**Chris Schiff – Final Project**

The algorithm I used for the 2D soft-body physics itself was based off of the Triple Spring example from the Linked Spring file on MyCourses. Given the positions of two masses in a mass-spring system, calculate the position deltas in each dimension and the angle between those deltas. Then, have one of the masses move to a point along the straight line to the other mass. The point’s x-position depends on the cosine of the calculated angle, while its y-position depends on the sine. Since the sine and cosine functions oscillate between 0 and 1, the algorithm uses this to quickly approximate the behavior that would result from using finite-element analysis and Hooke’s Law on the mass-spring system. The springing force is applied directly to the mass as an impulse, rather than as a force. This algorithm has variables for the springing force, the normal length of any given spring, and the spring friction, all of which can be tweaked in the Unity editor. To help provide a better sense of realism, two additional forces are applied on the soft-body. First, gravity is applied as constant acceleration. Second, when any given mass in the soft-body collides with the edge of the camera’s view, an impulse (which can also be adjusted in the Unity editor) is applied directly on the mass.

Post-Mortem: I would like to implement an additional algorithm that can prevent the masses from piercing through the opposite end of the soft-body. My current algorithm does nothing of the sort, so I would like to see what this would entail, even if it means something like calculating the system’s center of mass every frame and having masses spring away upon reaching a certain radius of it. Another feature I would like to add is the handling of collision between two soft-bodies. If I choose SAT for narrow-phase collision detection, it would be interesting to determine the most efficient way to get the local axes and moment of inertia for a given soft-body.