Owen Martin

Mrs. Awde

Academy of Information Technology III

3 November 2017

Quarter One Report of Progress

My project-based AOIT III project, CELL, was only partially successful at fulfilling the expectations I set for myself at the beginning of the school year. The spacial matrix that constitutes the area of program operation was created as expected and caused no difficulty. In lieu of creating a two-dimensional array of consecutive integers to determine object positions, I made use of the computer's screen resolution; the standard unit of length is one pixel, and both position and distance are determined by their appropriate quantities of pixels. Once this was decided, my first real step was to create a Particle class. Originally, there were only get/set methods for certain particle attributes such as fx, fy, px, and py, but eventually, two interparticle force methods were created: the circumferential elastic and repulsive forces. As each Particle object is stored in an array at the beginning of the program, every particle can exert a force on its neighbors by referencing their array indices. The circumferential forces were designed so that each particle would exert a force on the particles whose indices were one greater or one less than their own – every particle would pass these indices into the addRfC and addEfC methods, thereby applying a distributive force on their immediate neighbors.

For most of my time working on CELL, perfecting these distributive forces has been an ongoing challenge. At first, a pair of bound particles would fly off to one side of the screen and never conserve their momentum; I soon realized that I neglected to use the absolute value function in Java to negate any unwanted changes in direction, determined by the difference of two particles' positions on both the x and y-axes. After fixing this, it became obvious that the distances between two test particles grew after every time they passed each other. This increasingly oscillating distance was the exact opposite of what I needed to happen, so after considerable reflection I realized I could reverse the principal and obtain the desired result: I added some code to the program that created a less intense repulsive force after the two particles changed directions, making the distance between them decrease with each oscillation. Although the particles now appeared to stabilize at a certain distance, this only occurred on a single axis or when the ratio of their x to y distances was 1:1. After observing the particles' motion with different starting ratios, I began to understand that the method by which I distributed the forces along each axis was inaccurate. When I would determine the magnitude of the force applied to the x and y-axes of each particle, I would only use the distances between the two particles' x and y positions instead of the actual distance between them, or the hypotenuse of the triangle their component vectors would form. Fixing this retained their x to y ratios.

Currently, the CELL program has a significant fundamental inaccuracy. To stop particles from crossing over each other due to an overwhelming elastic force, setVx and setVy methods were introduced; the velocity of a pair of particles is set to zero if their distance falls below a certain number of pixels. This is ultimately a poor program solution because no motion can occur once this happens – the cell membrane becomes static. In an effort to rectify the situation, I have come across a new method of computing particle interactions called Verlet integration, which seems promising and will likely be the focus of my efforts in quarter two; the remainder of my expectations will be pushed back to complete later in the year.