**Week 6: Prototype 2 – L systems (continued)**

This week I looked further into creating L systems that would generate road networks, this involved making my own rule sets and my own actions/commands for each character of a sentence.

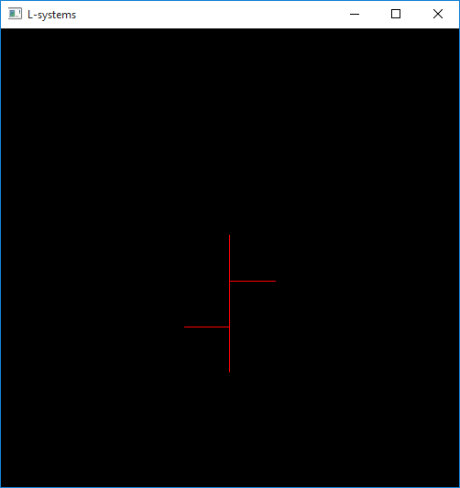
First, I looked at creating rule sets that would work with my current Turtle Graphics implementation; this isn’t how my final L-system will work, however it allowed me to experiment with different rule sets to get an understanding of what behaviour I wanted.

After looking at (GameDevStackExchange, 2014) I created a simple ruleset:

Axiom: X

Rule: X->F[+F]F[-F]F

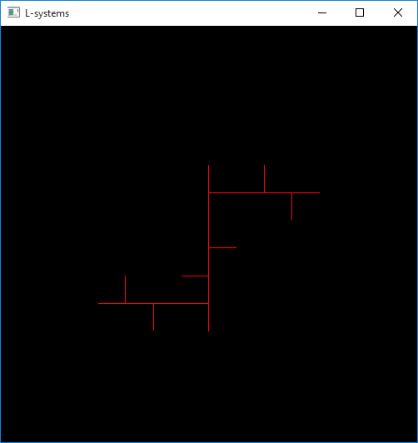
This created a single road with two roads branching off of it, one on each side:

*Generation 1*

From here, I could adapt the rule set to allow for roads to branch off of the branches themselves, and so on:

Rule: X->F[+FX]FX[-FX]F

Here, X doesn’t correspond to a certain action (such as move forward), it is simply used to control the generation of the sentence; this is a technique seen in other L-systems such as some Fractal Plant implementations.

*Generation 2*  


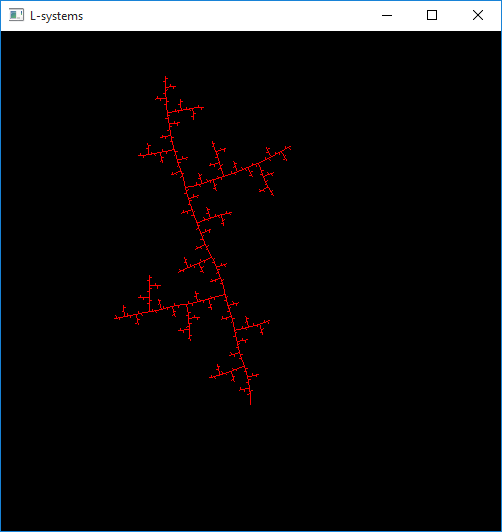
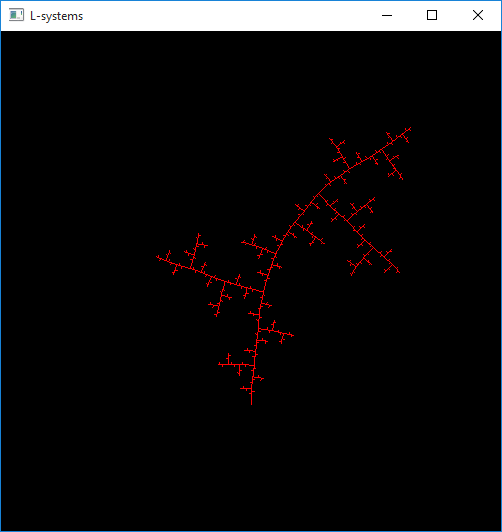
By using this ruleset, I could create a branching network of roads, however, it would be the same every time. For this reason, I wanted to add some variation into my L-system, this could by done by introducing randomness. As I mentioned previously, there were two ideas offered of how to do this; “Variation can be achieved by randomizing the turtle interpretation, the L-system, or both” (Prusinkiewicx, P and Janan, J, 1992).

I started by implementing the first concept; randomizing the turtles interpretation. I added an extra character, A, to my rule set, which can either rotate the cursor by positive two degrees or negative two degrees.

New ruleset:

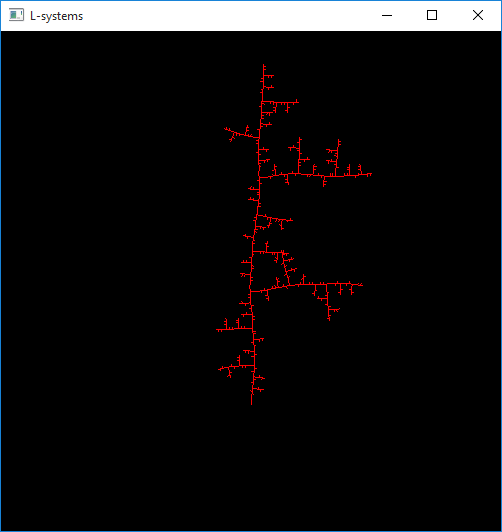
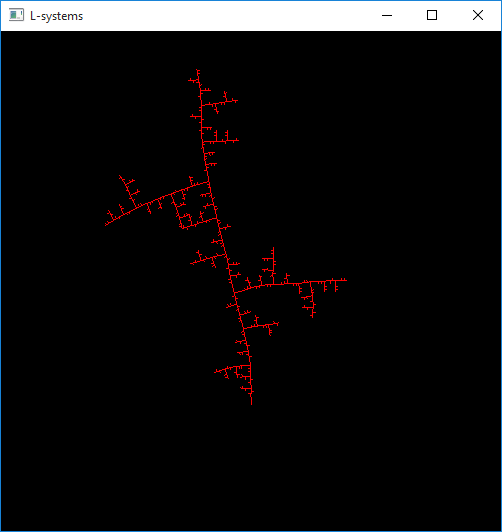
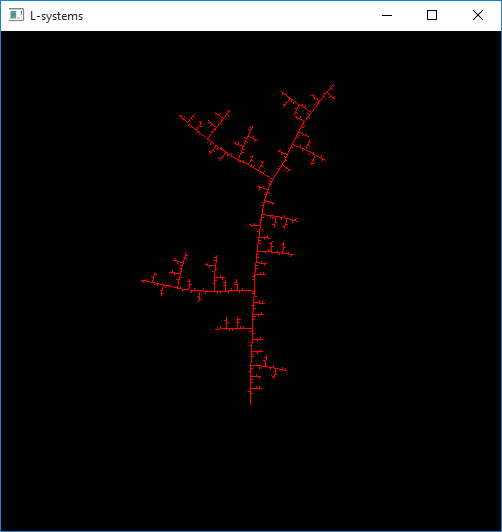
Rule: X->AFX[+FX]AFX[-FX]AFRX

This resulted in some variation in the shape of the roads:

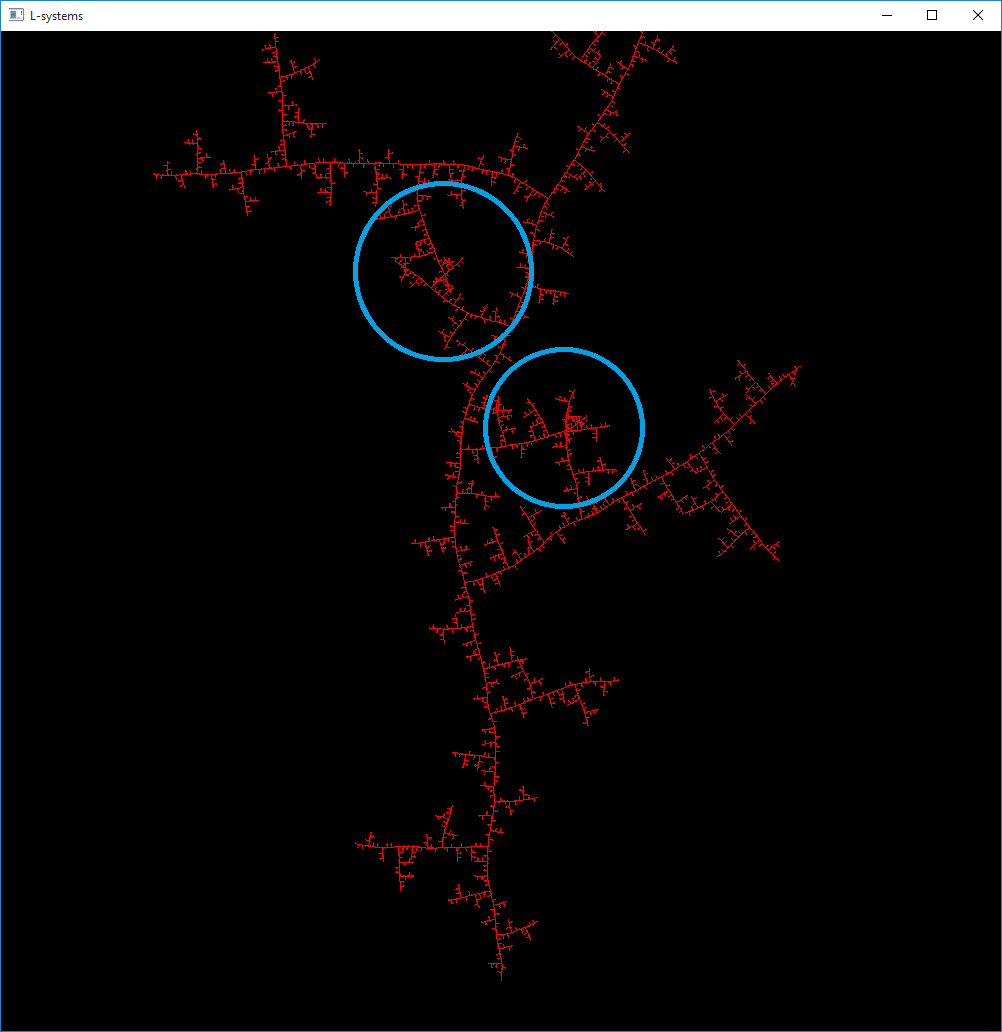
*Generation 4*  


Whilst this resulted in the road network looking a little different each time I ran the L-system, the structure of the road network remained the same: Extend forward, branch left, extend forward, branch right, etc. For this reason, I next added an extra random action: I swapped the ‘-‘ and ‘+’ characters with ‘~’ which would have a 50% chance of branching either left or right. This resulted in some variation of the structure of the road network:

Rule: X->AFX[~FX]AFX[~FX]AF

*Generation 4*

Currently, due to the fact that the L-systems are being interpreted and drawn with turtle graphics, there is no checking for global goals or local constraints. Global goals and local constraints are rules by which the l-system abide (Ilangovan, K, P, 2009), (Kelly, G and McCave, H, 2006), (Parish, Y I H and Muller, P, 2001). A local constraint common in road generation is that any roads that would intersect each other instead join to create a junction.

*Generation 6 – road overlapping*  


In order to implement this I had to make some large changes and swap from using a turtle graphics system.

I implemented my own adaptation of a turtle graphics system, to do this I created my own class RoadTurtle, and a new Road class. This system was similar to turtle graphics, however focused on creating Road objects as opposed to drawing lines. Because each road now knew information such as what road it is branching off of and its physical location, I could abide now add in behaviour to ensure that roads were all abiding by local constraints (for example, roads would not directly cross over each other, they would instead create a crossing).

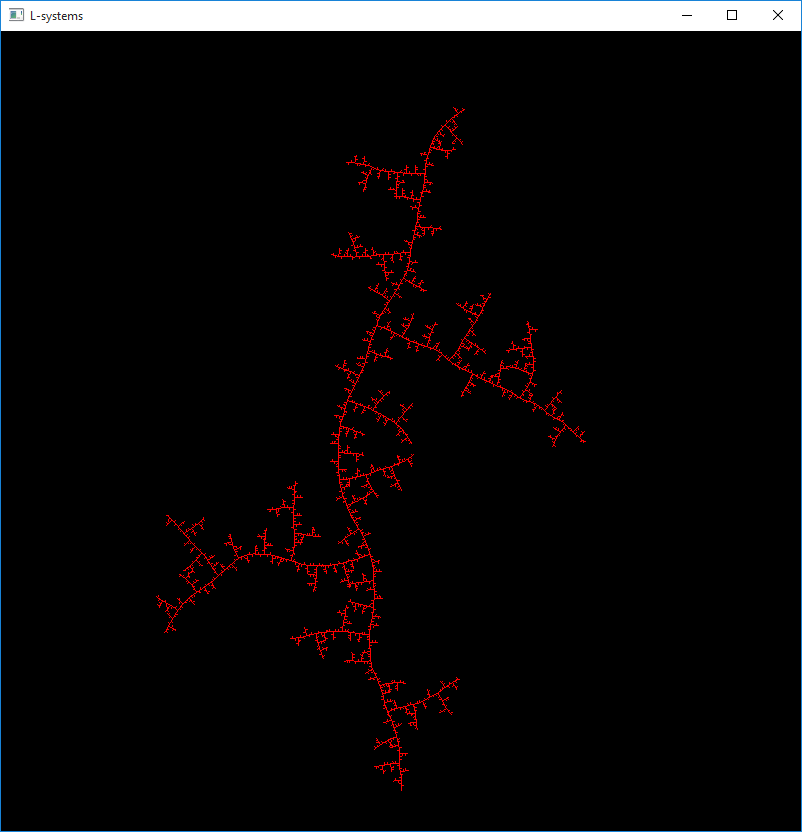
// @@ EXPLAIN IMPLEMENTATION DETAILS //

// HERE ARE THE HEADERS FOR THE CLASSES I MADE

// EXPLAIN LOGIC

//

Afters some tweaking and refining it draws and displays identical to the turtle graphics system, however this implementation is greatly improved because all required information is stored appropriately in each road object.

*Generation 6 – New turtle system*  


This means that local constraints can be applied to the road network.

/// @@ LOCAL CONSTRAINTS ATEMPT @@ //

I had attempted to join roads that would have otherwise intersected, this is a common local constraint in L systems (@@REFFERENCES HERE@@). I attempted to implement this by, whenever I extend a road forward, checking all other roads to see if the road being extended is within a certain distance of this road, if it is; I join the road I am extending to the road it is near, this creates a junction and joins the road network together more.

I was unsuccessful in implementing this, I feel I did not take the best approach. My approach resulted in my checking every other road every time I extended a road forward and this became largely costly.

I suspect there are ways I can improve this implementation, I can separate the world space to ensure I am only checking roads in a similar area to mine, as in higher generations I am checking upwards of two thousand roads, when less than five percent of those roads are necessary to check.

///

/// @@ ADDING NEW RULES FOR MINOR ROADS ATTEMPT @@ ///

Currently, all roads followed the same rule, and thus there was no distinction between major and minor roads. I experimented and created different rulesets so that minor roads and major roads would have different characteristics.

My intention was that minor roads would branch off of the major road and these roads would make up a majority of the city, this is common (@@REFFERENCES@@)

I attempted to create some rule sets that would ensure minor roads followed different rules than major roads:

Axiom: X

Rule1: X->AEWX[~FQ]AEWX[~FQ]AEWX  
Rule 2: Q->EW[~EEL] QEEQ[~EEL]Q  
Rule 3: L->ELELEL

Where:

A = Chance to rotate by -2, 0 or 2 degrees  
E = Extend road by 10  
W = Extend road by 1-5  
~ = Branch left or right (50/50)  
F = Extend road by 1

X, Q and L are simply used to control the growth of the sentence.

lsys.AddRule('X', "AEWX[~FQ]AEWX[~FQ]AEWX");

lsys.AddRule('Q', "E Q [~EE L] Q EE Q [~EE L] Q");

lsys.AddRule('L', "ELELEL");

@@ EXAMPLES OF RULE SETS USED

@@ IMAGES OF THEIR OUTCOME

///

// ------------------------------------------------------------------------------------ //

// @@@ TALK ABOUT HOW I PRUNED ROADS THAT WERE TOO SHORT

//

I ADDED METHODS TO CHECK X, SO NOW Y… BECAUSE Z…

NOW ROADS THAT ARE TOO SHORT ARE NICE INCLUDED IN THE TURTLES FINAL LIST OF ROADS

//

**Evaluation:**

Bibliography

GameDevStackExchange (2014). Using L-Systems to procedurally generate cities. (22 October 2014) [Online] Available at: <https://gamedev.stackexchange.com/questions/86234/using-l-systems-to-procedurally-generate-cities> [Date of access: 20 October 2017]

Goldman, R., Schaefer, S. and Ju, T. (2004). Turtle geometry in computer graphics and computer-aided design. \*Computer Aided Design\*, 36(14), pp.1471-1482.

Ilangovan, K, P. (2009) Procedural City Generaror, MSc thesus, Bournemouth University. Available at: <https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc09/Ilangovan/Thesis_i7834000.pdf> (Accessed: 30 May 2017).

Kelly, G. Mccave, H. (2006). “A Survey of Procedural Techniques for City Generation”. In ITB Journal, No. 14.

Parish, Y, I, H. Muller, Pascal. (2001). “Procedural modeling of cities”. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH ’01). Acm, New York, NY, USA, 301 – 308.

Prusinkiewicx, P. and Janan, J. (1992). “Lindenmayer systems, fractals, and plants.” New York, N.Y.: Springer-Verlag.

Tmwhere (Date unknown) *Procedural City Generation* (Date unknown) [Online] Available at: <http://www.tmwhere.com/city_generation.html> [Date of access: 28 october 2017]