

TP1 : Casse brique sans briques

Objectifs Définition de fonctions simples et utilisation des n-uplets.

Rappels

- Si e_1 et e_2 sont deux expressions OBJECTIVE CAML, l'expression (e_1, e_2) est la paire composée de e_1 et e_2 .
- Si e_1 est de type t_1 et e_2 est de type t_2 alors (e_1, e_2) est de type $t_1 * t_2$.
- Enfin, les fonctions `fst` et `snd` permettent d'accéder respectivement à la première et à la seconde composante d'une paire.

Introduction Le but de ce TP est de créer un squelette de jeu de casse-brique qui consiste en un cadre et le mouvement d'une balle à l'intérieur de celui-ci. Vous utiliserez des fonctions de dessin prédéfinies pour donner vie aux objets modélisés (cadre et balle).

Exercice 1 - Le cadre

Constantes

Définir dans un fichier `tp1.ml` les valeurs globales `up`, `down`, `left` et `right` qui sont respectivement égales à `450.0`, `50.0`, `50.0` et `150.0`. Ces valeurs représente les limites du cadre de jeu.

Compiler votre programme avec la commande suivante : `ocamlc -o tp1 tp1.ml`

Dessin

Définir la fonction `draw_frame` de type `unit -> unit` qui affiche le rectangle correspondant au cadre du jeu. Vous utiliserez la fonction `int_of_float` qui convertit les nombres flottants en entier et la fonction `Graphics.draw_rect` qui affiche un rectangle. Vous irez regarder la spécification de `Graphics.draw_rect` sur <http://caml.inria.fr/pub/docs/manual-ocaml/libref/Graphics.html>.

Compiler votre programme avec la commande la commande suivante : `ocamlc -o tp1 graphics.cma tp1.ml`

Pour tester votre programme recopiez le code suivant :

```
let test_draw_frame () =
  Graphics.open_graph " 300x500";
  draw_frame ();
  ignore (read_line())
;;

let main =
  test_draw_frame ()
;;
```

Exercice 2 - La balle

Représentation

La balle est représentée par une paire de nombres flottants correspondant à sa position. Définir la fonction `random_ball` de type `unit -> float * float` qui crée une balle à une position aléatoire dans le cadre. Vous utiliserez la fonction `Random.float` dont la documentation se trouve sur <http://caml.inria.fr/pub/docs/manual-ocaml/libref/Random.html>.

Dessin

Définir la fonction `draw_ball` de type `float * float -> unit` qui affiche la balle. Vous utiliserez la fonction `Graphics.fill_circle`.

Exercice 3 - Mouvement

La direction et la vitesse de la balle sont données par un vecteur vitesse. Ce vecteur sera représenté par une paire de flottants. Définir la fonction `random_vect` de type `unit -> float * float` qui crée un vecteur dont chacune des composantes est comprise entre `-0.5` et `0.5`.

Définir la fonction `translation` de type `float * float -> float * float -> float * float` qui à partir d'un point et d'un vecteur vitesse calcule la nouvelle position du point.

Pour tester votre programme recopier le code suivant :

```
let rec active_wait n = if n > 0 then active_wait (n-1)
;;

let rec run step state n =
  let state' = step state in
  active_wait n;
  (run step state' n : unit)
;;

let step_move (p,v) =
  Graphics.clear_graph ();
  draw_frame ();
  let p' = translation p v in
  draw_ball p;
  (p', v)
;;

let test_move () =
  Random.self_init ();
  Graphics.open_graph " 300x500";
  let p = random_ball () in
  let v = random_vect () in
  run step_move (p, v) 100000
;;

let main =
  test_move()
;;
```

Exercice 4 - Intersections

Nous voulons maintenant tester si la balle reste dans les limites du cadre. Pour cela, il nous faut définir les quatre fonctions `out_left`, `out_up`, `out_right` et `out_down` de type `float * float -> bool` qui testent si la balle est sortie du cadre par la gauche, le haut, la droite ou le bas.

Définir la fonction `out` qui teste la sortie des limites d'une balle en utilisant les fonctions `out*`.

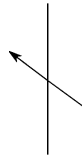
Définir la fonction `step_move_out` (en se basant sur `step_move`) pour que le programme termine lorsque la balle sort des limites. Vous utiliserez la fonction `failwith` pour indiquer la sortie.

Exercice 5 - Rebonds

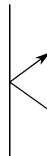
Enfin, il faut définir les fonctions qui calculent les rebonds. Comme pour les intersections, on définit le rebond suivant le coté.

Un rebond

Pour calculer un rebond sur le bord gauche, en supposant que la balle aurait du le traverser, c'est à dire dans la situation suivante :



On fait tout simplement la symétrie par rapport à l'axe pour obtenir la nouvelle position :



Le vecteur vitesse est aussi modifié en inversant sa composante x pour refléter le rebond. On peut de la même façon trouver les calculs correspondants au rebond sur chaque coté. Définir les quatre fonctions `rebond*` de type `(float * float) * (float * float) -> (float * float) * (float * float)` prenant en entrée la position en dehors du cadre et le vecteur vitesse de la balle et retournant la nouvelle position et le nouveau vecteur vitesse après le rebond.

Des rebonds

Définir la fonction `bounce` de type :

```
(float * float) -> (float * float) ->
(float * float) * (float * float)
```

prenant en entrée la nouvelle position de la balle, son vecteur vitesse et vérifiant si la nouvelle position est en dehors du cadre (avec les fonctions `out_*`) et le cas échéant retournant le nouvel état de la balle après rebond (sa position et son vecteur vitesse). Si la balle ne sort pas du cadre, on retourne la nouvelle position avec un vecteur vitesse inchangé.

Finalement, définir la fonction `step_bounce` en adaptant `step_move` pour gérer correctement les rebonds. Tester votre définition avec le code suivant :

```
let test_bounce () =
  Random.self_init ();
  Graphics.open_graph " 300x500";
  let p = random_ball () in
  let v = random_vect () in
```

```
run step_bounce (p, v) 100000  
let main = test_bounce()
```