PPS 2015 project 2: Planet Builder

Orestis Polychroniou George Koloventzos
{orestis,gkoloven}@cs.columbia.edu

Abstract

This document provides the background relevant to project 2 of Programming and Problem Solving Fall 2015. Section 2 describes the mathematics and physics of the orbits. Section 3 describes the mechanics of pushes and collisions. Section 4 describes the simulator and the default player `g0`.

1 Project Description

Asteroids of equal size orbit the sun in initially circular orbits at various radii. There is a lower and upper radius bound, so that the asteroids are in a formation like the asteroid belt between Mars and Jupiter. You get to push the asteroids a bit to change their trajectory. When asteroids collide there is conservation of momentum and the asteroids remain joined (and larger). Energy is not conserved: the newly formed asteroid heats up! Your goal is to create a single planet made up of at least half the initial asteroid mass within a fixed time T. Each time you exert a push, the energy required for that push is accumulated into your score; you want to minimize the total energy used. (Using a lot of energy to form a planet quickly scores worse than forming the planet more slowly with less energy, as long as the planet forms within T time units.)

Asteroids are influenced in their trajectory by the sun, whose gravity is strong. The gravitational attraction of other asteroids is assumed to be negligible and is ignored by the simulator. All interacations (including your pushing of asteroids) happen in the plane of rotation of the solar system. A push perpendicular to the movement of the asteroid will change its orbit, but sill retain the circularity of the orbit; a non-perpendicular push (or collision) may lead to an elliptical orbit.

2 Orbital mechanics

The orbit of an asteroid around the sun follows the Kepler laws of planetary motion. We explain how they are used as we go. The orbit of an object with only a single force attached to it will be in *some* orbit around the pulling object. In our setting, the force is the gravitational pull of the Sun. We assume there are no gravitational pulls between the planets and that the Sun is always at $(0, 0)$. Here, the space is 2-dimensional thus all vectors are of the form (x, y).

2.1 Ellipse (in Mathematics)

The orbit of an asteroid with periodic motion is always an ellipse. An ellipse can be summarized by two foci: $\vec{f_1}$ and $\vec{f_2}$ and the semi major axis (or major radius) a. The ellipse is defined as all the points whose sum of distances from the foci are constant ($= 2a$). The center $\vec{c}$ is the mean of the foci: $\vec{c} = (\vec{f_1} + \vec{f_2})/2$. The equation of an ellipse with $\vec{c} = (0, 0)$ and both foci on xx' is:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

where b is the semi minor axis (or minor radius). This ellipse can also be thought of as a circle of radius a where the y–coordinate is scaled by b/a. The distance between the foci is $2ae$, and e is defined as the eccentricity of the ellipse. Normally, we assume that $a > b$ and $0 < e < 1$. For instance, if $a < b$, the ellipse would be rotated by $\pi/2$. The eccentricity also relates a and b as follows:

$$e = \sqrt{1 - \frac{b^2}{a^2}}$$

Also, the distance between the foci is $2ae$ (and ae from the center). If $\vec{c} = (x_c, y_c)$ and two foci are parallel with xx', the ellipse equation becomes:

$$\frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2} = 1$$

In the general case, the ellipse would not be parallel with xx' but be rotated by some angle A, which is defined as the angle formed by the line that connects the foci with xx'. The general ellipse with center (x_c, y_c) and rotation A is:

$$\frac{((x - x_c)\cos A + (y - y_c)\sin A)^2}{a^2} + \frac{((x - x_c)\sin A - (y - y_c)\cos A)^2}{b^2} = 1$$

Note that while the final equation may appear complicated, in reality it is a simple transformation of the initial equation. If we translate the points (x, y) of the last equation to a different system by rotating the axes by A and by transferring the center to (x_c, y_c), we go back to the simplest ellipse equation.

2.2 Elliptical orbits (in Physics)

A object will have an elliptical orbit around some point P, if the motion is periodic and there is a pulling force, pointing to P, of the form (α is a constant):

$$F = \frac{\alpha}{r^2}$$

One focus of the ellipse is always the pulling body (Kepler's 1^{st} law). Here, the focus is the Sun, and we fixed it at $(0, 0)$. Thus, we can compute the center (x_c, y_c) from a, b, and A using the distance from the focus ($= ae$) and the inverse direction of the ellipse rotation ($= A + \pi$). Using only a, b, and A:

$$(x_c, y_c) = (a\sqrt{1 - \frac{b^2}{a^2}}\cos(A + \pi), a\sqrt{1 - \frac{b^2}{a^2}}\sin(A + \pi))$$

2.3 Shape of the elliptical orbit

The orbit of a moving object where only a single pulling force $F = \alpha/r^2$ is applied, is dictated *only* by two vectors, the current velocity $\vec{\nu}$ and the current location $\vec{r}$. The velocity vector represents both the speed (magnitude of the velocity vector) and the direction of motion. To compute the ellipse of the orbit, we first compute a, which is dictated only by the magnitude of velocity:

$$E = \frac{\nu^2}{2} - \frac{\mu}{r}$$
$$a = -\frac{\mu}{2E}$$

Here, E is the special orbital energy and these equations are known as *vis-visa*. The μ factor is the gravitational constant of the pulling object (the Sun) and here it is equal to GM where M is the mass of the Sun and G is the gravitational constant from Newton's law of universal gravitation:

$$F = G\frac{m_1 m_2}{r^2}$$

In order to compute the ellipse, we start at the eccentricity vector $\vec{e}$:

$$\vec{e} = \frac{(\nu^2 - \mu/r)\vec{r} - (\vec{r} \cdot \vec{\nu})\vec{\nu}}{\mu}$$

The eccentricity of the ellipse is $e = \vec{e}$ and we compute b through a and e. The product $\vec{r} \cdot \vec{\nu}$ is the dot product of the two vectors:

$$\vec{r} \cdot \vec{\nu} = x_r \cdot x_\nu + y_r \cdot y_\nu$$

The angle between the object ($\vec{r}$) and the axis connecting the foci and the center ($-\vec{c}$) is known as the true anomaly of the orbit and is computed as:

$$\theta = \cos^{-1} \frac{\vec{e} \cdot \vec{r}}{er}$$

and we substitute θ with $-\theta$ if $\vec{r} \cdot \vec{\nu} < 0$, assuming the $\cos^{-1}$ function returns angles in $[0, \pi)$. We cannot distinguish 2π angles by the cosine alone because $\cos\phi = \cos(-\phi) = \cos(2\pi - \phi)$. The rotation of the ellipse is:

$$A = \arg(x_r, y_r) - \theta$$

The $\arg(x, y)$ function is a generalization of $\tan^{-1}(y/x)$ that returns the direction of a vector (x, y) in all four quadrants. It is avaiable in most programming languages as `atan2(y,x)`. Now, we have all that we need to draw the orbit ellipse: a, b, and A. A circular orbit is a special case with $a = b$ (A is undefined).

Even if we know where the orbit, we don't know where exactly the object is, $\vec{r}(t)$, at every time t. We only know that $\vec{r}(0) = \vec{r}$ and $\vec{r}(t + kT) = \vec{r}(t)$ if T is the time for a full period. In elliptical orbits caused by inverse distance forces, unless the orbit is circular, the speed is not constant. The object will move faster when closer to the pulling object (here the Sun) and slower when it is far away. In order to find the orbital position at time t, we first compute the initial orbital "offset" at t_0 and then find the orbital position at $t \geq t_0$.

2.4 Orbital position $\vec{r}$ at time t

In order to find the position of the asteroid in time t, we use the fact that equal areas of orbit are covered in equal time intervals (Kepler's 2^{nd} law). The "angle" that increases linearly with t is known as the *mean anomaly* M. The angle of the point $\vec{r}$ with $\vec{f_1} - \vec{c}$ is known as the *eccentric anomaly* E. We compute the initial eccentric anomaly E_0 using the true anomaly θ and the eccentricity e:

$$E = \frac{e + \cos\theta}{1 + e\cos\theta}$$

and substitute with E_0 with $-E_0$ if $\vec{r} \cdot \vec{v} < 0$. We then compute the initial mean anomaly M_0 using the eccentric anomaly E_0 and the eccentricity e:

$$M = E - e\sin E$$

The orbital period T can be computed through a (Kepler's 3^{rd} law) as:

$$T = 2\pi\sqrt{\frac{a^3}{\mu}}$$

Here, the simulator does not treat time as a continuous dimension but as a discrete dimension where each time point is dt away from each other in real time. Each turn is at $t = k \cdot dt$ ($k \in \mathbb{N}$) and the number of turns per period is:

$$T_{dt} = \lceil \frac{T}{dt} \rceil$$

By definition, the mean anomaly M increases linearly with time and covers a 2π arc uniformly during a period. Using our approximation, we have T_{dt} turns per period and we split the 2π arc equally across T_{dt} points. To ensure consistency with the initial position, we also add the offset of the initial mean anomaly M_0:

$$M = M_0 + \frac{2\pi}{T_{dt}}t \quad , \quad \text{with} \quad t = 0, 1, 2, ..., T_{dt}, ..., 2T_{dt}, ...$$

In order to find the point, we need to find the eccentric anomaly E at time t. We showed how to calculate M from E but not vice versa. To solve this problem, we have to use an iterative method. The simulator uses the Newton-Raphson method that performs the following iteration until $f(x)$ is close enough to 0:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

To find E from M, we approximate the roots of $f(E)$ with derivative $f'(E)$:

$$\begin{aligned} f(E) &= E - e\sin E - M \\ f'(E) &= 1 - e\cos E \end{aligned}$$

but we still need a sensible starting x_0. From $f(E)$ we can see that $f(M-1) < 0$ and $f(M+1) > 0$, thus we search in the $(M-1, M+1)$ interval using $x_0 = M$.

If we know the eccentric angle E (at time t), we can now compute the $\vec{r_c}$ vector:

$$\vec{r_c} = (a \cos E, b \sin E)$$

which is the same logical vector as $\vec{r}$, but measured using $\vec{c}$ as the coordinate system center and by assuming that the focus is on the x axis (at $(0, ae)$). The actual location $\vec{r}$ is found by rotating the axes by A and by shifting the coordinate center from (x_c, y_c) to $(0, 0)$:

$$\hat{r} = \arg(x_{r_c}, y_{r_c}) + A$$
$$\vec{r} = (|\vec{r_c}| \cos \hat{r} + x_c, |\vec{r_c}| \sin \hat{r} + y_c)$$

Note that, while an orbit is calculated using the initial $\vec{r}$ and $\vec{\nu}$, the orbital position at $t = 0$ will not be identical to the original $\vec{r}$ due to precision errors.

2.5 Orbital velocity $\vec{\nu}$ at time t

In order to find the velocity of the asteroid at time t, we first use the *vis-visa* equation to find the length of the velocity vector with $\nu = |\vec{\nu}(t)|$ and $r = |\vec{r}(t)|$:

$$\nu^2 = \mu(\frac{2}{r} - \frac{1}{a})$$

In order to find the velocity direction we use the fact that it has to be tangent to the ellipse line. We first find the angle ϕ that is formed between the point $\vec{r}$ (that we showed how to compute) and the two foci. The distance from the first focus is r, from the second focus $(2a - r)$, and the distance between the foci is $2ae$, using the basic properties of any ellipse:

$$|\vec{r} - \vec{f_1}| + |\vec{r} - \vec{f_2}| = 2a \quad , \quad |\vec{f_1} - \vec{f_2}| = 2ae$$

We can compute ϕ using the law of cosines in this triangle:

$$\phi = \cos^{-1} \frac{r^2 + (2a - r)^2 - (2ae)^2}{2r(2a - r)}$$

The velocity direction is the tangent of the ellipse curve at $\vec{r}$. We use the fact that the angle formed by the points $\vec{f_1}$, $\vec{r}$, and $\vec{f_2}$ (say ϕ) is exactly orthogonal to the velocity direction, i.e. the perpendicular line of $\vec{\nu}$ at $\vec{r}$ splits ϕ in two equal halves. Thus, the two side angles of ϕ on the line formed by $\vec{\nu}$ are $(\pi - \phi)/2$ each. The angle that we add to $\hat{r}$ to get $\hat{\nu}$ is vertical to ϕ's side angles, thus:

$$\hat{\nu} = \arg(x_r, y_r) + \frac{\pi - \phi}{2}$$

After calculating both $\hat{\nu}$ and $|\vec{\nu}|$, we can compute the actual vector ν by translating the polar to Cartesian coordinates:

$$\vec{\nu}(t) = (|\vec{\nu}| \cos \hat{\nu}, |\vec{\nu}| \sin \hat{\nu})$$

As with $\vec{r}$, the orbital velocity at $t = 0$ will not be identical to the original ν that created the orbit due to arithmetic and precision errors.

3 Collisions & pushes

3.1 Asteroid collision

When a collision between two planets with masses m_1 and m_2 and velocities $\vec{\nu}_1$ and $\vec{\nu}_2$ occurs, the velocity of the combined asteroid after the collision is:

$$\vec{\nu} = \frac{m_1\vec{\nu}_1 + m_2\vec{\nu}_2}{m_1 + m_2}$$

and the mass is $m_1 + m_2$. The initial location $\vec{r}$ of the new asteroid is assumed here to be the barycenter of the colliding asteroids at the time t of collision:

$$\vec{r} = \frac{m_1\vec{r_1}(t) + m_2\vec{r_2}(t)}{m_1 + m_2}$$

and is used alongside the initial velocity $\vec{\nu}$ to compute the new orbit. We compute the volume V of the asteroid using the density ρ, which is assumed here to be the same for all asteroids, and the mass m. Using V, we can compute the asteroid radius R by assuming that each asteroid is a 3D sphere of:

$$V = \frac{4}{3}\pi R^3 = \frac{m}{\rho}$$

In the context of this game, two asteroids collide if, at some time $t = k \cdot dt$ ($k \in \mathbb{N}$), their distance is less than the sum of their radii:

$$|\vec{r}_1(t) - \vec{r}_2(t)| < R_1 + R_2$$

Note that the simulator checks if the asteroids are close enough at each $t = k \cdot dt$, not at every t. If asteroid X collides with asteroid Y and asteroid Y collides with asteroid Z (X may also collide with Z but it does not need to), then we have a collision of 3 asteroids. More formally, in a collision of $k-1$ asteroids, the k-th asteroid collides if it is close enough to *any* one of the $k-1$ asteroids. The above equations are generalized for k colliding asteroids as:

$$\vec{\nu} = \frac{\sum_{i=1}^{k} m_i\vec{\nu}_i}{\sum_{i=1}^{k} m_i} \ , \quad \vec{r} = \frac{\sum_{i=1}^{k} m_i\vec{r}_i}{\sum_{i=1}^{k} m_i} \ , \quad m = \sum_{i=1}^{k} m_i \ , \quad \text{with} \quad i = 1, 2, ..., k$$

3.2 Asteroid push

A push is specified by setting the energy of the push E and the direction of the push ϕ. The push is essentially an instantaneous acceleration that creates a velocity $\vec{\nu}_p$ with $\hat{\nu}_p = \phi$. The magnitude of $\vec{\nu}_p$ is computed by assuming that the push energy E is instantaneously converted into kinetic energy:

$$\vec{\nu}_p = (\sqrt{\frac{2E}{m}}\cos\phi, \sqrt{\frac{2E}{m}}\sin\phi)$$

The push velocity $\vec{\nu}_p$ is then added to the orbital velocity $\vec{\nu}_o$ computed at the time of the push. A new orbit is created using the new velocity $\vec{\nu} = \vec{\nu}_p + \vec{\nu}_o$ and the location $\vec{r}$ at the time of the push. If the push creates a new orbit that is not periodic, a "flyby" orbit with $e \geq 1$, the simulator will not permit it.

4 Simulator & default player

4.1 Orbit

When the simulator computes an orbit, it only calculates (and stores) a, b, A, and M_0. The orbit calculation takes $O(1)$ time. Orbital positions and velocities are calculated dynamically by the corresponding methods using a, b, A, M_0, and the time t, which is given as an integer k representing $t = k \cdot dt$. The `positionAt(k)` method returns the position at $t = k \cdot dt$ and the `velocityAt(k)` method returns the velocity at $t = k \cdot dt$. These functions also take $O(1)$ amortized time. The `period()` method returns the number of dt turns per period (an integer) and the `center()` returns the center of the ellipse. The parameters a, b, A, and M_0 are also public in the orbit class `Orbit`, which is immutable. Circular orbits are created by giving only the initial position $\vec{r}$. Elliptical orbits are created by giving both the initial position $\vec{r}$ and the velocity $\vec{v}$. The vectors are represented as points using the `Point` class. The turn time dt is also available in the `Orbit` class as a static variable. The `Orbit` class can be iterated to return all the T_{dt} orbital locations within an orbital period.

4.2 Asteroid

An asteroid is an object that has an orbit, a mass, and a epoch t_0 at which the orbit was created. To find where an asteroid is at $t = k \cdot dt$, you must subtract the time that the orbit was created by calling `positionAt(t-to)`. When `push(t)` is called on an Asteroid object, a new Asteroid object with a new orbit and epoch `t` is created. This way, you can "predict" when two asteroids will collide and "simulate" the future, without actually returning your actions to the simulator and without advancing the actual game time. By calling the `test_collision(a, k)` method on an array of asteroid objects `a`, you can test which sets of asteroids collide at $t = k \cdot dt$. The `test_collision` method calls the `overlaps` static method of `Point` that tests for circle overlaps given the centers and the radii. When `force_collision(a, k)` is called, a new asteroid will be formed as if the asteroids were colliding even if they are not close enough. The mass of the asteroid is public variable and the `radius()` method returns the radius of the `Asteroid` class. The density is a public static variable. Using the `test_collision` method is more expensive in terms of memory, because `Point` objects are allocated every time. If the player tests collisions between a pair of planets at a time, the `Orbit` methods should suffice.

4.3 Memory

It is a good idea to avoid generating new points if a very large number of orbital positions is consindered. The Java heap space can easily overflow and the Java Virtual Machine (JVM) will throw an exception. To contain memory use, `Point` objects can be passed as parameters to the `Orbit` methods and their contents will be modified. The `Point` class is intentionally left mutable for this purpose.

4.4 Default player

The default player `g0` tries to push a random asteroid by a random angle and energy. Then, by simulating the "future", the player finds if the asteroid will collide in the next `X` years. If yes, then the push is returned to the simulator and no further action is done until the time that the asteroid actually collides. The collision is tested using a pair of orbits at a time using the `Orbit` methods to calculate the positions. The player reuses `Point` objects when requesting orbital positions to avoid overflowing the heap space memory of the JVM.

4.5 Simulator program

The simulator is written using java 1.8; earlier versions are not supported. To run the simulator from a terminal, type: `java pb.sim.Simulator <arguments>` The simulator must be run from the parent directory of `pb`. The player source code is (re)compiled automatically. The command line arguments are:

- `-g, --group <g>`: group `g` plays the game (default: `g0`)
- `-a, --asteroids <a>`: randomly generate `n` asteroids in circular orbits
- `-s, --state <s>`: load (or store if `--asteroids` is set) state from file `s`
- `-t, --time <t>`: set the time limit (in *dt* units) to `t` (default: no limit)
- `--density <d>`: set the density of the asteroid ($\times$ the Sun density)
- `--orbit-range <min> <max>`: distance range for random orbits ($\times 10^9$)
- `--mass-range <min> <max>`: mass of random asteroids ($\times$ the Sun mass)
- `--init-timeout <t>`: set a timeout for the `init()` method (in ms)
- `--play-timeout <t>`: set a timeout for the `play()` method (in ms)
- `--gui`: enable the GUI (also enabled by all other `--gui-*` arguments)
- `--gui-fps <fps>`: set the frames per second of the GUI (default: 50)
- `--gui-fast-forward`: allow GUI to fast forward turns automatically
- `--gui-no-planets`: do not show solar system planets on the GUI

4.6 Simulator GUI

The GUI uses a web browser to render an HTML page where the asteroids and their orbits are drawn on HTML 5 canvas using javascript. On each turn, the representation is automatically scaled to fit the screen size. State updates are provided through AJAX requests. The web browser should then be opened at `http://localhost:<port>`. The `port` is printed on the terminal screen by the simulator if the GUI is enabled. The GUI should work on all latest browsers, such as Mozilla Firefox, Google Chrome, Internet Explorer (including Microsoft Edge), Safari, and Opera. Note that javascript must be enabled on the browser.