many houses can be attached to a RoadSegment, each House component can only be connected to one RoadSegment. Please note that the arrow direction in represented in the class diagram does not indicate the circulation flow in the road network. It only shows the connection between two elements.

All RoadSegment elements can only be connected to an Intersection component. Hence, they cannot be directly connected to one another, since there is not any attribute defining their position or their length, there is not any motivation to do this. We graphically represented 1-way and 2-way RoadSegments

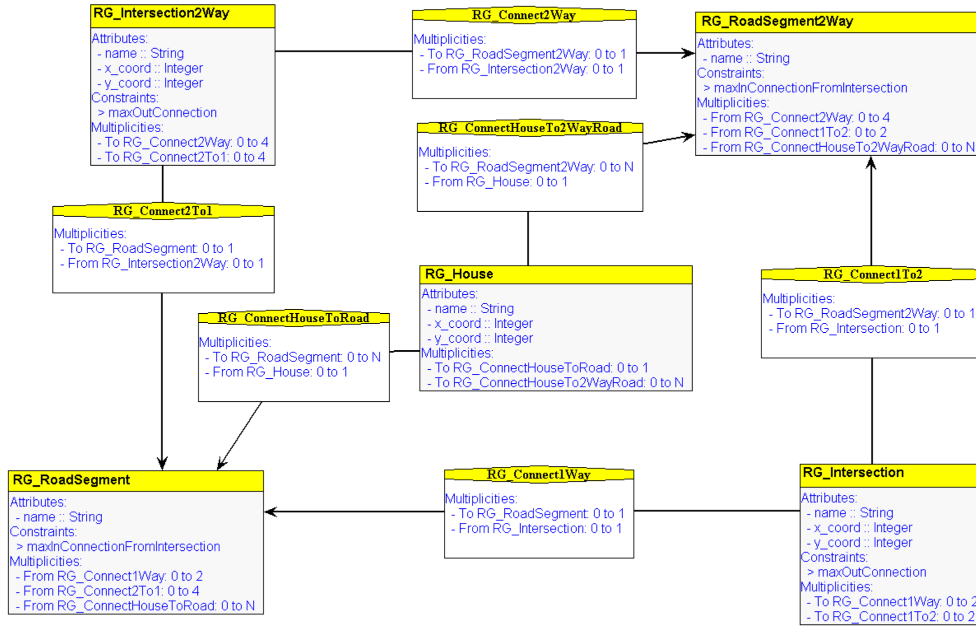

Fig. 3. Class diagram showing the 5 different components and their connections

differently because we want to be able to visually distinguish between them easily. House components can only be connected to RoadSegments. Their position determines where the house can be found on that particular road stretch.

4 Design and Parameters setting

In our approach, we use a grid to position the components according to their coordinates specified by the user. For design simplicity, an element is only allowed to be connected with an element contained in its top, bottom, left and/or right neighboring cells. No diagonal neighbor connections are valid. This idea is depicted in fig. 4 and 5

With the intention of providing some flexibility to the user when designing road networks, a set of input variables can be modified, this includes the grid map boundary, the mininum amount of space needed between two 2-way RoadSegments, the length of a dead end street, the minimum space between a 1-way and a 2-way RoadSegment, the size of a house, the minimum space between two houses or the house and a RoadSegment, the maximum length of a road stretch, and the maximum length and width of a 1-way street. The usefulness of these parameters will be explained in details in the next section.

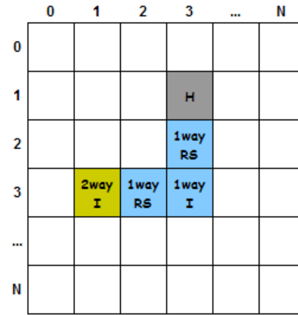

Fig. 4. Underlying grid to detect elements in map

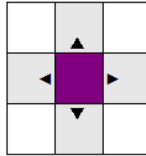

Fig. 5. 4 valid connections available: top, bottom, left, right

5 Transformation Rules and Design Constraints

Four sets of rules have been created in order to represent the different stages in city growth: road network growth, local expension, local population, and road framentations. All rules are assigned the same priority because we want to add randomness in our city map generation.

5.1 Road Network Growth

In this stage, long stretch of 2-way RoadSegments and Intersections are set up. Starting with a 2-way Intersection as input graph, we extend from it either at its top, bottom, left or right. Hence four rules have been created to do this: expend2wayUp, expend2wayDown, expend2wayLeft, expend2wayRight. Fig. 6 depicts one of the mentioned rules. Whenever a matching graph is found, we first verify that the 2-way intersection found still possess a free neighboring cell in our grid. This ensures that no more than 4 road segments are connected to that intersection. Also, this allows us to know in which direction it can be extended.

The position of the newly added 2-way Intersection is randomly determined. However, we must take into consideration the boundary of the map as well as the presence of existing road segments, such that new road segments do not overlap on old ones. Existing road segements' presence is detected by performing a search in our underlying grid map. The general idea consists of starting with a given intersection as our starting point and a desired direction to extend to.

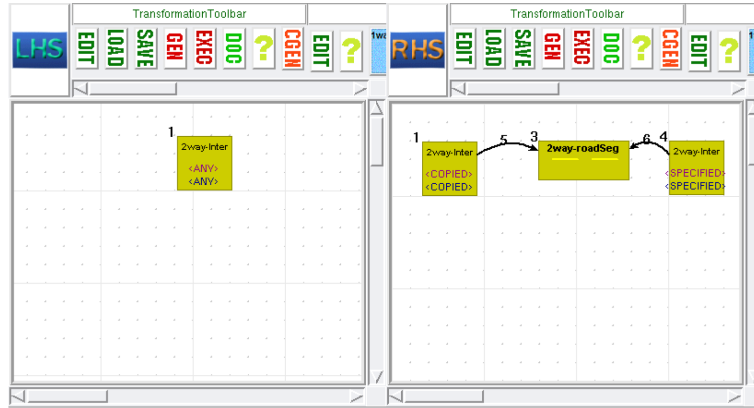

Fig. 6. expend2way-Up/Down/Left/Right transformation rule

We then loop in the grid map looking for an existing element until the map boundary is reached. If no element is found, then the length of the road segment will be randomly chosen between the our starting point and the map boundary. However, if an existing element is found, the length of the road segment will be bounded by our starting point position and the found element's position.

We can describe this more accurately with the use of an example. Assuming that we have a 2-way intersection element at position x = 500, y = 400 in the map. We now want to grow eastward (toward its right). We must first perform a search looking for any existing elements from x=500 to x=boundary value, and y = 400. If no element is found, the new intersection x coordinate could be any value between 500+min spacing and boundary-min spacing. Otherwise, its new x value would be a random value between 500+min spacing and existing element x coordinate min spacing. Note that min spacing here represents the minimum amount of space required between 2-way RoadSegments.

5.2 Local expension

At this stage, we assume that some road networks are already present in our map and we are trying to expend it very locally by creating dead end streets. Two rules have been created for this: expend1wayCulDSacLeft and expend1way-CulDeSacRight. One of these transformation rules is shown in fig. 7. Each rule creates a new road segment either to the left or to the right. Once a direction has been picked, the rule evaluates the situation and decides to expend either upward or downward.

Dead end streets are road stretch created using 1-way RoadSegments and Intersections. When designing with 1-way elements, we must be careful not to create any roads where once an individual enters, there is not a way to exit. Hence, a cycle like pattern should be created. Dead end streets are a natural representation of this cycle pattern, which forces a direction while ensuring that

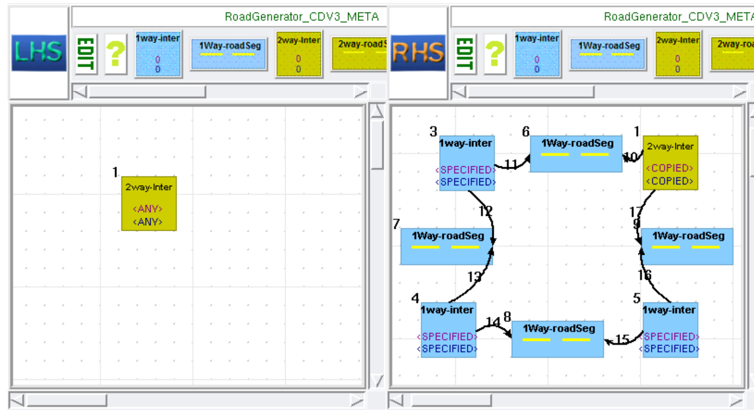

Fig. 7. expend1wayCulDeSac-Left/Right transformation rule

nobody will be stuck after entering that section. These ideas are illustrated in fig. 8.

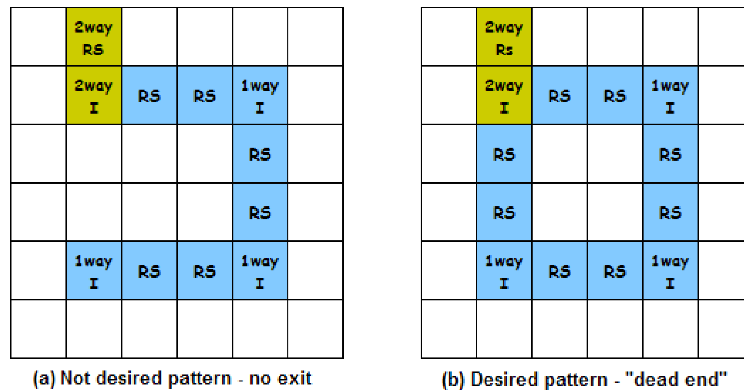

Fig. 8. Undesired and desired patterns

The length and width of dead ends are determined by the input parameters specified by the user. Also, when looking for a possible intersection to grow from, we must verify that there is enough space to create a dead end there. Once more, we search for existing elements in the map and determine if enough space is available, using the specified minimum number of space required between 1-way RoadSegments and existing 2-way RoadSegments in the map. The drawback of having a fixed number for the width and length of dead ends, all dead ends will look alike. However, this could easily be modified to support random numbers and thus making creating a more hectic looking map.

5.3 Local Population

Like in any city formations, new roads are only constructed if there are people to use it. Therefore, this step consists of building houses and populating the city. Eight transformation rules have been created to do this, although their graph transformation pattern is almost identical (see fig. 9), their underlying constraints are different.

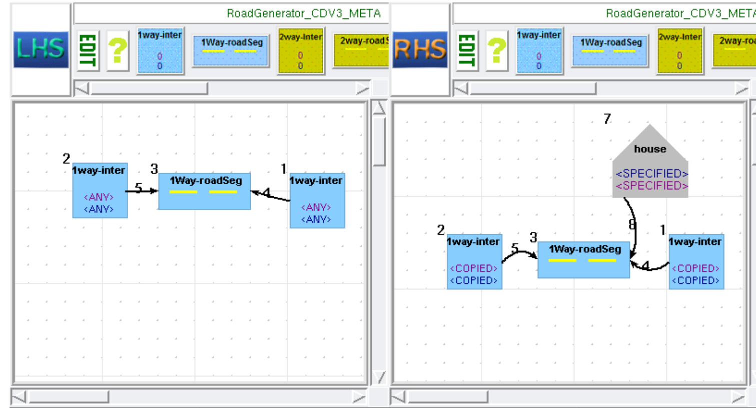

Fig. 9. addHouse transformation rule

Two intersections are needed to perfom the match in this case, because RoadSegments do not have any coordinates. In order to be able to calculate the position of the new house, we need to know the length of that road segment and see if there are not any other houses that may already be there. Moreover, knowing the size of a house, we verify that enough space is left between each houses or between the new house and an existing road segment. These values are to be specified by the user.

5.4 Road Fragmentations

In this last phase, we are trying to refine the existing map by shortening some roads that may be too long and by adding more 1-way cycles not following the dead-end pattern discussed in the Local Expension phase. Three transformation rules have been designed for this: splitHorizontal, splitVertical, createHorizontalCycle1way.

The first two rules consist of looking for a long road segment. If its length is larger than the specified maximum length of a 2-way road segment, then a new 2-way Intersection is added on that road. This is illustrated in fig. 10. On the other hand, createHorizontalCycle1way reside in adding a 1-way road segment parallel to an exisiting 2-way road, and connecting them together. To ensure not to have any extremely long 1-way roads, we use the maximum length and width

of a 1-way street parameters value as set by the user. The graph transformation rule for createHorizontalCycle1Way is shown in fig. 11.

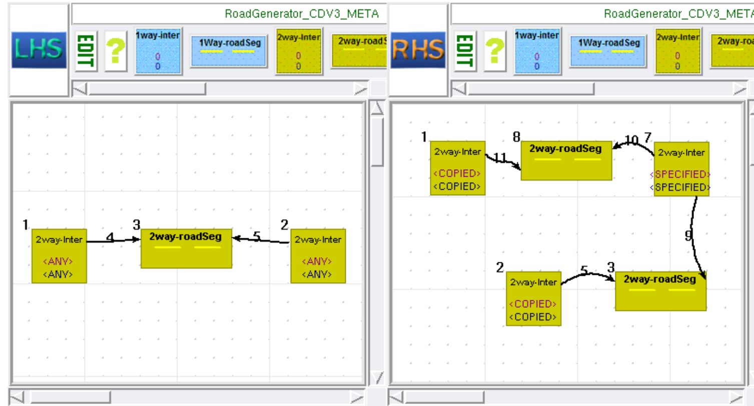

Fig. 10. splitVertical transformation rule

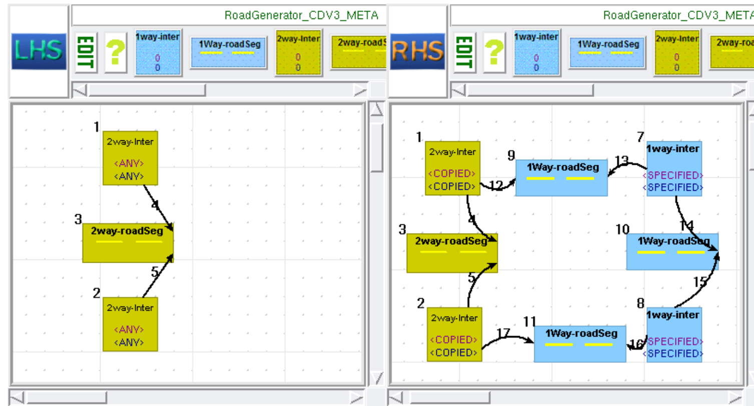

Fig. 11. createHorizontalCycle1Way transformation rule

6 Simple Example

Assuming that we always use a 2-way Intersection as our starting point, we first start by growing the city. A *Network Growth* graph transformation rule is randomly picked among the four existing rules: expend2wayUp, expend2wayDown, expend2wayLeft, expend2wayRight. After executing this for 3 steps, we get the

result graph shown in fig. 12. We now have four 2-way Intersections and three 2-way RoadSegments.

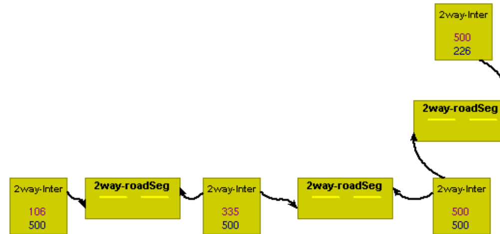

Fig. 12. Phase I: Road Network Growth rules execution results

It is now time to create smaller roads in order to build houses. We then executed rules from *Local Expansion* for 2 steps in order to get the graph in fig. 13. Two dead end cycles are created. One has been the result of executing the expend1wayCulDeSacLeft (upward) and the other, expend1wayCulDeSacRight (downward).

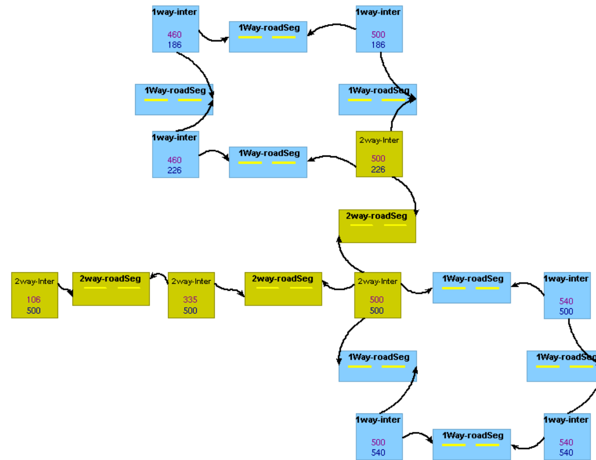

Fig. 13. Phase II: Local Expansion rules execution results

In this next step, we populate the map by executing rules from *Local Population*. House elements are then randomly attached to 1-way RoadSegments. Note that each house can only be connected to one road segment and every house is unique; unique name and coordinate. It would not be realistic to have 2 houses

at different position having the same name (or address in this case), nor would it be valid to have 2 distinct houses with the same coordinate. The resulting graph after executing this step illustrated in fig 14.

Fig. 14. Phase III: Local Population rules execution results

Once houses have been built, we need to refine the city map by fragmenting long roads into shorter segments. To do this, rules from *Road Fragmentations* are called and executed. In fig. 15, splitHorizontal has been invoked in order to get the resulting transformed graph. Only one rule is executed in this step because no other match has been found afterward.

Now that we have carried out all the four phases in city road network generation, we loop back to the first phase and repeat the process again. This iterative operation is done several times until the map is completely filled or until the city designer (user) decides that it is sufficient. A more advance map generation example in which several rules from each phases previously described are applied iteratively is shown in fig. 16.

7 Conclusion

When using this approach, we can easily create city road network maps quickly. With more detailed rules, we could generate more realistic maps as well. However, in order to create more realistic maps, more input variables need to be defined by the user. This method always guarantees the generation of distinct maps. Since the resulting map depends on the intersections positions, the number of time

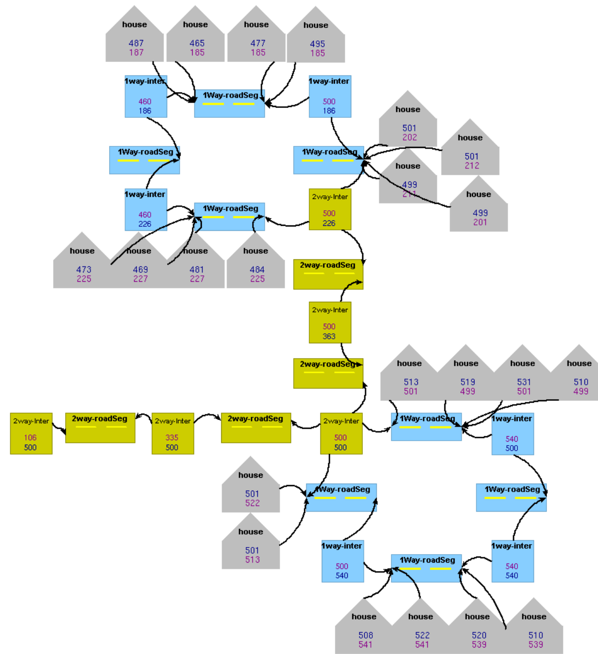

Fig. 15. Phase IV: Road Fragmentation rules execution results

a given rule is executed, and the rules execution order, varying either ones will ensure map distinctiveness. Within this project, we have shown that procedural techniques for city generation can also be used with graph grammars and graph transformations.

References

1. George Kelly, Hugh McCabe: A Survey of Procedural Techniques for City Generation ITB Journal **Issue: 14** (Dec 2006)

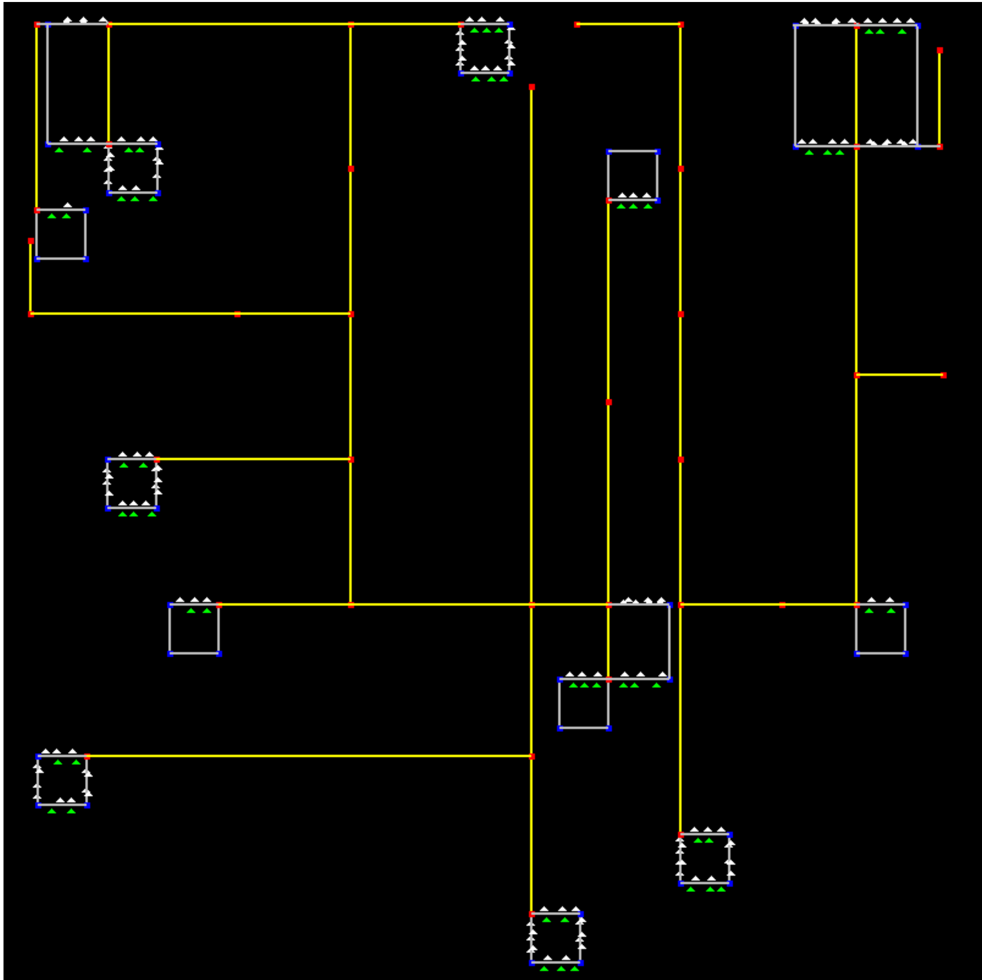

Fig. 16. More advance map generation