Ricochet Robots

2h : avec documents

Année 2019-2020

Préambule

- Les tests unitaires de chaque fonction doivent être donnés.
- Vous devez tout écrire dans le fichier `src/pos.ml`.
- Ne pas modifier les autres fichiers. Le seul que vous aurez besoin de lire est `src/plateau.mli`, vous pouvez ignorer le contenu des autres.
- Les noms et types de modules et fonctions doivent être respectés (tests automatiques).
- Pour tester dans `utop`, vous devez, dans le dossier `src` ouvrir les modules Be et Pos (**open** Be.Pos;;) pour avoir accès aux modules et interfaces que vous définirez.
- Le code rendu doit impérativement compiler. Pour cela, les fonctions non implémentées d'un module peuvent être remplacées par un code quelconque, par exemple **let** set = **fun** _ -> **assert** false .
- Les exercices sont indépendants mais il est préférable de les traiter dans l'ordre.

Sujet d'étude

Le but est de réaliser un programme trouvant des solutions optimales pour le jeu Ricochet Robots [1].
Le jeu est composé d'un plateau de 16 par 16 cases. Cinq robots de couleurs différents peuvent se déplacer sur ce plateau de la façon suivante :

- les robots se déplacent en ligne droite (horizontalement ou verticalement) ;
- les robots ne peuvent pas traverser les murs ;
- les robots ne peuvent pas sauter par dessus un autre robot ;
- les robots ne s'arrêtent *que* lorsqu'ils rencontrent un obstacle (mur ou autre robot) ou le bord du plateau.

Par exemple, sur la Figure **??**, le robot vert en $(8, 1)$ peut bouger en $(4, 1)$ (mur), en $(8, 0)$ ou en $(15, 1)$ (bords du plateau) ou en $(8, 2)$ (robot rouge).
Le but du jeu est d'amener les robots sur une cible donnée en un minimum de mouvements. Par exemple, sur la Figure **??**, le robot rouge peut rejoindre la cible lune rouge en $(4, 1)$ en trois mouvements : vert vers le haut, rouge vers le haut puis rouge vers la gauche.

Une partie du code est déjà donnée. Vous devrez uniquement :

- proposer un encodage de l'ensemble des positions des robots ;
- écrire des fonctions qui permettent de calculer l'ensemble des déplacements possibles en une unique étape.

1. https://fr.wikipedia.org/wiki/Ricochet_Robots

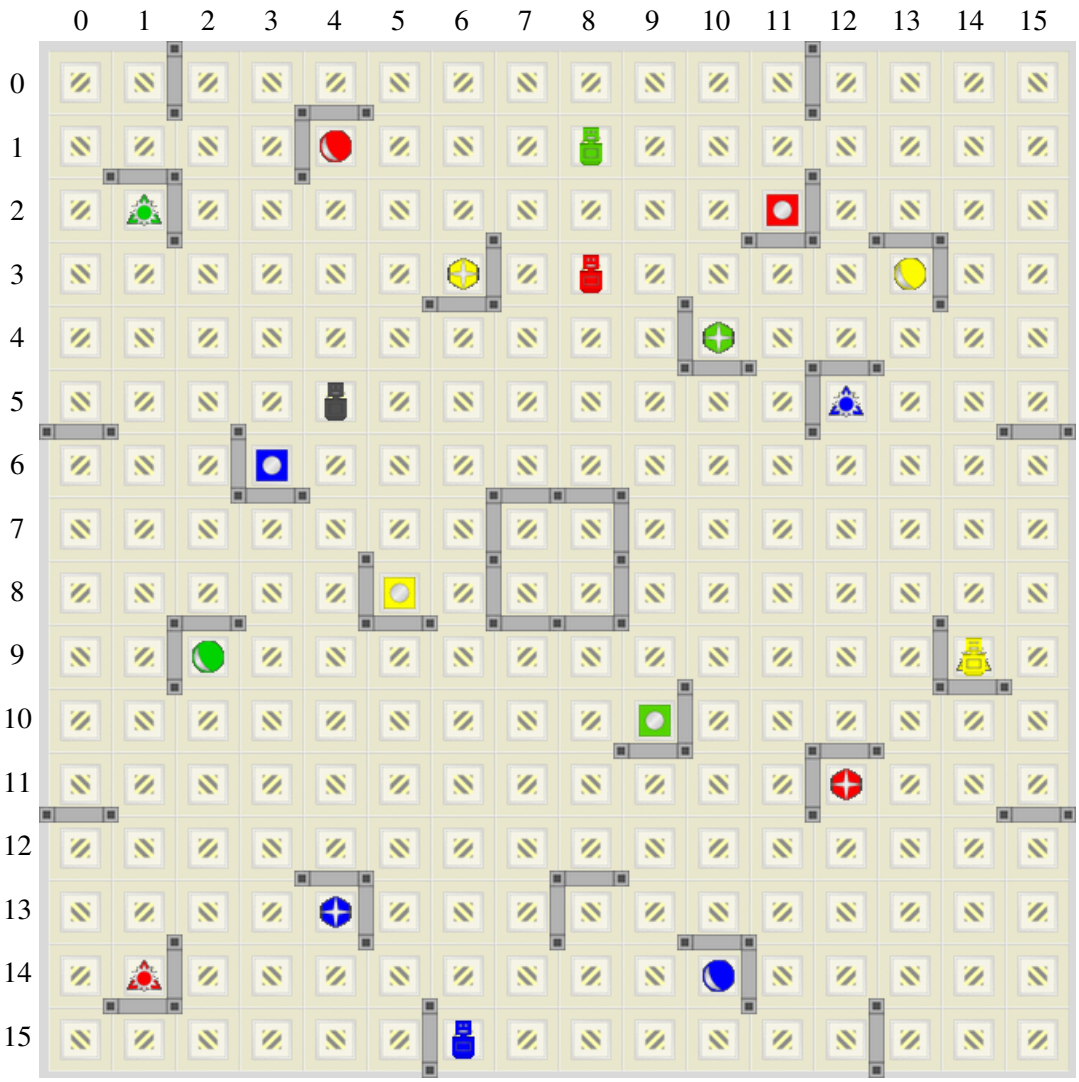

FIGURE 1 – Plateau de Ricochet Robots

1 Robots et directions

Le fichier `src/plateau.mli` contient la définition des types des robots et des directions.

▷ **Exercice 1** *Compléter la liste* tous_les_robots *(resp.* toutes_les_directions *) de façon à ce qu'elle contienne une instance de chaque robot (resp. direction).*

2 Position des robots

La position d'un robot est représentée par un couple d'entier *(abscisse, ordonnée)*. Nous souhaitons maintenant encoder l'ensemble des positions des robots et les fonctions d'accès, "modification" et comparaison associées.

▷ **Exercice 2** *Écrire un module* PosSimple, *conforme à l'interface* P *pour encoder les positions des robots. Remarques :*

- *Il y a exactement cinq robots.*
- *Le type* P.t *peut être défini à l'aide d'un n-uplet, de listes ou tout autre solution au choix.*
- *Attention, votre type doit garantir une représentation canonique des données. Par exemple, si vous utilisez des listes (sans invariant de type), le test d'égalité suivant*
  `[(Rrouge, (2, 3)); (Rvert, (4, 5))] = [(Rvert, (4, 5)); (Rrouge, (2, 3))]`
  retourne false *alors que* `[(Rrouge, (2, 3)); (Rvert, (4, 5))]` *et* `[(Rvert, (4, 5)); (Rrouge, (2, 3))]` *représentent la même position.*
  L'usage de listes (du moins sans invariant de type particulier) n'est donc pas une bonne solution.

3 Mouvements possibles

▷ **Exercice 3** *Écrire un foncteur* MakeNextPos : **functor** (Pos : P) -> NP **with type** t = Pos.t *avec les fonctions demandées pour calculer les mouvements possibles sur un plateau et à partir d'une position données. Ne pas hésiter à utiliser les fonctions sur les listes* List.map, List.filter *ou* List.flatten *par exemple.*

Pour tester vos fonctions, vous sont données :

- positions_sujet : la position des robots de la Figure **??**
- plateau_vide : un plateau vide de taille 16
- plateau_sujet : le plateau de la Figure **??**

4 Plus courtes solutions

Vous pouvez maintenant compiler le programme complet :

- se déplacer dans le répertoire parent (`cd ..`);
- compiler le programme entier avec `make`;
- puis l'exécuter avec `./orrobot`;
- vous pouvez alors essayer votre code (`C-c` ou `C-d` pour quitter le programme).

5 Implémentation plus efficace (BONUS)

▷ **Exercice 4** *On peut remarquer que les coordonées étant des paires d'entiers plus petit que 16, elles peuvent être encodées sur 8 bits. On peut donc encoder la position des cinq robots sur un unique entier (63 bits). Implémenter un module* PosMasque : P **with type** t = int *qui encode les positions des cinq robots dans un entier. On utilisera pour cela les opérations de décalage* lsr *et* lsl *ainsi que et, ou et ou exclusif bit à bit* land, lor *et* lxor *(toutes infixes).*