Programmation Web Avancée
Master 1 Informatique

16/1/2016

TP nº 2

Moteur Physique

Le but de ce TP est d'implémenter en Javascript un petit moteur physique paramétrable et au comportement réaliste. Il fonctione en utilisant le principe fondamental de la dynamique ($\Sigma\vec{f} = m \cdot \vec{a}$) et utilise quelques notions de calcul vectoriel basiques (mais judicieusement appliquées). Il n'est pas nécessaire de comprendre la physique ou les maths sous-jacentes pour écrire le code.

On s'attachera par contre à suivre un modèle MVC. Les classes gérant le moteur physique et les classes faisant le rendu graphique de la scène sont bien distinctes.

Questions

1. Récupérer l'archive sur la page du cours. Éditer le fichier `index.html` pour charger dans la balise <head> un script `vector.js`.

> **Réponse:** Voir fichiers joints.

2. Créer un constructeur pour un objet `Vector` prenant deux arguments, `x` et `y` et initialisant les propriétés `x` et `y` de l'objet à ces valeurs, de manière à ce qu'elles soient *read-only*. Ajouter ensuite au prototype de `Vector` les méthodes :
   - `.add(v)` qui crée un *nouveau* vecteur étant la somme du vecteur courant et de `v`
   - `.sub(v)` qui crée un *nouveau* vecteur étant la différence du vecteur courant et de `v`
   - `.mult(k)` qui crée un *nouveau* vecteur en multipliant chaque composante du vecteur courant par la constante `k`
   - `.dot(v)` qui renvoie le produit scalaire entre le vecteur courant et le vecteur `v` (on rappelle que le produit scalaire se calcule par $x \times x_v + y \times y_v$).
   - `.norm()` qui renvoie la norme du vecteur courant (i.e. $\sqrt{x^2 + y^2}$)
   - `.normalize()` qui renvoie un nouveau vecteur colinéaire au vecteur courant, mais de norme 1 (i.e. le vecteur $\vec{v} \times \frac{1}{|\vec{v}|}$).

   Ajouter enfin à l'objet `Vector` une constante `ZERO` qui contient le vecteur nul.

   Charger la page HTML et vérifier l'absence d'erreur de syntaxe.

3. Créer un fichier `rect.js` contenant un constructeur `Rect`. Ce dernier prend en argument un `Vector` `v`, une largeur `w` et une hauteur `h` et s'en sert pour définir les propriétés `origin` (le coin supérieur gauche), `width` et `height` de l'objet construit. Faire en sorte que `width` et `height` soient *read-only*. Ajouter ensuite au prototype de `Rect` les méthodes suivantes :
   - `.move(v)` qui déplace l'origine du rectangle du vecteur `v`.
   - `.mDiff(r)` qui calcule la *soustraction de Minkowksi* entre le rectangle courant et le rectangle `r`. Cette dernière est un *nouveau* rectangle défini de la manière suivante. Si on appelle $x_{\mathtt{c}}$, $y_{\mathtt{c}}$, $w_{\mathtt{c}}$, $h_{\mathtt{c}}$ les coordonnées et dimentions du rectangle courant `c` et $x_{\mathtt{r}}$, $y_{\mathtt{r}}$, $w_{\mathtt{r}}$, $h_{\mathtt{r}}$ celles de `r`, la soustraction de Minkowski $\mathtt{r} \ominus \mathtt{c}$ est un rectangle défini par :

$$\begin{aligned} x &= x_{\mathtt{r}} - x_{\mathtt{c}} - w_{\mathtt{c}} \\ y &= y_{\mathtt{r}} - y_{\mathtt{c}} - h_{\mathtt{c}} \\ w &= w_{\mathtt{r}} + w_{\mathtt{c}} \\ h &= h_{\mathtt{r}} + h_{\mathtt{c}} \end{aligned}$$

   - `.hasOrigin()` qui renvoie `true` si et seulement si le point $(0, 0)$ est contenu dans le rectangle courant.

> **Réponse:** Voir le fichier joint. Bien rappeler aux étudiants de charger régulièrement leur fichier dans Chrome pour vérifier la syntaxe.

4. Créer un fichier `body.js` contenant un constructeur `Body`. Ce dernier prend les même paramètres que `Rect` et prend en plus un paramètre `m`. Faire en sorte que `Body` hérite de `Rect` et appelle le constructeur de `Rect` avec les bons paramètres. On initialisera ensuite les propriétés suivante pour l'objet :
   — `mass` à `m` (la masse du corps)
   — `invMass` à `1/m` (pratique pour les calculs)
   — `velocity` à `Vector.ZERO` (la vitesse du corps)
   — `force` à `Vector.ZERO` (la somme des forces exercées sur le corps).
   Initialiser correctement les propriétés `prototype` et `prototype.constructor` de `Body` pour permettre l'héritage. Ajouter ensuite au prototype une méthode `.collision(b)` qui renvoie pour l'instant `null`. Cette fonction compare le corps courant et le corps `b`. Si ces derniers ne rentrent pas en collision, alors la méthode renvoie `null`. Sinon, la méthode renvoie un objet `{ velocity1 :` v`, velocity2 :` v'`}` qui represente les nouveaux vecteurs vitesse des deux objets après collision (à faire dans la deuxième partie du TP).
5. Ouvrir le fichier `engine.js` et le lire. Ce dernier contient la boucle principale du moteur physique. Le moteur consiste en un tableau de corps (objets de type `Body`). La mise à jour du « monde » se fait ainsi. La méthode `.update(dt)` est appellée avec un certain intervalle de temps `dt` (le temps qui s'est écoulé depuis la dernière mise à jour). Pour chacun des objets, on effectue la chose suivante :
   — On le compare à chacun de tous les autres objets et on vérifie si les deux objets sont en collision. Si c'est le cas, on ajuste leur vecteur vitesse (les objets « rebondissent » l'un sur l'autre)
   — On calcule ensuite l'accéleration de l'objet (l'ensemble des forces qui lui sont soumises, divisé par la masse de l'objet). Et on réinitialise les forces appliquées à l'objet à ($\overrightarrow{(0,0)}$).
   — On calcule la variation de vitesse induite en multipliant l'accéleration par `dt` (que l'on ajoute à la vitesse totale de l'objet)
   — On calcule la nouvelle position de l'objet en multiplant sa vitesse par `dt`.
6. Créer un fichier `sprite.js` contenant un constructeur `Sprite`. Ce dernier hérite de `Body` et prends les même paramètres plus un paramètre `dom`. Le constructeur appelle celui de `Body` avec les bons paramètres et initialise la propriété `display` à `dom` (on s'attend à ce que dernier soit un élément `div`, dont l'attribut classe vaut `object` et qui soit un fils de `div#canvas`). Ajouter au prototype de `Sprite` une méthode `draw` qui redessine l'objet en méttant à jour les propriétés `left` `top` `width` et `height` du style CSS (comme pour le TP 1).
7. Ouvrir le fichier `renderer.js` et le lire. Ce dernier contient un objet `Renderer` qui permet de faire le rendu graphique d'un `Engine`.
8. Ouvrir le fichier `main.js`. En dessous de la création des 4 murs, créer un nouvel objet `Engine` et y ajouter les quatres murs. Créer ensuite un objet `Renderer` en lui passant l'objet `Engine` en argument. Faites en sorte que la méthode `.update(dt)` de votre objet `renderer` soit appelé 60 fois par secondes. Afin que le programme ne boucle pas en cas d'erreur, rattraper les éventuelles exceptions levées par `renderer.update`, arrêter la répetition du *callback* (avec la fonction `clearInterval`) et relancer l'exception. Relancer le programme et vérifier que lorsque qu'on clique dans le `div#canvas` des objets se créent et sont supprimé quand on clique dessus. Lire attentivement le code de gestion de la souris et le commenter le judicieusement. Essayer de commenter la première ligne « `if (this != ev.target) return;` » pour qu'elle ne s'exécute pas et expliquer ce qui se passe. Proposer une explication.

> **Réponse:** Le code de gestion de la souris est associé au canvas. Quand on clique dessus, on crée un nouvel objet div, que l'on positionne aux coordonnées de la souris. On donne à cet objet la classe `"object"` afin que le style CSS s'applique. On rajoute à ce nouvel objet un handler pour l'évènement `"click"`. Lorsque l'on clique sur l'objet, celui-ci est supprimé du moteur et son div est retiré de la page.
>
> Le premier test est nécessaire car les évènements sont *propagés* en Javascript, de la cible vers tout ses ancêtres. Lorsque l'on clique sur une petite boite, cette dernière disparaît, mais l'évènement clic est passé au parent, i.e. le `div` d'id `canvas`. Sur reception de cet évènement le gestionaire est

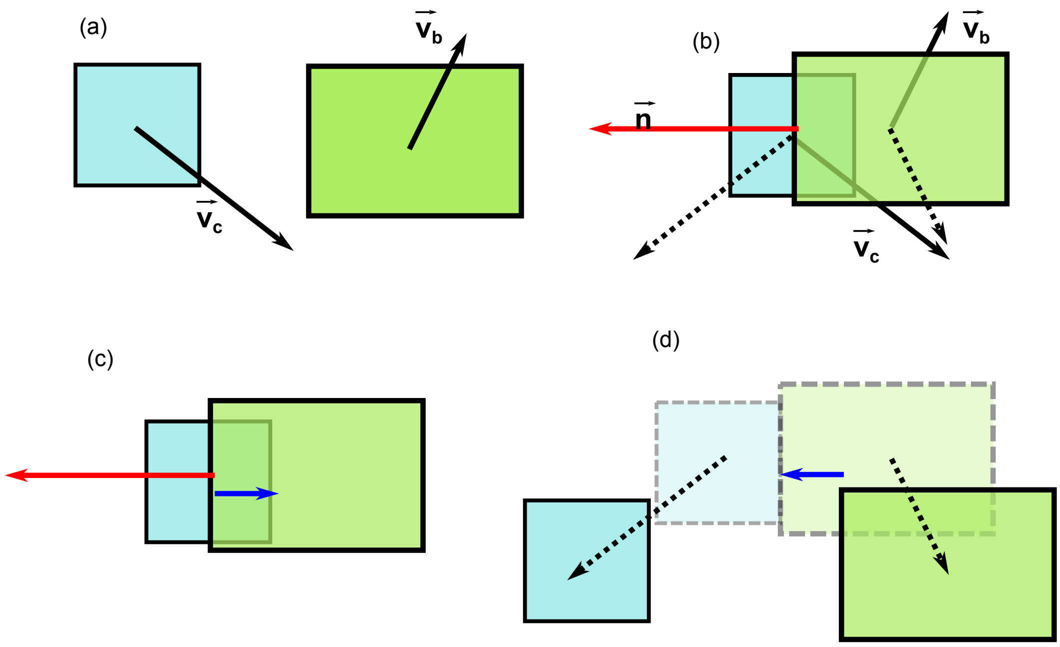

FIGURE 1 – Illustration de la résolution de collision entre l'objet c (bleu clair) et l'objet b (en vert).

réactivé et on recrée une nouvelle petite boite. Il faut donc tester dans le gestionnaire si l'objet (HTML) courant qui reçoit l'évènement est bien celui qui était la cible initiale.

Gestion des collisions

La manière dont nous avions traîté les collisions dans le TP 1 (jeu Pong) était rudimentaire et *ad-hoc*. Rajouter un obstacle signifiait rajouter un cas particulier de test. De plus, les tests de collisions n'étaient faits qu'avec les « faces » des raquettes. Le modèle physique que l'on implémente est facilement extensible (voir [1, 2] ainsi que d'autres ressources sur Internet).

Nous représentons les objets par leur AABB (*axis-aligned bounding box*), *i.e.* des rectangles dont les côtés sont parallèles aux axes (x, y). Cette technique est utilisée dans de nombreux jeux (et peut se généraliser en 3D avec des parallépipèdes. Évidemment on peut dessiner un objet de forme *non-rectangulaire* dans la boîte, une voiture, un personnage, ...mais on utilisera sa boîte pour detecter des collisions). Tout le code doit être placé dans la méthode `.collision(b)` de la classe `Body`. Cette dernière renvoie `null` si l'objet courant (nommé `c` dans la suite) et l'objet cible `b` n'ont pas de collision et sinon renvoie leur nouvelle vitesse. La figure 1 illustre le phénomène. On applique l'algorithme suivant :

1. Est-ce que les deux rectangles s'intersectent ? Si non, pas de collision.
2. Si les rectangles s'intersectent, on trouve *le vecteur normal* au point d'intersection, $\vec{\mathbf{n}}$. Dans notre modèle simplifié, ce vecteur est perpendiculaire aux faces qui se rencontrent (voir figure 1 (b)). C'est par rapport à ce vecteur que sont calculées les nouvelles vitesses qui représentent le « rebond » des objets (en pointillé sur la figure).
3. On ajuste la position des objets. Cette partie est nécessaire et cause de bug graphiques si elle est mal implémentée (les objets « coulent » les uns dans les autres). En effet, s'il y a collision, les objets sont partiellement superposés. Supposons qu'ils ont une vitesse de rebond très faible (ou nulle, par exemple dans le cas d'un mur), alors à l'itération suivante, ils ne se seront pas assez déplacés pour se séparer et seront toujours en collision. Cependant, les vecteurs vitesse des objets sont maintenant orientés de manière à séparer les objets. On a donc deux objets qui se s'éloignent mais qui sont l'un dans l'autre. L'algorithme de collision peut ne pas prévoir ce cas, ou alors considérer que l'un des objets est à l'interieur de l'autre et qu'il se cogne en essayant de sortir, et va donc le faire rebondir dans la mauvaise direction .... On doit donc déplacer les objets d'un montant équivalent au vecteur

de pénetration (figure 1 (c), en bleu), dans la direction $\vec{\mathbf{n}}$ pour l'objet c et dans la direction $-\vec{\mathbf{n}}$ pour l'objet b.

4. Une fois calculé $\vec{\mathbf{n}}$, on applique les formules de la mécanique du point pour calculer les vecteurs de rebonds. Ces calculs font intervenir les vitesses initiales, les masses des objets (un objet léger aura du mal a « pousser » un objet plus lourd, et en particulier il ne poussera pas du tout un objet de masse infinie comme un mur).

La soustraction de Minksowski $\mathsf{s} = \mathsf{b} \ominus \mathsf{c}$ entre les deux boîtes possède des propriétés particulièrement intéressantes.

— Si $(0,0)$ est dans le rectangle s, alors b et c sont en collision
— La plus petite distance entre $(0,0)$ et un bord de s donne exactement le vecteur de pénétration (figure 2).

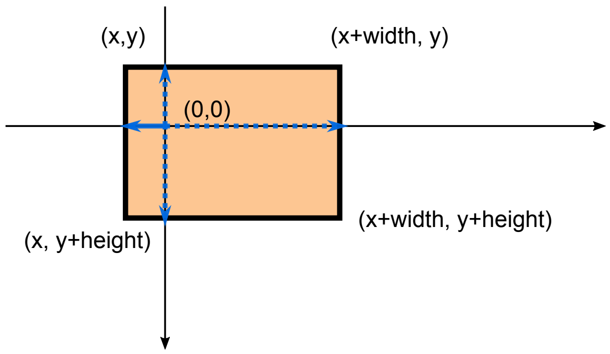

FIGURE 2 – Illustration de la différence s de Minkowski entre c et b

Algorithme détaillé de calcul des collisions

— Calculer $\mathsf{s} = \mathsf{b} \ominus \mathsf{c}$
— Si s ne contient pas l'origine (utiliser `.hasOrigin()`), alors renvoyer `null`.
— Sinon, calculer les 4 vecteurs bleus de la figure 2 et garder celui qui à la norme la plus petite. On appelle $\vec{\mathbf{n}}$ ce vecteur.
— calculer le rapport des vitesses entre b et c :

$$N_{\mathsf{c}} = \frac{|\vec{v}_{\mathsf{c}}|}{|\vec{v}_{\mathsf{c}}|+|\vec{v}_{\mathsf{b}}|}$$

$$N_{\mathsf{b}} = \frac{|\vec{v}_{\mathsf{b}}|}{|\vec{v}_{\mathsf{c}}|+|\vec{v}_{\mathsf{b}}|}$$

Cela permet de « replacer » les objets proportionnellement à leur vitesse (dans le cas où l'un des objets est un mur de vitesse nulle, on déplace complètement l'autre objet, si les deux objets ont la même vitesse, on déplace chacun de la moitié de $\vec{\mathbf{n}}$, ...). Si $|\vec{v}_{\mathsf{b}}| = |\vec{v}_{\mathsf{c}}| = 0$, alors :

— Si les deux objets sont de masse infinie, la fonction renvoie `null` (on considère que deux murs ne font pas de collision)
— Si b est plus lourd que c on définit $N_{\mathsf{c}} = 1$ et $N_{\mathsf{b}} = 0$, sinon on définit $N_{\mathsf{b}} = 1$ et $N_{\mathsf{c}} = 0$. (i.e. on considère que l'objet le plus lourd des deux « éjecte » le plus léger.

On déplace c de $N_{\mathsf{c}} \times \vec{\mathbf{n}}$ et b de $-N_{\mathsf{b}} \times \vec{\mathbf{n}}$

— On normalise $\vec{\mathbf{n}}$
— On calcule la vitesse relative $\vec{v} = \vec{v}_{\mathsf{c}} - \vec{v}_{\mathsf{b}}$
— On calcule l'impulsion j :

$$j = \frac{-(1+e) \times \vec{v} \cdot \vec{\mathbf{n}}}{\frac{1}{M_{\mathsf{c}}} + \frac{1}{M_{\mathsf{b}}}}$$

où M_i représente la masse de l'objet i et « $\cdot$ » représente le produit scalaire de deux vecteurs. Ici $e = 1$ (elasticité parfaite). Si on remplace par e = 0, les objets absorbent intégralement les chocs et ne rebondissent pas (et une valeur entre 0 et 1 absorbera plus ou moins les chocs créant des rebonds plus ou moins forts). En pratique e n'est pas une constante du système mais est le *coefficient de restitution* et dépend des matériaux des deux objets (par ex : acier/acier $= 19/20$, bois/bois $= 1/2$, ...).

— On clacule :

$$\vec{v'}_{\mathtt{c}} = \vec{v}_{\mathtt{c}} + \vec{\mathbf{n}} \times \frac{j}{M_{\mathsf{C}}}$$

$$\vec{v'}_{\mathtt{b}} = \vec{v}_{\mathtt{b}} - \vec{\mathbf{n}} \times \frac{j}{M_{\mathsf{b}}}$$

(attention l'addition et la soustraction ci-dessus sont celles sur les vecteurs).

— On renvoie (enfin !) `{ velocity1 :` $\vec{v'}_{\mathtt{c}}$`, velocity2:` $\vec{v'}_{\mathtt{b}}$ `}`

On peut ensuite s'amuser à faire varier les masses, et divers coefficients, à ajouter des objets de tailles et masses différentes sur la scène, à modifier le style graphique (en fonction de la vitesse par exemple). On remarque aussi que si les objets vont trop vites, ils passent à travers les murs. Une technique plus élaborée de collision peut être mise en œuvre (*swept* collision ou *continuous collision detection*).

Références

[1] My physics lab, article sur les collisions entre solides. `http://www.myphysicslab.com/engine2D/collision/collision-en.html`.

[2] Un tutoriel sur l'utilisation de la soustraction de minkowski pour les collisions. `https://hamaluik.com/posts/simple-aabb-collision-using-minkowski-difference/`.