**Logic Overview:**

Some of the inherent public traits of the Newtonian object class will include its velocity, as this will need to be modified quickly and efficiently; the two dimensions of its net velocity (x,y) = netVel[x-value,y-value], because velocity is just the sum of x and y vector components; its mass because this is one of the few variables to change to result in non-uniform accelerations; as well as the actual size of the Newtonian object on screen. The set traits will include acceleration due to gravity of a given Newtonian object which will be determined in the class as a mathematical function of its mass and the distance between another mass using the inverse square law. An arbitrary gravitational constant will have to be built into the program to allow the numbers to work nicely with one another so unit conversion will take place immediately following a public trait’s entrance into the NewtonianObject constructor to allow the program to move smoothly. Distances will be arbitrary as the scale of real gravity would be far too undetectable therefore a given pixel will not directly correspond to a distance unit in real life.

Vector logic:

-a vector can be considered a numeric quantity with direction, and magnitude

DIRECTION: Absolute value of distance between objects determines which way to move, this only works at right angles SO to change the rate along each x, y distance the program has to take the sine of the inverse tangent of the angle to get the direction of acceleration

MAGNITUDE: (Inverse square law) multiply the math above by the gravitational constant in the program, a corresponding mass, then divide by the distance between the two, in this way all objects are attracted to all others at varying magnitudes as a function of their distances and respective masses (just like in real life) ~~Next up special relativity~~

**To double an array (the reason for this was explained earlier):**

*Loop through array, each step sets the newly generated element the element of the current index of the loop*

**To make all things attracted to one another:** *(Solving attraction of the array of ‘things’)*

*Loop through half of a doubled array (terms repeat once as stated above)*

*Within this loop is another of the same length and calculates the attraction of one object to the next object denoted as the current index plus the value of the inner loops’ counter variable. With a variable to attract each object to each other object, always ‘i’-places ahead in the array of things*