Introduction to Programming

— Practical #4 —

1 Structures

Warning : In this practical, we will have bodies evolve under gravitation and have elastic shocks. It is a long practical that will take several sessions. Actually, the next practical is a reorganization of this one. After question 1.1.4 the remainder will be considered optional and can be coded after reorganization in the next practical. In section 1.2 some functions to use are presented, and in 1.3 their physics justification.

1.1 Steps

1.1.1 Translation motion

1. *To begin, study the project :*
   Download `Tp4_Initial.zip`, decompress it and launch Qt Creator. Input your name in the placeholder of file `gravitation.cpp`. Browse the project, look out for global variables and the function main (do not waste time with the contents of functions already defined but not used yet). The program lets a point (`x`, `y`) evolve according to a constant translation (`vx`, `vy`), and displays regularly a disk centered at this point. For that, to erase it, we store the position of the disk at the last display (in `ox` and `oy`) ; besides, two instructions beginning with `noRefresh` nest the graphics instructions to have a smoother display without flicker.
2. *Use a structure :*
   Modify the program so as to use a structure Ball keeping all information about the disk (position, speed, radius, color).
3. *Display :*
   Implement (and use !) a function void DisplayBall(Ball D) printing the disk D, and another void EraseBall(Ball D) that erases it.
4. *Make the disk move properly :*
   To have the position of the disk evolve, replace the corresponding instructions already present in the main by a call to a function modifying the coordinates of a Balle, by adding the speed of the `Balle` multiplied by a certain time step defined as a global variable ($dt = 1$ for now).

1.1.2 Gravitation

5. *Evolution with acceleration :*
   Create (and use) a function that modifies the speed of a `Ball` so that it undergoes a constant acceleration $a_x = 0$ and $a_y = 0.0005$. Hint : proceed as before, that is add $0.0005 * dt$ to v_y...
6. *Add a sun :*
   We wish not to have a uniform gravitation. Add a field storing the mass to the structure `Ball`. Create a sun (of type `Ball`), yellow, fixed (null speed) at the window center, of mass 10 and radius 4 pixels (the mass of the moving planet is 1). Display it.
7. *Gravitational acceleration :*
   Create (and use instead of the uniform gravitation) a function that takes as argument the planet and the sun, and that makes the speed of the planet evolve. Reminder in physics : remember since Newton that the acceleration is $-G\,m_S/r^3\ \overrightarrow{r}$, here $G = 1$ (meaning our mass is not expressed in kg but in another unit, which does not matter). You may need the square root function double sqrt(double x). Do not forget the factor dt... Let it run and observe. Try different initializations of the planet (such as $x =$ `width`$/2$, $y =$ `height`$/3$, $v_x = 1$, $v_y = 0$). Note that the expression of acceleration becomes huge when r is near 0 ; we will therefore apply no acceleration when r is too small.

8. *Random initialization :*
   Create (and use instead of initial conditions given for the sun) a function initializing a `Ball`, with position in the window, null speed, radius between 5 and 15, and mass being radius divided by 20 (beware not to use Euclidean division). You will need the function Random...
9. *Suns galore...*
   Place 10 suns randomly (and take them into account in the computation of the motion for the asteroid...).
10. *Tune the time step of the computation :*
    To avoid errors due to discretization of time, decrease the time step dt, such as 0.01 (or even 0.001 if the machine is powerful enough). Tune the display frequency accordingly (inversely proportional to dt). Launch several times the program.

1.1.3 Simple elastic shocks

11. *Make the asteroid bounce :*
    Have the asteroid undergo elastic shocks each time it goes too close to a sun, so that it never goes inside it (function ShockSimple), and put back dt to a higher value, such as 0.1 (modify the display frequency accordingly). To know if two spherical bodies will undergo a shock, use the function Collision. Do not modify the given functions `Shock`, `ShockSimple` and `Collision`. Instead, you will create new ones taking as argument `Ball` (with the same name, it is fine as long as the function arguments are different, it is called function overload), they call the correponding initial function.

1.1.4 Shooting game

(figure 1 right)

12. *Create a new project :*
    To be more precise, we will add a second executable program to our project. Create a file `Duel.cpp` and containing a copy of `Tp4.cpp` initially. Modify then the file `CMakeLists.txt` to append two lines indicating that the executable `Duel` depends on `Duel.cpp` and uses the `Graphics` library of Imagine++.
13. *Your turn !*
    Transform the project `Duel`, with the help of the functions already defined, in a shooting game with two players. Each player has a fixed position, and various suns are placed randomly on screen. Each player in turn can launch a `Ball` with the chosen initial speed, the ball undergoing the gravitation of the different suns and disappearing after 250 display time steps. The winner is the first one that launches the ball to the other one... Practical advice : place the players symmetrically with respect to the center, at mid-height leaving a margin of one eighth of the width ; use the function GetMouse to know the position of the mouse at a click ; deduce the chosen speed by subtracting from these coordinates the ones of the ball center to launch and multiplying by a factor 0.00025.
14. *Improvements :*
    Make sur there is a large sun at the screen center (but its mass may not be big) to prevent direct hits.
15. *Correct initialization :*
    Modify the function putting suns so that they do not intersect initially, and that they are at least at distance 100 pixels from the players.

1.1.5 Elastic shocks

(figure 1 left)

16. *Everything moving, everything bouncing :*
    Get back to project `Gravitation`. Have everything move, including suns. Use for elastic shocks the function Schock (that makes the two bodies bounce).

1.2 Help

Given functions :

```
void initRandom();
```

(in Imagine++) is to be executed before the first call to random (and only once).

```
double doubleRandom();
```

(in Imagine++) returns a double drawn randomly between 0 and 1.

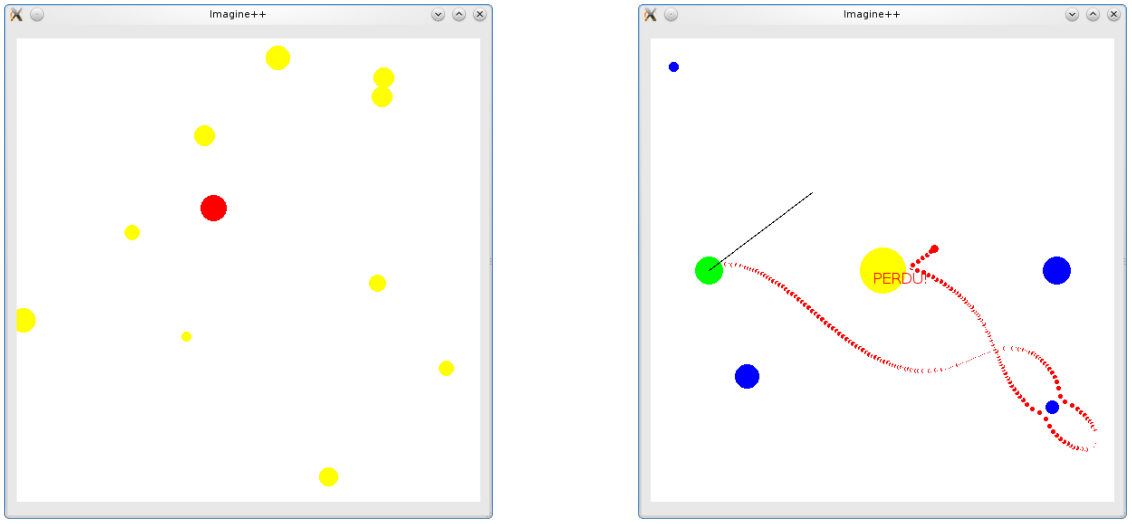

FIGURE 1 – Celestial bodies and shooting game...

```
void shockSimple(double x, double y, double &vx,double &vy,double m,
                 double x2,double y2,double vx2,double vy2);
```

makes the first particle bounce, with coordinates (`x`, `y`), speed (`vx`, `vy`) and mass `m`, on the second one, with coordinates (`x2`, `y2`) and speed (`vx2`, `vy2`), without moving the first one.

```
void shock(double x, double y, double &vx, double &vy, double m,
           double x2,double y2,double &vx2,double &vy2,double m2);
```

makes *both* particles bounce.

```
bool collision(double x1,double y1,double vx1,double vy1,double r1,
               double x2,double y2,double vx2,double vy2,double r2);
```

returns true if the body at (`x1`, `y1`), with speed (`vx1`, `vy1`) and with radius `r1` is about to collide with the body at (`x2`, `y2`), with speed (`vx2`, `vy2`) and radius `r2`, and false otherwise.

1.3 Physics

Remark : this section is here only to explain the contents of the function already programmed. You can skip it if not interested.

1.3.1 Acceleration

The sum of forces applying on a body A is equal to the product of its mass by the acceleration of its center gravity.

$$\sum_i \overrightarrow{F}_{i/A} = m_A \overrightarrow{a}_{G(A)}$$

1.3.2 Universal gravitation

Given two bodies A and B, A undergoes an attraction force from B :

$$\overrightarrow{F}_{B/A} = -Gm_A m_B \frac{1}{d_{A,B}^2} \overrightarrow{u}_{B\to A}.$$

1.3.3 Elastic shocks

Let A and B two particles involed in a collision. Knowing all parameters before the shock, how to determine their values after ? Actually, only the speed of the particles need to be updated, since at the shock moment, no position is changed.

During a shock said *elastic*, three quantitites are preserved :

1. momentum $\overrightarrow{P} = m_A \overrightarrow{v}_A + m_B \overrightarrow{v}_B$
2. cinetic moment $M = m_A \overrightarrow{r}_A \times \overrightarrow{v}_A + m_B \overrightarrow{r}_B \times \overrightarrow{v}_B$ (a real number in the case of plane motion).
3. cinetic energy $E_c = \frac{1}{2} m_A v_A^2 + \frac{1}{2} m_B v_B^2$.

Therefore, we get 4 equations for 4 unknowns.

1.3.4 Shock resolution

We move into the corrdinate frame of the center of mass. We get at each time :

1. $\overrightarrow{P} = 0$ (by definition of this frame), hence $m_A \overrightarrow{v}_A = -m_B \overrightarrow{v}_B$.
2. $M = (\overrightarrow{r}_A - \overrightarrow{r}_B) \times m_A \overrightarrow{v}_A$, hence, noting $\Delta\overrightarrow{r} = \overrightarrow{r}_A - \overrightarrow{r}_B$, $M = \Delta\overrightarrow{r} \times m_A \overrightarrow{v}_A$.
3. $2E_c = m_A(1 + \frac{m_A}{m_B})v_A^2$.

The constancy of E_c indicates that in this frame, the norm of speed is preserved, and the constancy of the cinetic moment that the speeds vary parallely to $\Delta\overrightarrow{r}$. Apart from the initial speeds, there is a unique possibility left : multiply by -1 the component of $\overrightarrow{v}_i$ along $\Delta\overrightarrow{r}$. That leads to a simple algorithm for the shock.

1.3.5 Deciding if a shock is going to happen

We won't be happy, along the step by step evolution of the coordinates of the disks, while deciding if a shock will happen between t and $t + dt$, with jsut estimating the distance between the two candidates to collision at time t, not even by taking in consideration this distance at $t + dt$, because, if the speed is too high, one disk may already have gone through the other and gone out at $t + dt$... The best solution is to explicitly compute the minimum distance between the disks along the time, between t and $t + dt$.

Let $N(u) = (\overrightarrow{r}_A(u) - \overrightarrow{r}_B(u))^2$ be the square of the distance. We get :

$$N(u) = (\overrightarrow{r}_A(t) - \overrightarrow{r}_B(t) + (u - t)(\overrightarrow{v}_A(t) - \overrightarrow{v}_B(t)))^2$$

This gives, with supplementary notations :

$$N(u) = \Delta\overrightarrow{r}(t)^2 + 2(u - t)\Delta\overrightarrow{r}(t) \cdot \Delta\overrightarrow{v}(t) + (u - t)^2\Delta\overrightarrow{v}(t)^2$$

The norm, always positive, is minimal at point u so that $\partial_u N(u) = 0$, hence :

$$(t_m - t) = -\frac{\Delta\overrightarrow{r}(t) \cdot \Delta\overrightarrow{v}(t)}{\Delta\overrightarrow{v}(t)^2}$$

Thus :

1. if $t_m < t$, the minimum is reached at t,
2. if $t < t_m < t + dt$, the minimum is reached at t_m ;
3. otherwise, $t + dt < t_m$, the minimum is reached at $t + dt$.

That gives easily and explicitly the smallest distance between the bodies between times instants t and $t + dt$.