**COSC 460 Project Description – Fall, 2016**

Write a Java program to simulate the interaction of entities in an infinite two-dimensional space under the influence of gravity.

Entity Parameters

Each entity will be treated as circular and will be characterized by the following basic parameters: position, velocity, radius, and mass.

The “position” of an entity is given by its (x, y) coordinates relative to a fixed coordinate system. At the beginning of any simulation, the graphics window will have dimensions rows x columns (default values are 800 x 800 but can be changed at the beginning of any run) and will map an XY coordinate space from (-columns/2, -rows/2) to (columns/2, rows/2). This, of course, positions the center of the coordinate system at (0, 0) and sets the initial scaling so that one pixel = one unit in the visible coordinate space. Any entity, however, can be located at any position, including outside the visible area.

“Velocity” for any entity consists of vx and vy components (i.e., the components of its velocity in the x and y directions).

“Acceleration” for an entity consists of ax and ay components (i.e., the components of its acceleration in the x and y directions).

The “radius” of an entity is initially defined relative to the scaling established at set-up. At the beginning of a simulation run, for example, an entity of radius .5 (and hence a diameter of 1) would be approximately one pixel in size.

The “mass” of an entity is equal to its area. Thus for an entity of radius r, its area (and hence its mass) is simply πr2. Making this further simplification and saving the mass in the entity parameters data structure (until it changes) will improve processing performance. Because area = πr2, r = (area/π)1/2, or just (since mass = area) r = (mass/π)1/2. This will be useful for recomputing the radius for an entity formed after the collision of two or more entities. Note: because the user can zoom in or out, the apparent (displayed) area of an entity is relative to the window parameters. That is, zooming in (out) will make the area of a circle look larger (smaller) but its actual area as a fraction of the displayed window will be unchanged (as will its mass). In other words, mass is always related to the original (starting) absolute coordinate system.

Interaction

Every entity affects every other entity due to the gravitational attraction between them. The magnitude of the force f of that attraction between any two bodies with masses m1 and m2 is given by Newton’s Law of Universal Gravitation as f = Gm1m2/r2 where (in this case) r is the distance between their centers and G is the universal gravitational constant. This force will act along the line connecting the centers of the two bodies and the magnitude of the corresponding acceleration a of one (with mass m1) toward the other can be calculated from Newton’s Second Law of Motion (f=ma for some mass m). Thus, because the forces in the two expressions are the same, to compute the acceleration of m1 toward m2 we have a = Gm2/r2. (Note that m1 cancelled out.) A similar calculation computes the acceleration of m2 toward m1. Unless m1 and m2 are the same, the accelerations will be different. For our purposes, it only matters that G is a constant, so we do not have to use the actual value. Instead, we will leave it in the equation, recognizing that its magnitude simply influences the value of the acceleration in a proportional way (i.e., if one has been using a value G1 and decides to use 2\*G1 instead, value for acceleration will be twice what they would have been for G1). Our default value for G1 will be 1.0 but the user will be able to change this at any time during a run.

Because of gravity, the velocity of an entity can change during a time increment t as **vt+1** = **vt** + **a**t (where **a** is the acceleration due to gravity). The distance traveled in time increment t by an entity whose velocity is **v** at the beginning of the time increment is **d** = **v**t + ½**a**t2. Note that this is a vector equation, so the actual trajectory (if there is non-zero acceleration) could be non-linear. However, we will make the simplifying assumption that during a short enough time step the trajectory is linear. This will make it easy to compute the endpoint of some entity’s motion during a time step, thus simplifying subsequent calculations. Of course distance is relative to the original mapped absolute coordinate system (set at the beginning of a simulation run). Consider, however, that if zooming occurs, the *apparent* distance in the graphics window can change.

Note that, because every entity affects all others, all pairwise interactions must be considered in calculating the overall acceleration, which will have components in both the x and y directions. For this simulation, all acceleration and velocity calculations will be made for each time step and no further changes to the computed values will occur during that time step.

*Example:*

Consider a body of mass m at initial location (xi, yi) at the beginning of a time step. Imagine that this body is acted on by three other bodies a, b, and c which exert forces **fa**, **fb**, and **fc** on the body under consideration. As indicated above, the acceleration on the body from these three sources will be **aa** = Gma/ra2, **ab** = Gmb/rb2 and **ac** = Gmc/rc2 (where the r values are just the distances between the body under consideration and each of the other bodies). These equations provide the magnitude of the respective accelerations. Because we know the position of each pair of bodies and because the acceleration acts along the line between each pair, it is easy to compute the x and y components of the acceleration for each. For instance, if body a is at position (xa, ya) at the beginning of a time step, then the attractive force that body a exerts on the body under consideration is along the line with slope (yi – ya)/(xi - xa). The acceleration components in the X and y directions can be determined from the values in this expression because we can tell if the force is acting toward a particular quadrant or along one of the axes. Use the supplied method *computeVectorAngle* to get the actual angle the acceleration vector makes with the x-axis (paying close attention to the direction of the force when specifying the method parameters!), then multiply the magnitude of the force times the cosine and sine of that angle to get the acceleration components in the x and y directions, respectively. Do this for all three acceleration vectors, then simply add the x and y components to get the total acceleration in the x and y directions (e.g., the total acceleration on the body under consideration in the x direction is ax = aax + abx + acx and the total acceleration on the body under consideration in the y direction is ay = aay + aby + acy. You can then compute the final location (xf, yf) of the mass m at the end of the time step as xf = xi + vxt + ½axt2 and yf = yi + vyt + ½ayt2 where t is just the duration of a time step (as defined below). Repeat this for all bodies to determine their final positions at the end of the time step.

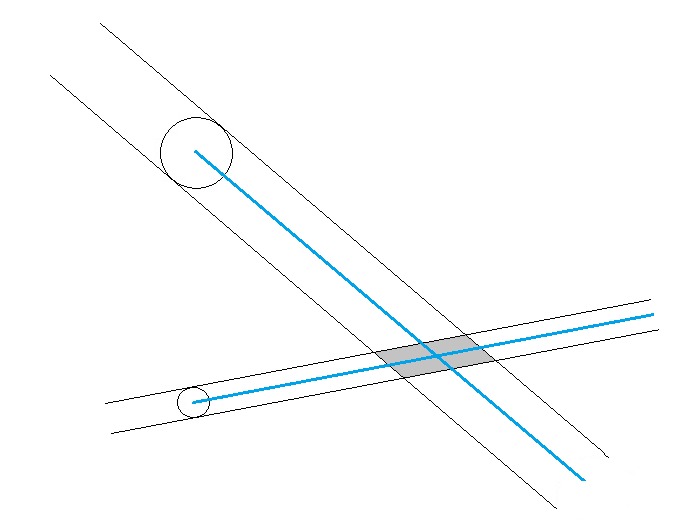
Collisions

We will assume that all collisions are perfectly inelastic (i.e., the two colliding bodies stick together) and that momentum is conserved. Consider, for example, two colliding bodies having masses m1 and m2 and velocities **v1** and **v2**.

Following a collision between these two bodies, the combined body will have mass m3 and velocity **v3**. But m3 = m1 + m2 so, due to conservation of momentum, we have **p3** = **p1** + **p2** or m3**v3** = m1**v1** + m2**v2** or **v3** = (m1**v1** + m2**v2**)/m3. Since **v1**, **v2**, and **v3** are vectors, we can simply compute v3x = (m1v1x + m2v2x)/m3 and v3y = (m1v1y + m2v2y)/m3 to get the x and y components of the velocity of the combined entity.

The area of the newly formed entity will be m1 + m2 = m3 (since mass = area) and the radius can be calculated as described above.

If more than two bodies collide at the same time, simply extend the calculations above to compute the overall x and y velocity components and the new radius based on the combined masses. Note that the colliding bodies cease to exist as individual entities after the collision and that only the newly formed body remains.

To compute whether bodies will collide it is necessary to determine if they will occupy the same space *during* a time step. The diagram below shows the areas swept out by two entities on a possible collision course. (It is important to remember that bodies can collide not only if they are moving toward each other but also if one overtakes the other (which is really the same thing in a relative sense!).)

Let the point in the diagram where the two trajectories intersect along the blue lines be (x, y). This is easily computed using basic algebra if one knows the equation of each line. Those equations are likewise easily computed using the point slope form for a straight line since we already know the current location of the center of the body (i.e., a point) and the direction of travel (i.e., the slope). For example, consider two entities 1 and 2 with slopes s1 and s2 and starting locations (x1, y1) and (x2, y2), respectively. If the trajectories of these entities intersect, it will be where the lines y = s1\*(x- x1)+ y1 and y = s2\*(x- x2)+ y2 intersect. Solving for x we have:

s1\*(x- x1) + y1 = s2\*(x- x2) + y2

s1\*x- s1\*x1 + y1 = s2\*x- s2\*x2 + y2

s1\*x- s2\*x = s1\*x1- s2\*x2 + y2 - y1

(s1- s2)\*x = s1\*x1- s2\*x2 + y2 - y1

x = (s1\*x1- s2\*x2 + y2 - y1)/(s1- s2)

y can now be computed using either of the equations above.

Note that if s1- s2 = 0 in the expression for x, the lines are parallel, yet the entities could still collide. This should be handled in the code.

In any case, if the trajectories intersect during the time step or if the entities pass within the sum of their radii from each other during the time step, then the entities collide.

If the time to intersection is within the time it takes for two entities to come within the sum of their radii of each other, there is a collision…

*We’ll work out the simplifying details of collisions a bit later…*

Time Steps

We will call the basic unit of time in the simulation a “time step.” The value for the duration of a time step is used directly in the applicable equations. At the beginning of a simulation run, the duration of a time step has value 1.0, but that value can be changed (doubled or halved) by the user at any time. Consider, for example, an entity moving in the x direction with velocity 2 and accelerating in the same direction at a rate of 4 at the beginning of a time step with duration 1.0. Assuming no collisions, the x component of velocity at the end of the time step is vnewX = voldX + at = 2 + 4\*1 = 6 and the distance traveled in the x direction during that time step is dx = voldXt + ½axt2 = 2\*1 + ½\*4\*12 = 4. One time step is the maximum amount of time that passes for each iteration of the system, but as described later, it is possible for less time to pass on any given iteration if there is a collision.

Timing adjustments when collisions occur

Coming…

Entity Color

Will be based on mass as described later…

Set-Up

Radius

Random

Identical

Specify

Position

Random

Group

Specify

Velocity

Random

Explode

Converge

Specify (default to random)

Menu

Iterate I

Zoom ZI=Zoom In; ZO=Zoom Out

Pan PU=Pan Up; PD = Pan Down; PL=Pan Left; PR=Pan Right

Time TD=Double Time increment; TH=Half Time increment

Change C

Restart R

Exit E

Option C permits the user to change certain parameters during a run. These include:

The value for G used in calculating acceleration due to gravity

Whether to use solid colored entities or outlines only

*Recall that this is still a work in progress and additions and changes will be forthcoming…*