Dan Hervé

CSCD 580

Task 1

Part 1:

What I did: I created a triple object that contains 3 doubles (x, y, z) and a line3D object, which contains a startpoint, endpoint, and length. The triple has the capability to be constructed with random values. The number of lines desired is set in the call to the herveTest1 method.

How well did it work: it worked as expected. No problems here.

Pros and Cons: The pros are that the classes contain many methods that give functionality that can be used and built on for further tasks. It did everything I needed it to, so I did not identify any clear cons.

What to improve: The tester could be made more robust in terms of starting options. In particular, a command line setup would be nice.

Part 2:

What I did: I created the ability to set intervals in Part 1 via a method I added to the Line3D class. When a double interval is input, it will return a Triple[] of the size of the interval. I also added a helper method in the Line3D class to quickly calculate length of the 3D line, which is just a little more work than I want to do more than once.

How did it work: Again, this task went smoothly.

Pros and Cons: Having the method in the Line3D class is both a pro and con. Because of my setup, it worked nicely to integrate it here, but if I wanted to come back later and only use Triples to plot lines, I would have to rewrite my method.

What to improve: It I did it again, I would probably create a data structure that held Triples, which would give me more versatility.

Part 3:

What I did: I created a new class of tools called Pathing, which holds a number of static methods to use. For this part, I created an orient() method which takes a Line3D and endeavors to generate a Triple that contains the yaw and pivot info. The z slot is saved for roll later. I attempted to calculate pitch and yaw by 2 2d views, x, y for pitch, x,z for yaw, and using tan to calculate the angle. I did not have a lot of luck, likely because I didn't properly understand how I was calculating these values relative to the world.

How did it work: Though I could get the bird pointed in various directions, it only seldom pointed in the direction I wanted.

Pros and Cons: While I feel my approach was the right one, it clearly has not born fruit yet. The one method is efficient in providing the necessary data, since it is only around 10 lines of code, but it is missing something.

What to improve: I need to get the bird facing the right way…

Part 4:

What I did: I created a plotLineRandom() method in my Line3D class that works like the original plotline() method, but moves the y-values and x-values up and down randomly. I made sure that it this variance was only slight and that it could not dip below the ground.

How did it work: it definitely looks less artificial than strait lines, but it can be erratic at times due to randomness. It runs fine.

Pros and Cons: It does break up the lines quite nicely. A couple cons are that it can look erratic, and the method within the Line3D class means that I would have to make a new solution if tracked my points in another fashion.

What to improve: If I could make the rises and dips act more like a wave, it would make it appear less unreal.

Part 5:

What I did: I applied Bezier curves to the lines to give them a great smooth look. I had to create a couple new methods in my Pathing Class, one to calculate the points given a Triple and a t-value and one to try to smooth the sharp edge when two curves are put together. For this solution, I ditched the Line3D class and used and ArrayList to hold control points and rendering points.

How did it work: Half of the edges are beautifully smoothed, and the other half are better, but some jagged edges can still occur.

Pros and Cons: The main pro is that when it works, it looks great. The con is that it is a fairly complex solution and I still have not managed to nicely merge multiple difference curves.

What to improve: get those curves to merge together.