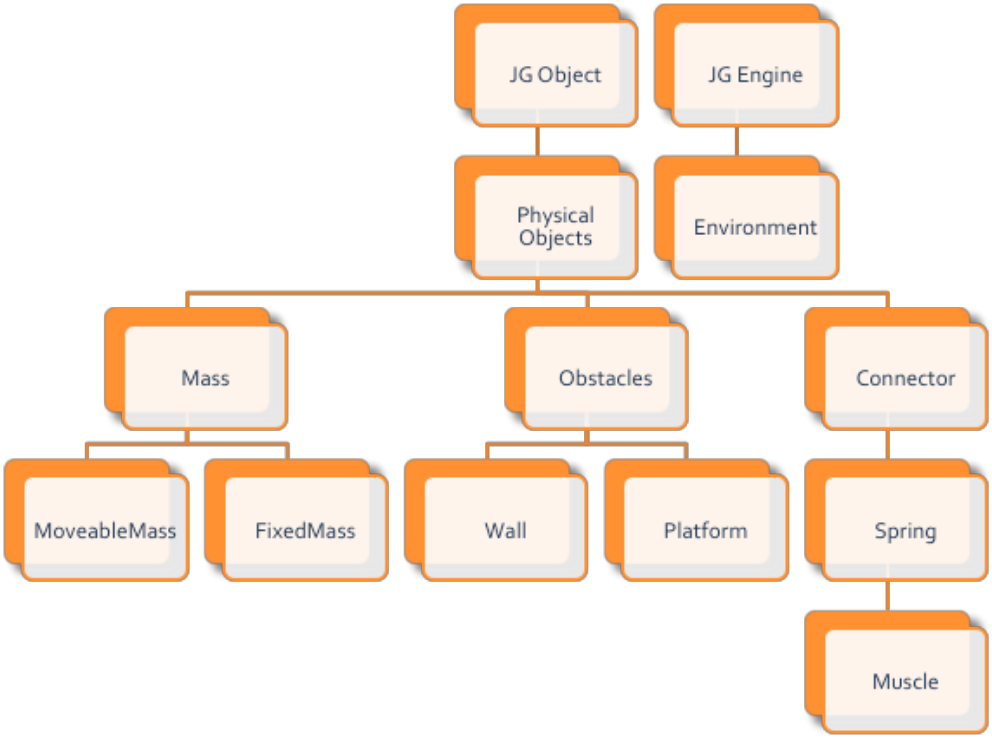
Cody Lieu
Zanele Munyikwa
Compsci 308
1/28/2014

Springies Part 1: Design Document



We have several objects that inherit from the JGObject Interface

1) Physical Objects (which extends JGObject)
    a) Mass
        i) MoveableMass: ID, x, y, vx, vy, mass
        ii) FixedMass: ID, x, y, mass
    b) Structures
        i) Wall: ID, magnitude, exponent
        ii) Platform/Blocks: ID, magnitude, exponent
    c) Connector
        i) Spring: ID m1, ID m2, rest length, K constant
        ii) Muscle: ID m1, ID m2, rest length, phase, amplitude, K constant

In order to make our code the most flexible/extensible, we looked for objects/classes that served similar functions. With this in mind, we thought about how an inheritance hierarchy could be implemented so that adding on later parts could be done most efficiently. For instance, springs and muscles serve similar functions. The difference between the two is that springs

exert an outward force on the masses when compressed and an inward force when extended while muscles compress and extend according to a harmonic motion determined by the muscle's rest length. Even with these differences, however, muscles and springs serve one basic function, they connect two masses. Therefore, we are planning on making a connector superclass that we can use to make the subclass that performs the functions of springs. Since muscles are basically springs that expand and contract in time with an energy wave, we can just extend springs to create muscles. By implementing springs and muscles in this fashion, we have created a connector superclass that can be used later to create objects that connect masses that have a slightly different function than springs and muscles.

Similarly, some of the example xml files that the assignment linked showed us that wall objects need to be created. These wall objects serve as the outermost boundaries and tell the objects what to do when they make contact (bounce off, reverse motion, stop, etc). Even though walls are the outermost boundary, the fact that we can create objects that check for collisions with the masses and springs could be useful later. Therefore, we generalized this concept to a structures superclass. This structures superclass could be used not only to make walls, but also to make other objects on the field like blocks that the moving objects make contact with.

Finally, we will also have a mass superclass that will be used to implement the subclasses fixed mass and movable mass. These two objects will be very similar. They will both have ID's that can be referred to by other objects like springs, x position values, y position values, and the mass value. The only difference between the two is that the movable mass will also have velocity values for the x and y direction. While the initial x and y velocities of the movable mass can be set to zero, its velocity can still change if a force is exerted on it. The fixed mass will not move no matter what force is exerted on it. It will instead exert a force back on the object.

We would also have an Environment class that holds the state of forces. These would include variables such as gravity, viscosity, center of mass, and wall repulsion. They would only apply to mass elements since all other elements such as springs and muscles are considered massless. Making this Environment class will allow us to store states in order to pass them to the WorldManager Class. Also, it will be instrumental in implementing the Muscle class. Although we will not be directly implementing the physics through this class, this will be where JGame and JBox pull useful information for the simulation. Gravity, friction, and springiness will be stored in the environment class. The environment class will also have methods in which it modifies physical objects based on changes in these various states. For example, increasing gravity would increase the amount of force that movable masses and fixed masses exert on other objects in the direction of gravity.

To do all of this, we need a scanner class that would parse xml files. This class would read in assemblies from an xml file that creates physical objects and sets the initial values of the environment. The scanner class should be able to take in new information, allowing the user to add new assemblies, read from a new data file, or stop the simulation and clear it. Since the scanner class is only responsible for reading in previously constructed models, we do not need to build a GUI. Additionally, JGame and JBox are responsible for handling/animating the physics simulation, so the scanner would only have to read in the environment data file and set the initial values for values like gravity (direction and magnitude) and viscosity.