UPMC – Licence informatique 2ème année
Fonctions et procédures de calcul – 2I008

Projet – Moteur physique

Présentation générale du projet

Ce projet a pour but d'implémenter un moteur physique. Nous allons manipuler un monde (`world`) dans lequel se trouveront des corps (`body`). Ces corps seront soumis à des forces (`force`) qui feront se mouvoir les corps.

Étant donné, à un instant de temps, un ensemble de corps soumis à un ensemble de forces, il n'est pas toujours possible de calculer de manière exacte les positions des corps à tout instant dans le futur (e.g. : `https://en.wikipedia.org/wiki/Three-body_problem`)

Pour cette raison, notre moteur physique calculera une approximation des solutions des équations qui régissent le mouvement des corps. Pour des raisons de simplicité, nous nous placerons uniquement en 2D et dans le cas où nos corps sont des points massiques. Étant donné un tel corps b, de masse m_b, de position $P_b = (x_b, y_b)$, de vitesse $\overrightarrow{v_b} = (v_{b,x}, v_{b,y})$, d'accélération $\overrightarrow{a_b} = (a_{b,x}, a_{b,y})$, soumis à un ensemble de force $\mathfrak{F}$, le principe fondamental de la dynamique nous donne que :

$$m_b \overrightarrow{a_b} = \sum_{\overrightarrow{\in}\mathfrak{F}} \overrightarrow{F}$$

On rappelle de plus que la vitesse et l'accélération d'un corps b sont tels que :

$$\overrightarrow{v_b} = \frac{dP_b}{dt} \tag{1}$$

$$\overrightarrow{a_b} = \frac{d\overrightarrow{v_b}}{dt} \tag{2}$$

Ainsi nous pouvons (peut-on vraiment?) faire les deux approximation suivante (où $dt \in \mathbb{R}$):

- Si un corps a un accélération $\overrightarrow{a_b}(t)$ et un vitesse $\overrightarrow{v_b}(t)$ à un instant t alors à l'instant $t + dt$ sa vitesse sera : $\overrightarrow{v_b}(t + dt) = \overrightarrow{v_b}(t) + dt\overrightarrow{a_b}(t)$ (on remarquera habilement que ceci se réécrit sous la forme $\frac{\overrightarrow{v_b}(t+dt)-\overrightarrow{v_b}(t)}{dt} = \overrightarrow{a_b}(t)$ rappellant la forme de l'équation 2)
- Si un corps a une vitesse $\overrightarrow{v_b}(t)$ et une position $P_b(t)$ à un instant t alors à l'instant $t+dt$ sa postion sera : $P_b(t + dt) = P_b(t) + dt\overrightarrow{v_b}(t)$ (on remarquera encore habilement que ceci se réécrit sous la forme $\frac{P_b(t+dt)-P_b(t)}{dt} = \overrightarrow{v_b}(t)$ rappellant la forme de l'équation 1)

Cette méthode d'approximation est connue sous le nom de la méthode d'Euler (`https://en.wikipedia.org/wiki/Euler_method`). Elle consiste à approximer une fonction par une suite de droite dont les coefficients directeur sont les dérivées de la fonction à trouver.

Armé ainsi du principe fondamental de la dynamique (qui nous donnera à chaque instant l'accélération d'un point) et des deux approximations ci-avant mentionnées, nous sommes en mesure de calculer une approximation des trajectoires d'un ensemble de points soumis à un ensemble de force.

Architecture du projet

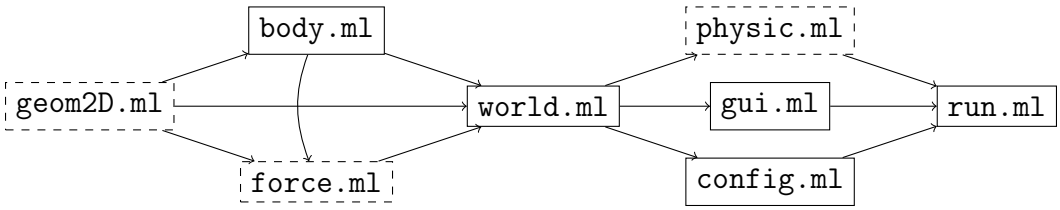

Figure 1: Architecture du projet. Les fichiers en pointillé sont les parties que vous aurez à modifier au cours du projet

`geom2D.ml` : le fichier implémenté au TP9 contenant les définitions du module `Vector` et `Point`.

`body.ml` : contient le type d'un corps.

`force.ml` : contient une définition (comme un type somme) de l'ensemble possible des forces pouvant s'éxercer sur nos corps. Ces défintions seront étendues dans les prochaines séances.

`world.ml` : contient la définition d'un monde. Cette défintion sera étendu dans les prochaines séances.

`physic.ml` : contient le moteur physique, décomposé en 3 principales fonctions qui doivent être implémentées pendant cette séance.

`gui.ml` : le module graphique.

`config.ml` : contient le type des configurations devant être fournies pour qu'un run soit effectué.

`run.ml` : contient la boucle d'évenement et l'exemple à lancer par le programme.

Implantation

Nous allons, dans un premier temps, nous intéresser à la modélisation d'une force de gravité et d'une force de friction. Nous fournissons une architecture contenant (dans certains fichiers) des trous que vous devrez compléter.

Pour compiler le projet, un `Makefile` vous est fourni. La règle principale `all` construit votre projet (`make`). La règle `run` compile et lance le projet (`make run`).

1 – Géometrie

Tout d'abord, il est nécessaire de copier le travail effectué au TP7. L'implémentation (.ml) doit correspondre à l'interface (.mli) qui vous est donné au risque de ne plus pouvoir compiler le projet. Dans le fichier `geom2D.ml`, fournissez aux modules `Vector` et `Point` les définitions des fonctions manquantes.

2 – Les corps et les forces

Commencez par ouvrir le module `Body` (`body.ml`). Un corps est défini par 3 attributs : une masse, une position dans le plan et un vecteur de vitesse. De plus, nous donnons également à un corps un rayon et une couleur qui nous servirons à l'affichage. Pour des soucis d'efficacité, à chaque objet sera associé un identifiant entier (`body_id`).

Dans le module `Force` (`force.ml`), nous définissons le type modélisant les forces. Le type `t` est composé d'un identifiant de corps (`body_id`) et d'une force (`force_on` qui va être appliqué au corps. Le type `force_on` va, quant à lui, contenir un type somme de toutes les forces que l'on va définir.

Dans un premier temps, nous allons nous concentrer sur la force de gravité. Ajoutez au type `force_on`, le champ `Grav` muni d'un identifiant de corps.

Pour illustrer, si nous souhaitons modéliser la force de gravitation exercée par la terre ($id = 1$) sur un objet ($id = 0$), nous écrirons **(0, Grav 1)**.

3 – Mondes

Afin de créer des "scénarios", il faut pouvoir représenter le monde dans son ensemble. Dans le module `World`, le type `t` contient 3 champs : un tableau de corps **où chaque index sera considéré comme l'identifiant de l'objet**, une liste des forces modélisant les relations entre chaque corps et une constante gravitationnelle.

4 – Physique

Maintenant que nous sommes capables de construire des corps, des forces et un monde, il faut calculer les interactions entre les corps. Dans le module `Physique` (d'interface `physique.mli`), nous souhaitons définir trois fonctions principales :

- `eval_force` : la fonction qui prend en argument un monde et une force et calcule le vecteur résultat de l'application de cette force.
- `sum_force` : cette fonction prend en argument un monde et calcule, pour chaque corps, la somme des forces (sous forme de vecteur) à laquelle il est soumis. Ce calcul donne un tableau de vecteur résultat.

- `step` : enfin, la fonction step prend en argument un monde, le tableau des forces calculé par la fonction `sum_force` et un flottant représentant un pas de temps. Cette fonction calcule un nouveau monde dans lesquels les corps auront leurs positions et leurs vecteurs de vitesse mis-à-jour.

1. Donnez une définition de la fonction `eval_force` qui va (dans un premier temps) calculer la force gravitationnelle donnée par l'équation :

$$\overrightarrow{F_{A/B}} = G\frac{m_A m_B}{\|\overrightarrow{BA}\|^3}\overrightarrow{BA}$$

   où G est la constante gravitationnelle et d la distance entre les deux corps.

2. Donnez une définition de la fonction `sum_force` qui va calculer la somme de toutes les forces entre chaque objets. La manière la plus simple de procéder est de déclarer un tableau de vecteur nul de taille le nombre de corps et de mettre à jour, pour chaque force associée, le vecteur-résultat de la fonction `eval_force`.

3. La dernière fonction `step` va mettre à jour les corps (vitesse et position) en fonction d'un pas de temps dt. La nouvelle vitesse d'un corps se calcule ainsi : $\overrightarrow{v}' = \overrightarrow{v} + \frac{\overrightarrow{f}}{m}dt$ où $\overrightarrow{f}$ est la somme des forces du corps. La nouvelle position est une simple translation de l'ancienne position avec le nouveau vecteur de vitesse : $p' = p + \overrightarrow{v}dt$.

4 – Tester le moteur

Tous les composants sont maintenant présents pour pouvoir tester le moteur. Le fichier `run.ml` est le point d'entrée du programme. En l'état, ce module lance un monde pré-défini via une configuration. Ces configurations (définies dans `config.ml` permettent de donner des options au programme (taille de l'écran, point de référence, pas de temps, écriture des logs, etc.). Si tout a été correctement implanté, vous devriez voir afficher deux planètes en orbite autour d'un astre.

En vous inspirant de l'exemple donné (`orbital_world.ml`), créez un monde contenant une balle et une terre espacés de $\text{rayon}_{\text{terre}} + 10$ mètres. Ajoutez également une force de gravité. Modifiez les paramètres à votre guise et observez le résultat.

5 – Aller plus loin

Ajoutez maintenant un champ $Fric$ dans le module `Force` qui va permettre de modéliser une force de friction. Cette force, prenant un coefficient de friction en paramètre (un flottant), devra réduire la vitesse de l'objet à chaque pas de temps.

$$\overrightarrow{F}_{fric} = -\lambda\overrightarrow{v}$$

Pour tester cette nouvelle force, vous pouvez lancer le moteur avec la configuration donnée par le fichier `ball_world.ml` (ou encore écrire la votre).