Projet de programmation fonctionnelle: Un jeu de plateforme

Benoît Barbot

Licence 2, Université Paris Est Créteil

Résumé

Les Jeux de plates-formes sont un grand classique des jeux vidéos, dans ce projet il vous est proposé de réaliser un tel jeu. Les graphismes du jeu seront entièrement en ASCII art pour s'exécuter dans un terminal. Vous pouvez voir un exemple ici `https://prog-reseau-m1.lacl.fr/projetfun2023/exemple_mario.html`

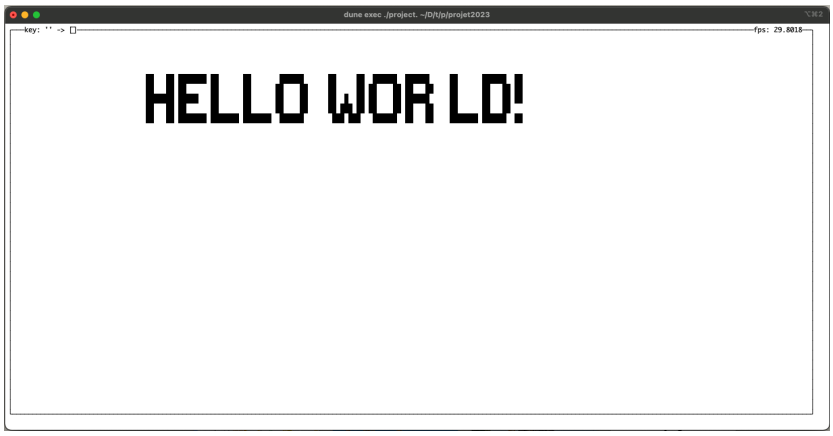

FIGURE 1 – Lancement du projet initial.

1 Consignes

toutes les informations pour le projet se trouve sur `www.caseine.org`. Vous y trouverez ce document, une archive `projet.zip` contenant les fichiers du projet, une activité de rendu et un lien vers une page démontrant le projet. Le but de la première partie de ce projet est de construire les briques nécessaires à la réalisation d'un jeu de plateforme en OCaml en utilisant les notions vues dans le cours de programmation fonctionnelle. Cette première partie est assez guidée comme les TDs. Les questions sont de moins en moins guidées au fur et à mesure de l'avancée du projet. La seconde partie vous demande de réaliser un tel jeu.

1.1 Compilation

Dans l'archive vous trouverez les fichiers suivants :

— Le fichier `projet.ml` est le fichier principal, c'est le seul que vous devez modifier, il est presque vide pour le moment.

— Les fichiers `engine.ml` et `engine.mli` contient du code OCaml qui met en place les entrées sorties et fournit un moteur physique pour le projet. Vous n'avez pas à le modifier

— Le fichier `Makefile` contient un script pour compiler le projet.

Pour compiler le projet vous aurez besoin d'installer `https://opam.ocaml.org/` un gestionnaire de paquet pour OCaml. Sous Windows, vous pouvez suivre les instructions disponibles ici `https://moodle.caseine.org/mod/book/view.php?id=43384`.

Le projet peut être exécuté de deux manières différentes :

— En mode compilé dans un terminal, après avoir compilé le projet avec `make` taper `./project` pour l'exécuter ;

— En mode application web, rendez vous à cette adresse `https://prog-reseau-m1.lacl.fr/projetfun2023/`. Sélectionnez le fichier source de votre projet (`projet.ml`) et cliquer sur soumettre.

1.2 Date de rendu

le projet est a réaliser par binôme le choix des binômes ce fait sur caséine : `https://moodle.caseine.org/mod/vpl/view.php?id=65204`. La première partie du projet doit être rendu mardi 7 novembre 23 h 59.

1.3 Les définitions qui vous sont données

Le fichier `engine.ml` définis des types et fonctions que vous devrez utilisez pour le projet. On rappelle ici quelques fonctions principales, les détails se trouvent dans la documentation `ici`.

```
type vec = float * float
val ( +.. ) : vec -> vec -> vec
val ( *.. ) : float -> vec -> vec
type solid = vec * string
val get_bounds : string -> vec
val is_above : solid -> solid -> bool
val is_below : solid -> solid -> bool
val is_right : solid -> solid -> bool
val is_left : solid -> solid -> bool
val collision : solid -> solid -> bool
val update_physics : solid -> vec -> vec list -> solid list
        -> vec * vec * (solid option * solid option)
val loop : 'state ->
           ('state -> key_mod -> 'state) ->
           ('state -> solid list) -> unit
```

Le fichier `projet.ml` contient un squelette de fonctions que vous aurez à modifier

```
open Engine

let hello = "..."

(* Calcule la taille en nombre de caractères horizontaux et
 verticaux du message*)
let h_width, h_height = get_bounds hello
```

```
(* Etat initial du système, c'est un couple de vecteurs, le
 premier contient la position du message, le second contient
 sa direction de déplacement *)
let init_state = ((10.0, 15.0), (1.0, 1.0))

(* Cette fonction est appelée de manière répétée plusieurs fois
 par seconde pour mettre à jours l'état du système. Ici l'état
 est un couple composé de la position du message et de sa
 direction. Elle doit renvoyer le nouvel état.*)
let update ((x, y), (xincr, yincr)) _ =
  let incr =
    ( (if x < 0.0 || x >= width -. h_width then -.xincr else xincr),
      if y < 0.0 || y >= height -. h_height then -.yincr else yincr )
  in
  let new_pos = (x, y) +.. incr in
  (new_pos, incr)

(* Affiche l'état du système, ici le message à la position pos,
 et les nombres 1,2,3,4 aux quatre coins de l'écran. *)
let affiche (pos, _) =
  [
    (pos, hello);
    ((0.0, 0.0), "1");
    ((width -. 1.0, 0.0), "2");
    ((width -. 1.0, height -. 1.0), "3");
    ((0.0, height -. 1.0), "4");
  ]

(* Appelle de la boucle principale le premier argument est l'état
 initial du système. Les autres arguments sont les fonctions update
 et affiche *)
let _ = loop init_state update affiche
```

1.4 Utilisation des caractères pour dessiner.

Dans tout le projet, on utilise uniquement des caractères pour dessiner le jeu. Vous pouvez copier-coller des caractères Unicode trouvés par exemple ici `https://www.compart.com/en/unicode/block`. Le type `solid = vec*string` est utilisé pour représenter un solide. Le vecteur indique la position en bas à gauche du solide. La chaîne de caractère peut contenir des sauts de lignes (`"\n"`) qui sont interprétés correctement. Par exemple on peut dessiner un personnage avec la chaîne de caractère suivante : `" o \n/O\\\n ^ "`. Qui sera dessiné par

```
 o
/O\
 ^
```

Pour savoir si deux solides sont en collision l'un avec l'autre les fonctions la fonction `get_bounds` calcul le plus petit rectangle qui contient l'objet décrit par la chaîne de caractère. Sur l'exemple précédent, `get_bounds` renvoie `(3,3)`.

2 Préliminaires

Commencez par compiler le projet, avec OCaml d'installé vous devez taper `make` dans un terminal où le répertoire courant est le dossier du projet ensuite, tapez `./project` pour lancer le projet. Vous devez voir apparaitre le message

"Hello world !" qui se déplace dans la fenêtre du terminal comme dans la figure 1. Lisez bien le fichier `project.ml` pour en comprendre chaque ligne. Le point principal est de comprendre ce que fait la dernière ligne qui appelle la fonction `loop: 'state->('state->key_mod->'state)->('state->solid list)->unit`. Le type polymorphe `'state` représente l'état complet du système. Pour le moment il est égal à `vec*vec` pour représenté la position du message et sa direction. Dans la suite vous devrez utiliser un type différent.

2.1 Les entrées clavier

Dans cette partie on va utiliser les touches du clavier pour mettre à jours l'état du système.

question 1 *Simplifier le programme dans* `project.ml` *en enlevant les déplacements du message. L'état du système sera uniquement la position du message (c.-à-d. de type `vec`). la fonction `update` va être presque vide.*

Le second argument de la fonction `update` (inutilisé jusqu'ici) est de type `key_mod = (key*bool*bool) option`. Sa valeur est `None` si aucune touche du clavier n'est pressée par l'utilisateur sinon il contient un triplet qui représente la touche pressée éventuellement modifiée par les touches Shift ou Control.

question 2 *Modifier la fonction `update` pour qu'une pression sur la touche 'd' déplace le texte vers la droite d'un caractère. Similairement 'q' déplace vers la gauche, 'z' vers le haut et 's' vers le bas. Si en plus la touche Shift est pressée, il faut déplacer de 5 caractères au lieu d'un seul.*

2.2 Moteur physique

La fonction `update_physics` implémente un moteur physique qui permet de simuler les déplacements d'un point en fonction de sa vitesse et des forces qui lui sont appliquées. Cette fonction prend aussi en charge les collisions avec les obstacles. Lisez sa documentation.

question 3 *remplacer le message "hello world !" par un personnage dessiné en caractère, par exemple*

```
" o
 /O\
  ^ "
```

Modifier l'état, de votre projet pour qu'il représente la position et la vitesse du personnage. Modifier votre fonction `update` pour qu'elle utilise `update_physics`. L'objet va être le personnage avec sa position sa vitesse. Les touches du clavier doivent modifier la vitesse dans la direction adaptée. La liste des forces doit être `[gravity]`, la liste d'obstacle est vide (`[]`). En exécutant votre projet, le personnage doit tomber.

question 4 *Ajouté une ligne de `'#'` en bas de l'écran en fabriquant un solide aux coordonnées `(0.0,0.0)`. Ajouter ce solide aux obstacles donné à la fonction `update_physics` et l'ajouter au personnage dans la fonction `affiche`. Le personnage doit maintenant s'arrêter sur la ligne.*

question 5 *Modifier la fonction* `update` *pour que l'on puisse déplacer le personnage horizontalement selon les déplacements voulut en fonction de la touche clavier, on peut ajouter un vecteur dans la liste des forces pour une accélération réaliste ou modifier directement la vitesse pour une accélération instantanée.*

2.3 Défilement

Quand le personnage se déplace, on veut pouvoir translater l'écran pour garder le personnage au centre de l'écran. Une manière de réaliser cela est de translater toutes les positions dans la fonction d'affichage `affiche`.

question 6 *Écrire une fonction* `translate : vec -> solid list -> solid list` *qui translate la position de chaque solide par le vecteur donnée en paramètre.*

question 7 *Modifier la fonction* `affiche` *pour qu'une fois que le personnage atteint la moitié de l'écran tout les éléments d'affichage soient translatés pour maintenir le personnage au centre.* Pensez a ajouter des obstacles pour que le défilement soit visible.

2.4 Plusieurs entités en mouvement

Pour que plusieurs entités puissent être en mouvement simultanément (par exemple, des ennemies, des projectiles, une partie des obstacles du terrain), il faut ajouter toutes ces entités à l'état du système et mettre à jour leurs positions indépendamment.

question 8 *Ajouté aux personnages la capacité d'envoyé des projectiles qui disparaisse quand il touche un obstacle.*