Introduction to Programming

— Practical #5 —

1 Separate files

We are going to finish in this practical the simulations of gravitation and elastic shocks. We will concentrate on code presentation and separate the code into different files.

1. *Good bases :*
   Take again the project of last week and check that it compiles again. If you have reached question 7 included, keep a backup copy to refer to it if by refactoring the code during this practical you break everything.

1.1 Tools functions

2. *A file of definitions...*
   Add a new source file named `tools.cpp` to the project. Put there the initial functions of TP4 (InitRandom, Random, Shock, ShockSimple and Collision), removing them from `Gravitation.cpp`. Do not forget the follwing lines, that we can then remove from `Gravitation.cpp` :

   ```
   #include <cstdlib>
   #include <ctime>
   using namespace std;
   ```

   Make sure is file is created in the "source directory" (where other source code files live). Do not forget to modify the arguments of `add_executable` in the `CMakeLists.txt` to add `tools.cpp` (source files are separated just by a blank space). Qt Creator should note the change when you save the file and should relaunch CMake, as you should observe in the tab "General Messages". If not, relaunch CMake through the menus.
3. *... and a declaration file*
   Add a header file `tools.h`. Insert the protection against double inclusion with #pragma once, instead of the other preprocessor instructions (beginning in #) put there by Qt Creator. Place the declarations of functions of `tools.cpp`, so as the definition of `dt`, removing it from `main`. Add at the beginning of `tools.cpp` and `Gravitation.cpp` the inclusion directive

   ```
   #include "tools.h"
   ```

1.2 Vectors

4. *Structure Vector :*
   Create in a new file `vector.h` a structure representing a plane vector, with two members of type double. Do not forget the mechanism against double inclusion. Declare (but do not define yet) the operators and functions that follow :

   ```
   Vector operator+(Vector a,Vector b);        // Sum
   Vector operator-(Vector a,Vector b);        // Difference
   double norm2(Vector a);                     // Euclidean norm
   Vector operator*(Vector a,double lambda);   // Mult. scalar
   Vector operator*(double lambda,Vector a);   // Mult. scalar
   ```

5. *Functions and operators with* `Vector` *:*
   Create a new file `vector.cpp`. Put the #include for the header file and define the operators declared there (Reminder : `sqrt` is provided by the header file `<cmath>`; don't forget either using namespace std; allowing to use directly its name : once a version of operator∗ is defined, the second one can use the first in its definition...[1]

1. It is elegant to do like that, it makes sure the two calls are consistent.

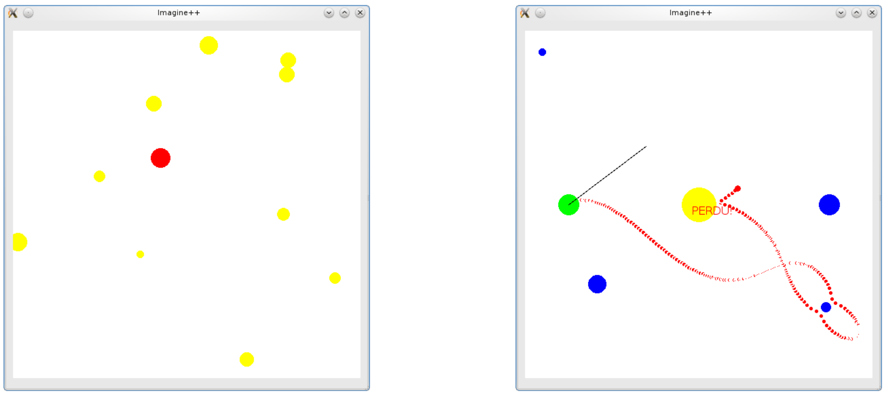

FIGURE 1 – Celestial bodies and shooting game...

6. *Vector speed and vector position :*
   Replace in `Gravitation.cpp` the speeds and positions by structures of type `Vector` (also in the definition of structure `Ball`). Use as much as possible the operators and functions defined of `vector.h`.

1.3 Putting Ball apart

7. *Structure* `Ball` *:*
   Move the definition of structure `Ball` in its own header file `Ball.h`. Since `Ball` uses types `Vector` and `Color`, you need to add the lines :

```
#include <Imagine/Graphics.h>
using namespace Imagine;

#include "vector.h"
```

8. *Associated functions :*
   Move all functions taking some `Ball` as argument to a new file `Ball.cpp`. At this point, in `Gravitation.cpp` the only function left should be main. Declare in `Balle.h` the functions defined in `Ball.cpp`. Add the #include necessary in this file and in `Gravitation.cpp` and perform necessary adaptations (for instance, if some functions use `width` or `height`, since these constants are defined only in `Gravitation.cpp`, they need to take them as argument...)

1.4 Back to physics

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.

1.4.1 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.

1.4.2 Shooting game

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. *Do not make twice the effort :* Since we need many functions from project Gravitation, add to the project (without creating new files !) the files
    `vector.h`, `vector.cpp`, `Ball.h`, `Ball.cpp`, `tools.h`, `tools.cpp`
    The files are *the same* as in project Gravitation, they are not copies. Include at the beginning of `Duel.cpp` the #include necessary. Try to compile `Duel.cpp`.
14. *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.
15. *Improvements :*
    Make sur there is a large sun at the screen center (but its mass may not be big) to prevent direct hits.
16. *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.

1.4.3 Elastic shocks

17. *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).