Projet PFA

Cours 6

kn@lri.fr
http://www.lri.fr/~kn

université PARIS-SACLAY

Aide sur quelque points

- Lecture de fichiers depuis Javascript
- Ajout de la gravité/frottement pour un jeu de plateforme
- Entrées utilisateur plus flexibles
- Scrolling (déplacement de l'écran) la semaine prochaine

Plan

Lecture en Javascript ?

En Javascript, on n'a pas directement accès au système de fichiers (pour des raisons de sécurité).

Le compilateur `js_of_ocaml` propose une solution:

- Créer des fichiers et leur contenu comme des chaînes de caractères stockées dans un tableau global en Javascript
- Modifier les fonctions OCaml `open_in`, `input_line`, … pour aller lire dans ces chaînes plutôt que dans des vrais fichiers.

Comment faire en pratique ?

On modifie le fichier `prog/dune` pour passer les bonnes commandes :

```
(executable
 (name game_js)
 (modes js)
 (modules game_js)
 (preprocess (pps js_of_ocaml-ppx))
 (libraries game js_of_ocaml gfx_jsoo)
 (js_of_ocaml (javascript_files "jsoo_fs.js"))
)

(rule
 (deps (glob_files ../resources/static/*))
 (target "jsoo_fs.js")
 (action (run js_of_ocaml build-fs -I . -o %{target} %{deps})))
```

Ce fichier est fourni dans le code d'exemple, vous pouvez le récupérer ou copier les lignes dans le fichier `prog/dune`
Un fichier `f` de chemin `resources/static/f` sera accessible dans le code OCaml (compilé vers Javascript) via le chemin `/static/f`.

Plan

Rappels de physique

On rappelle le principe fondamental de la dynamique:

$$\sum \vec{F_i} = m\vec{a} \quad \text{ou} \quad \vec{a} = \frac{1}{m}\sum \vec{F_i}$$

En quoi cela nous aide ?

- Chaque tour de boucle principale prend t ms
- Si on calcule $\vec{v} = t \times \vec{a}$ on obtient la variation de vitesse
- On ajoute $\vec{v}$ à la vitesse courante de l'objet

On ajoute un nouveau composant `SumForces` qui est simplement un vecteur représentant la somme des forces. À chaque frame, on accumule les forces sur les objets qui ont se composant. On rajoute un nouveau système `Force_S` exécuté avant la detection de collision. Ce dernier :

- Récupère l'ensemble des forces $\vec{f}$
- Ajoute les forces « globales » gravité, frottements
- Calcule $\vec{a}$ puis $\vec{v} = t \times \vec{a}$ et ajoute à la vitesse courante

À propos de la gravité

Attention, l'accéleration due à la gravité ne dépend pas de la masse. (On lache 100g de plomb et 100kg de plomb à la même hauteur et ils tombent à la même vitesse).

On ne peut donc pas utiliser une constante pour la gravité, mais on doit multiplier par la masse de l'objet (et on redivisera par la masse une fois faite la somme de toutes les forces, ce qui compense).

Interet de l'approche

On traîte avec un cadre générique plein de choses :

- Gravité
- Frottements
- Déplacement d'un personnage (en donnant une impulsion à gauche, a droite ou vers le haut)

De plus, les valeurs de certaines constantes peuvent avoir des effets intéressants:

- Dans `Collision_S`, `e = 0`⇒pas de rebond, `e=1`⇒rebond parfait
- Un frottement très fort donne un mouvement, net. Un frottement faible fait que le personnage « glisse » quand on arrête de marcher.

Il faut beaucoup expérimenter pour trouver des bonnes valeurs pour un jeu. On ne prend pas en général de valeurs « réalistes ». ($G = 6.67430 \times 10^{-11} \mathrm{Nm}^2\mathrm{kg}^{-2}$, c'est très petit, ⇒ erreurs d'arrondis).

Frottements, position de repos

Les frottements sont appliqués lorsqu'on se déplace le long d'une surface (on parle uniquement des frottements du support). Ils ont une force proportionnelle à l'oposée de la composante horizontale de la vitesse.

Pour pouvoir les appliquer, on veut savoir si un objet est « appuyé » sur un autre. Cette information est disponible lors de la détection de collision. On ajoute un nouveau composant `Resting` qui est un booléen indiquant si on l'objet en question repose sur un autre.

Ce booléen `Resting` est mis à `true` si on a une collision verticale vers le haut entre deux boîtes.

Après, on peut autoriser le personnage à sauter uniquement s'il est en repos sur une surface.

Plan

Entrées utilisateur

Le système d'entrées utilisateurs que l'on a mis en place n'est pas idéal, en particulier il ne prend pas en compte les pressions simultanées. On propose la variation suivante :

- Détecter les évènements (touche enfoncée etc...) et mettre des flags booléens à vrai ou faux selon l'état de la touche.
- Une fois par frame, appliquer les commandes en ayant connaissance de tous les flags.

Plan

TP 6

Le « TP » 6, déjà codé, illustre tous ces concepts dans un « jeu » de plateforme minimaliste (c'est plutot un embryon de moteur). Il utilise

- Le squelette de code donné
- Les composants et types de base du Pong (`Vector`, `Box`, `Collision_S`, ...)

Il est vivement recommandé de s'en inspirer si on est bloqué