# 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

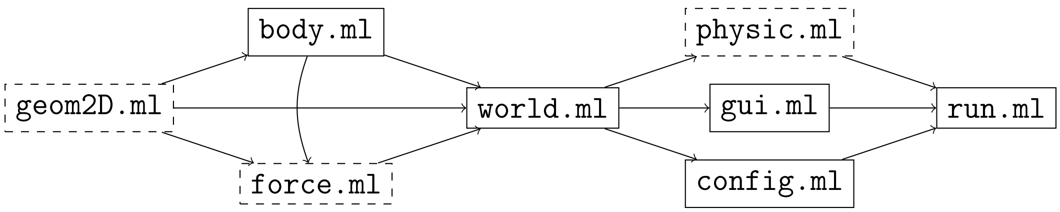


Figure 1: Architecture du projet. Les fichiers en pointillé sont les parties que vous aurez à modifier au cours du projet

`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 – Nouvelles forces

On va ajouter dans notre moteur physique de nouvelles forces. Pour chacune d'elles, il faudra donc rajouter un constructeur au type somme `forceon` du module `Force` et le calcul effectif de cette force dans la fonction `calc_force` du module `Physique`.

- **Force de frottement**:

$$\overrightarrow{F_{fric}} = -\lambda.\vec{v}$$

  où $\lambda$ est un nombre flottant et $\vec{v}$ la vitesse du corps sur lequel s'applique le frottement. Le constructeur associé à cette force sera `Fric`.
  Par exemple, si une force de frottement de constante 0.05 est appliquée sur le corps d'`id` 1, on écrira : `(1, Fric 0.05)`

- **Force électromagnétique de Lorentz $\overrightarrow{L}$.** Elle est semblable à l'expression de la gravité et le constructeur associé à cette force est nommé `Elec`. La force électromagnétique s'écrit comme :

$$\overrightarrow{L_{A/B}} = K\frac{q_a q_b}{\left\|\overrightarrow{BA}\right\|^3}\overrightarrow{BA}$$

  où $K$ est la constante de charge du monde et $q_A$ la charge du corps $A$.
  Il faut donc rajouter la constante de Coulomb, appelée $K$, dans le monde (dans le module `World`) et la charge d'un corps dans le module `body`.
  Par exemple, pour exprimer la force électromagnétique que le corps d'`id` 1 exerce sur le corps d'`id` 1, on écrira :`(0, Elec 1)`

- **Tension d'un ressort $\overrightarrow{K_{A/B}}$** (constructeur nommé `Spri`). La tension d'un ressort dépend de la longueur à vide $l_0$ du ressort et de sa raideur $k_{ressort}$:

$$\overrightarrow{K_{A/B}} = k_{ressort}(\left\|\overrightarrow{BA}\right\| - l_0)\vec{u}$$

  où $\vec{u}$ est un vecteur unitaire (de norme égale à 1) dans la direction de $\overrightarrow{AB}$.
  Par exemple, si on un ressort de raideur 1.5 et de longueur initiale 1.0 entre les corps d'`id` 0 et 1, on écrira : `(0, Spri (1.,1.5,1)`.

## 6 – Champs de vecteurs

Jusqu'à présent, on exprimait les forces grâce à leur équation. On va maintenant rajouter la possibilité d'exprimer une force grâce à un champ de vecteur. Un champ de vecteur permet, pour chaque point de l'espace, de lui associer un certain vecteur représentant.

Q1 – Dans un premier temps, nous allons rajouter les types nécessaires à cette implémentation. Dans un monde, on aura désormais, en plus d'une liste de force, une liste de champs appelée `fields`. Un champ sera modélisé comme une fonction d'un point vers un vecteur ou un floatant (pour les frottements). Un champ (type `field` dans le module `Force)` sera désigné comme :

- `E of (point -> vector)` pour désigner un champ électrique
- `G of (point -> vector)` pour désigner un champ gravitationnel.
- `F of (point -> float)` pour désigner un champ de frottement.

Ainsi, la liste `fields : field list` de notre monde donne la définition des champs présents dans le monde.

Pour préciser que des corps de notre monde sont soumis à forces induites par des champs, on ajoute les constructeurs au type `force`:

- Force de frottements local, constructeur `LFric` (associée au champs `F`)

- Force de gravité local, constructeur `LGrav` (associée au champs `G`)
- Force électrique locale,constructeur `Lelec` (associée au champs `E`)

On rappelle que les constructeurs de force servent à préciser quels corps du monde sont soumis à quels champs.

Q2 – Étant donné un ensemble de champs, certaines nouvelles forces peuvent s'appliquer sur les corps présent dans notre monde. Ajouter à la fonction `calc_force` les nouvelles forces possibles (`LElec`,`LGrav` et `LFric`) sachant que :

- La force s'éxerçant sur un corps de masse $m$ à la position $p$ soumis à un ensemble de champs de gravité $\mathfrak{G}$ est : $\overrightarrow{F} = m \sum_{g \in \mathfrak{G}} \overrightarrow{g(p)}$
- La force s'éxerçant sur un corps de charge $q$ à la position $p$ soumis à un ensemble de champs électrique $\mathfrak{E}$ est : $\overrightarrow{F} = q \sum_{e \in \mathfrak{E}} \overrightarrow{e(p)}$
- La force s'éxerçant sur un corps de vitesse $\overrightarrow{v}$ à la position $p$ soumis à un ensemble de champs de frottements $\mathfrak{F}$ est $\overrightarrow{F} = -(\sum_{f \in \mathfrak{F}} f(p))\overrightarrow{v}$

**Remarque:** L'ajout de champs rend plus simple la simulation effectuée dans `Ball_world.ml`. En effet nous devions simuler la présence d'une planète très loin en dessous des deux balles. Il suffit maintenant de faire l'hypothèse que la gravité est la même partout à la surface de la terre et d'ajouter un champs de gravitation qui à chaque point de l'espace associe le vecteur $\overrightarrow{g}$ qui est un vecteur pointant vers le bas, de norme 9.81.

# 1 Application : Tir à l'arc

Dans cet exercice, nous allons étudier la trajectoire d'une flèche tirée avec un arc et vérifier que notre simulation numérique est cohérente.

## 1.1 Partie analytique

On tire une flèche de masse $m_f$ avec une vitesse initiale $\overrightarrow{v_0}$ [1] et un angle $\theta$ par rapport au sol. Cette flèche est uniquement soumise à la gravité terrestre.

On chercher à calculer la hauteur maximale $H$ à laquelle la flèche monte et la position $D$ où la flèche va atterrir. Celles si sont données par :

$$H = \frac{v_0^2}{2g} \sin^2(\theta) \qquad D = \frac{v_0^2}{g} \sin(2\theta)$$

[1] On appelle $v_0$ la norme de $\overrightarrow{v_0}$

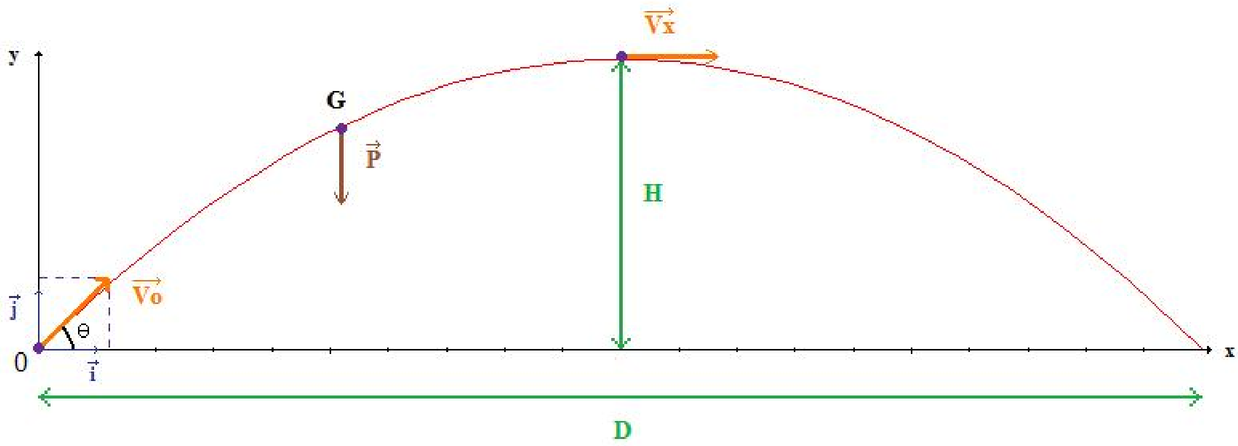


Figure 2: Modélisation du problème. Source : `https://www.ilephysique.net/`

où $g$ est l'intensité de la pesanteur et vaut 9.81.

Q3 – Donner la définition d'un monde `w` avec un corps correspondant à une flèche soumise à la gravité terrestre (deux choix pour cela : soit rajouter un "gros" corps représentant la Terre, soit exprimer la gravité exprimée par la Terre avec un champ gravitationnel). La position initiale de la flèche est $(0, 0)$, son poids est de 30 grammes.

Q4 – Donner la fonction `hauteur_max : float -> float -> float -> float` qui étant donné $g$, $\theta$ (en radians), $v_0$ calcule la hauteur maximale à laquelle la flèche va monter.

Q5 – Donner la fonction `position_max : float -> float -> float -> float` qui étant donné $g$, $\theta$ (en radians), $v_0$ calcule la position d'arrivée de la flèche.

## 1.2 Simulation numérique

Dans le module `Physic`, nous avons implémenté la fonction `step`. Cette fonction prend en paramètres un monde, une liste des sommes des forces pour chaque corps du monde et un pas de temps. Elle calcule les positions et les vitesses des corps du monde après le pas de temps.

Q6 – Donner une fonction qui va appliquer cette fonction sur votre monde et trouver la hauteur maximale de la flèche et sa position d'arrivée. Pour chaque étape (pour chaque appel de la fonction `step`), il faudra vérifier si la position de la flèche est maximale jusqu'à présent et si elle a touché le sol (i.e. si la première coordonnées de sa position est égale à 0).

Q7 – Comparer les valeurs de votre simulation numérique et celles de la partie analytique. Sont-elles identiques ? Diminuer le pas de temps $dt$. Que remarque-t-on?