Algorithmique et Programmation

G1: monasse*(at)*imagine.enpc.fr G2: nicolas.audebert*(at)*onera.fr
G3: alexandre.boulch*(at)*onera.fr G4: maxime.ferrera*(at)*onera.fr
G5: laurent.bulteau*(at)*u-pem.fr G6: pierre-alain.langlois*(at)*eleves.enpc.fr

— TP #5 —

1 Fichiers séparés

Nous allons poursuivre dans ce TP les simulations de gravitation et de chocs élastiques entamées la semaine dernière, en séparant dans différents fichiers les différentes fonctions et structures utilisées.

1. *De bonnes bases :*
   Reprenez le travail en cours de la semaine derniére. Si vous avez été au bout de celui-ci ou au moins jusqu'à la question 7 incluse, faites-en une sauvegarde séparée pour pouvoir vous y reporter si les travaux entrepris dans ce TP ont tendance à tout casser.

1.1 Fonctions outils

2. *Un fichier de définitions...*
   Ajouter un nouveau fichier source nommé `Tools.cpp` au projet. Y placer les fonctions fournies à l'avance au début du TP4 (InitRandom, Random, Choc, ChocSimple et Collision), en les retirant de `Gravitation.cpp`. Ne pas oublier les lignes suivantes, que l'on pourra retirer de `Gravitation.cpp` :

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

   Attention de ne pas placer le nouveau fichier dans le "build directory" de CMake. Vous devez bien placer ce fichier dans le dossier des sources (le même que Gravitation.cpp). N'oubliez pas de modifier les arguments de `add_executable` du `CMakeLists.txt` pour y ajouter `Tools.cpp`. Il faut alors relancer Cmake. Sous Visual Studio, il devrait constater le changement du `CMakeLists.txt` et vous proposer de recharger le projet. Acceptez sa proposition, il relance Cmake en coulisses. Sous QtCreator, relancez Cmake (depuis QtCreator).
3. *... et un fichier de déclarations*
   Ajouter un nouveau fichier d'en-tête nommé `Tools.h`. Inclure la protection contre la double inclusion vue en cours (`#pragma once`). Y placer les déclarations des fonctions mises dans `Tools.cpp`, ainsi que la définition de `dt`, en retirant celle-ci de `main`. Rajouter au début de `Tools.cpp` et de `Gravitation.cpp` un

   ```
   #include "Tools.h"
   ```

1.2 Vecteurs

4. *Structure Vector :*
   Créer dans un nouveau fichier `Vector.h` une structure représentant un vecteur du plan, avec deux membres de type double. Ne pas oublier le mécanisme de protection contre la double inclusion. Déclarer (et non définir) les opérateurs et fonction suivants :

   ```
   Vector operator+(Vector a, Vector b);       // Somme
   Vector operator-(Vector a, Vector b);       // Différence
   double norme2(Vector a);                    // Norme euclidienne
   Vector operator*(Vector a, double lambda); // Mult. scalaire
   Vector operator*(double lambda, Vector a); // Mult. scalaire
   ```

5. *Fonctions et opérateurs sur les `Vector` :*
Créer un nouveau fichier `Vector.cpp`. Mettre un #include du fichier d'en-tête correspondant et définir les opérateurs qui y sont déclarés (Rappel : `sqrt` est défini dans le fichier d'en-tête système `<cmath>`; ne pas oublier non plus le using namespace std; qui permet d'utiliser cette fonction). Astuce : une fois qu'une version de operator∗ est définie, la deuxième version peut utiliser la première dans sa définition...
6. *Vecteur vitesse et vecteur position :*
Systématiquement remplacer dans `Gravitation.cpp` les vitesses et positions par des objets de type `Vector` (y compris dans la définition de la structure `Balle`). Utiliser autant que possible les opérateurs et fonction définis dans `Vector.cpp`.

1.3 Balle à part

7. *Structure `Balle` :*
Déplacer la structure `Balle` dans un nouveau fichier d'en-tête `Balle.h`. Puisque `Balle` utilise les types `Vector` et `Color`, il faut aussi ajouter ces lignes :

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

#include "Vector.h"
```

8. *Fonctions associées :*
Déplacer toutes les fonctions annexes prenant des `Balle` en paramètres dans un nouveau fichier `Balle.cpp`. Il ne devrait plus rester dans `Gravitation.cpp` d'autre fonction que main. Déclarer dans `Balle.h` les fonctions définies dans `Balle.cpp`. Ajouter les #include nécessaires dans ce dernier fichier et dans Gravitation.cpp et faire les adaptations nécessaires (par exemple, si des fonctions utilisent `largeur` ou `hauteur`, comme ces constantes ne sont définies que dans `Gravitation.cpp`, il faut les passer en argument...)

1.4 Retour à la physique

9. *Des soleils par milliers... :*
Placer 10 soleils aléatoirement (et en tenir compte à l'affichage, dans le calcul du déplacement de l'astéroïde...).
10. *Diminuer le pas de temps de calcul :*
Afin d'éviter les erreurs dues à la discrétisation du temps, diminuer le pas de temps dt, pour le fixer à 0.01 (voire à 0.001 si la machine est assez puissante). Régler la fréquence d'affichage en conséquence (inversement proportionnelle à dt). Lancer plusieurs fois le programme.

1.4.1 Chocs élastiques simples

11. *Faire rebondir l'astéroïde :*
Faire subir des chocs élastiques à l'astéroïde à chaque fois qu'il s'approche trop d'un soleil, de façon à ce qu'il ne rentre plus dedans (fonction `ChocSimple`), et rétablir dt à une valeur plus élevée, par exemple 0.1 (modifier la fréquence d'affichage en conséquent). Pour savoir si deux corps sont sur le point d'entrer en collision, utiliser la fonction `Collision`.

1.4.2 Jeu de tir

12. *Ouvrir un nouveau projet :*
Afin de partir dans deux voies différentes et travailler proprement, ajouter un nouveau projet appelé `Duel`, dans cette même solution. On ajoute un dossier `Duel` au même niveau que `Gravitation` et on modifie le `CMakeLists.txt`.
13. *Ne pas refaire deux fois le travail :* Comme nous aurons besoins des mêmes fonctions dans ce projet que dans le projet Gravitation, ajouter au projet (sans en créer de nouveaux!) les fichiers `Vector.h`, `Vector.cpp`, `Balle.h`, `Balle.cpp`, `Tools.h`, `Tools.cpp`. Les fichiers sont les *mêmes* que dans le projet Gravitation, ils ne sont pas recopiés. Mettre au début de `Duel.cpp` (fichier à placer dans le répertoire `Duel`) les #include correspondants. Essayer de compiler `Duel.cpp`. Comme le compilateur n'arrive pas à trouver les fichiers inclus, qui ne sont pas dans le même répertoire, il faut lui indiquer où les trouver :

```
#include "../Gravitation/Tools.h"
```

Pour le `CMakeLists.txt` du répertoire Duel, inspirez-vous de celui de Gravitation.

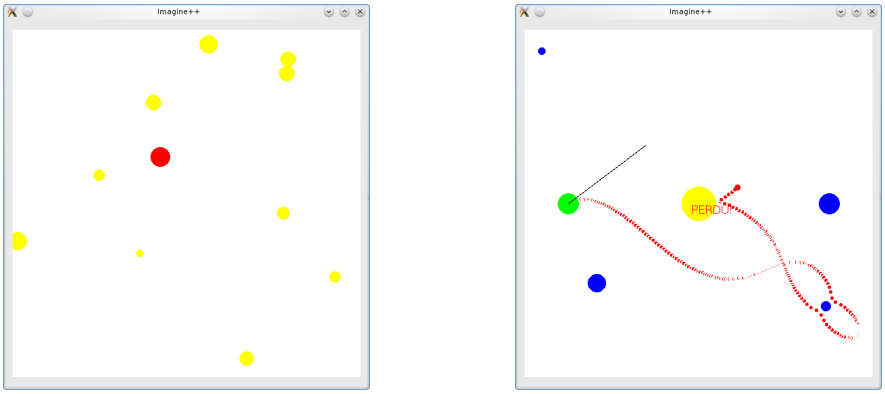

FIGURE 1 – Corps célestes et jeu de tir...

14. *À vous de jouer !*
Transformer le projet `Duel`, à l'aide des fonctions définies auparavant, en un jeu de tir, à deux joueurs. Chacun des deux joueurs a une position fixée, et divers soleils sont placés aléatoirement dans l'écran. Chaque joueur, à tour de rôle, peut lancer une `Balle` avec la vitesse initiale de son choix, la balle subissant les effets de gravitation des divers soleils, et disparaissant au bout de 250 pas de temps d'affichage. Le gagnant est le premier qui réussit à atteindre l'autre... Conseils pratiques : positionner symétriquement les joueurs par rapport au centre, de préférence à mi-hauteur en laissant une marge d'un huitième de la largeur sur le côté ; utiliser la fonction `GetMouse` pour connaître la position de la souris ; en déduire la vitesse désirée par le joueur en retranchant à ces coordonnées celles du centre de la boule à lancer, et en multipliant par un facteur 0.00025.
15. *Améliorations :*
Faire en sorte qu'il y ait systématiquement un gros soleil au centre de l'écran (de masse non nécessairement conséquente) afin d'empêcher les tirs directs.
16. *Initialisation correcte :*
Modifier la fonction de placement des soleils de façon à ce que les soleils ne s'intersectent pas initialement, et qu'ils soient à une distance minimale de 100 pixels des emplacements des joueurs.

1.4.3 Chocs élastiques

17. *Tout faire évoluer, tout faire rebondir :*
On retourne dans le projet `Gravitation`. Tout faire bouger, y compris les soleils. Utiliser, pour les chocs élastiques, la fonction `Chocs` (qui fait rebondir les deux corps). Faire en sorte que lors de l'initialisation les soleils ne s'intersectent pas.