PP1-2 : Mini-projet - Fling

Amélie Ledein & Gabriel Hondet
`nom (at) lsv (dot) fr`

1 Objectif du mini-projet

Après les vacances de la Toussaint, vous allez devoir programmer en OCaml un compilateur du langage *C-* vers de l'assembleur *Intel x86*. L'objectif avec ce mini-projet est de vous habituer à utiliser des codes plus volumineux que ce que vous avez fait à présent. C'est aussi l'occasion de vérifier que vous savez tous programmer en OCaml.

Comme pour le compilateur que vous devrez programmer, pour ce mini-projet je vous fournis un squelette de code qu'il vous faudra alors compléter (les trous sont répérés par des `failwith "TODO"`). L'intérêt de vous fournir un squelette de code c'est bien évidemment de vous éviter de passer trop d'heures à tout réimplémenter. Cependant, si vous voulez refaire le mini-projet *from scratch*, c'est à votre guise !

2 Description du projet

Ce mini-projet vous demande d'implémenter un petit jeu qui s'appelle *Fling*. Fling fait parti de ces jeux dont le concept est simple, mais difficile à maîtriser. Dans ce jeu de réflexion, votre objectif va être de pousser des boules jusqu'à obtenir une seule boule sur le tableau à la fin. Votre but dans ce mini-projet va être de réimplémenter la *logique* du jeu ainsi qu'un petit solveur pour résoudre ce jeu automatiquement.

2.1 Les règles du jeu

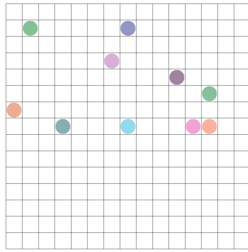
Le jeu commence avec n boules posées sur un quadrillage. A chaque tour de jeu, vous pouvez lancer une boule afin qu'elle se cogne dans une autre boule (un peu comme le billard). Le jeu se termine lorsqu'il reste qu'une seule boule ou bien lorsqu'il n'y a plus de coup possible. On va voir ci-dessous dans quels cas il est possible de lancer une boule.

Pour lancer une boule, il faut sélectionner la boule et une direction vers laquelle la lancer. Un tel coup est possible si et seulement si, il y a une boule sur son chemin qui soit à au moins une case de distance. Deux boules côte à côte ne peuvent pas se lancer l'une l'autre. Si le coup est valide, alors la boule va s'arrêter sur la case se trouvant avant la première boule qu'elle va toucher. Puis, cette dernière se retrouve propulsée à son tour, dans la même direction : soit elle sort du terrain et dans ce cas là le coup est fini, soit elle peut recogner une boule qui se retrouve à son tour propulsée et ainsi de suite. Pour y voir plus clair, la page suivante montre un exemple de partie.

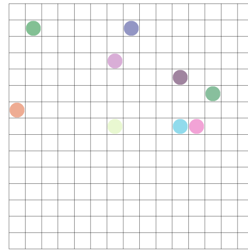
Pour ceux qui sont dotés d'un téléphone mobile sous Android ou iOS, vous pouvez télécharger l'application du même nom.

2.2 Tâches à faire

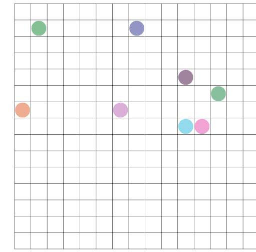
- Dans un premier temps, je vous demande à partir du squelette de code fourni de compléter les trous afin d'obtenir une version de Fling jouable.
- Puis, on verra comment avec un algorithme assez simple, il est possible de résoudre le jeu.
- Enfin, je vous propose diverses améliorations à implémenter à votre bon vouloir.



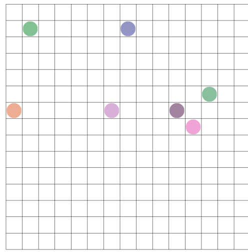
(a) Configuration initiale



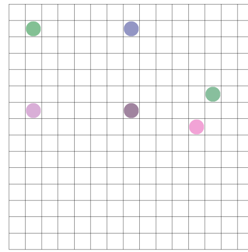
(b) Déplacement effectué : la boule bleu marine (devenue blanche) à droite



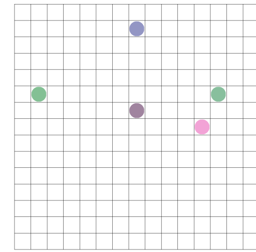
(c) Déplacement effectué : la boule mauve vers le bas



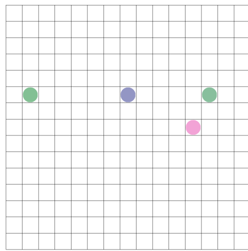
(d) Déplacement effectué : la boule violette vers le bas



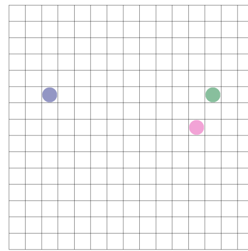
(e) Déplacement effectué : la boule violette à gauche



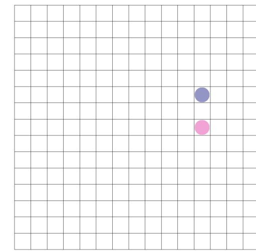
(f) Déplacement effectué : la boule verte foncée vers le bas



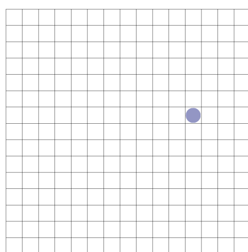
(g) Déplacement effectué : la boule bleu vers le bas



(h) Déplacement effectué : la boule bleu à gauche



(i) Déplacement effectué : la boule bleu droite



(j) Déplacement effectué : la boule bleu en bas

Exemple d'exécution d'une partie de Fling

3 Prise en main du squelette et compléter les trous

Vous trouverez le squelette de code à l'adresse suivante : `http://www.lsv.ens-cachan.fr/~ledein/fling.tar.gz`. Le code est divisé en plusieurs modules :

- le module *position* qui abstrait les coordonnées en 2 dimensions ;
- le module *rules* qui définit toutes les fonctions qui encodent les règles du jeu comme `apply_move` ;

- le module *game* qui définit le fonctionnement du jeu global comme choisir entre résoudre et jouer une partie de Fling ;
- le module *draw* qui définit toutes les fonctions nécessaires à l'affichage du jeu.

Pour compiler le code vous pouvez utiliser la commande `$ make` . Attention cependant, comme le code contient des trous, l'exécutable fourni va échouer lamentablement. Je vous invite à lire le code fourni pour comprendre comment le jeu fonctionne, et notamment comment y jouer. Le jeu utilise à la fois la souris et le clavier.
Question 0 : Compléter les types de `ball`, `move` et `game`.
Question 1 : Compléter le module `rules.ml` ainsi que la fonction `loop` dans `game.ml`.

4 Résoudre une partie de Fling

En jouant à Fling, vous aurez peut-être observé que dès qu'on dépasse un certain nombre de boules, la résolution d'une partie devient assez vite compliquée. Dans cette section, on va automatiser la résolution d'une partie. L'algorithme que je vous demande d'implémenter est juste bête et méchant, ce sera à vous ensuite de l'améliorer !

4.1 Les digraphes

Un digraphe $G = (V, E)$ est composé d'un ensemble de sommets V et d'un ensemble d'arcs $E \subset V \times V$. Pour résoudre une partie de Fling, on va considérer qu'un nœud correspond à une configuration du jeu, et qu'il y a un arc du nœud c_1 au nœud c_2 si la configuration c_2 est accessible depuis le nœud c_1 en **un seul** mouvement.

On note c_i la configuration initiale, C_f l'ensemble des configurations finales, c'est-à-dire celles contenant qu'une seule boule. L'objectif est donc de trouver un *chemin* dans un graphe qui parte de c_i et qui aille jusqu'à $c_f \in C_f$.

4.2 Parcours dans un graphe

Pour savoir s'il y a un chemin depuis c_i vers une configuration finale, il va falloir regarder tous les nœuds accessibles depuis c_i. Si parmis ces nœuds accessibles se trouvent une configuration finale, on a gagné et la solution consiste à retrouver le chemin.

Il existe deux algorithmes majeurs de parcours de graphe : le *depth-first search* et le *breath-first search*. Le premier consiste à aller le plus loin possible dans l'exploration des nœuds du graphe, tandis que le second consiste à regarder d'abord tous les nœuds à distance 1 du sommet initial, puis tous les nœuds à distance 2, et ainsi de suite. Ici, le *depth-first search* est l'algorithme qui nous intéresse. Je ne donne volontairement pas plus de détails afin que vous puissiez réfléchir à l'implémentation.
Question 2 : Compléter la fonction `solve` dans le fichier `solver.ml` dont le type est `game -> (move list) option`. La fonction renvoie `None` s'il n'y a pas de solution, ou bien `Some l` où `l` contient la liste des mouvements qui amènent à la solution finale.

5 Améliorations possibles

Dans cette section, je vous suggère quelques améliorations possibles. Dans l'évaluation, les améliorations compteront comme un bonus.

5.1 Générer des terrains

Vous avez dû remarquer que c'est pénible d'avoir à écrire à la main les configurations initiales. Il pourrait être intéressant, afin de tester votre algorithme, de générer automatiquement des terrains.
Question 3 : Créer un algorithme de génération bête qui tire n boules au hasard sur la map, teste si la configuration est valide, et sinon recommence.
Bonus : Si vous avez le temps essayez de regarder la probabilité qu'un tel terrain soit valide en fonction de la taille du terrain et du nombre de boules. Quand est-ce que cette probabilité est maximale ?

On va s'intéresser à générer des terrains qui sont toujours valides. Pour cela, une idée possible c'est de partir d'une configuration qui contient une seule boule. Puis de générer toutes les configurations qui permettent d'arriver à cette configuration. Et on recommence cette étape jusqu'à avoir le nombre de boules nécessaires.
Question 4 : Implémenter l'algorithme de génération ci-dessus. Lequel est le plus rapide ?

5.2 Raffiner l'algorithme de parcours de graphe

Question 5 : Améliorer le temps mis par votre solveur pour trouver une solution. Pour cela, vous pouvez essayer de trouver des critères qui, étant donnée une configuration, permettent à coup sûr de dire si cette configuration a une solution ou non. Il faut bien évidemment que ces critères soient rapides, sinon cela n'a aucun intérêt.

5.3 Entrées/sorties

Il pourrait être intéressant que votre programme puisse lire et écrire des configurations du jeu.
Question 6 : Inventer un format de fichier pour des configurations du jeu Fling.
Question 7 : Implémenter deux fonctions `open` et `save` qui permettent de lire et écrire des fichiers de configurations depuis ce format.

5.4 Un meilleur affichage

Question 8 : Visuellement, cela peut paraître brute que, pour un mouvement donné, nous affichons seulement la position de départ et la position d'arrivée. De même, juste dessiner un cercle pour une boule n'est pas très esthétique. Voilà deux pistes pour améliorer l'affichage, mais si vous en avez d'autres, n'hésitez pas.

5.5 Augmenter les dimensions

Il est possible d'étendre Fling à des dimensions supérieurs : 3D, 4D, etc.
Question 9 : Réimplémenter la seconde partie du projet, de telle sorte que vous soyez capable de résoudre des configurations Fling dans n dimensions. Même si cette amélioration n'a aucun intérêt *pratique*, elle permet de voir si votre code est extensible.

6 Evaluation

L'évaluation de ce mini-projet va se fera autour de 3 critères :

- La correction du code (est-ce que le programme fait bien ce qui est attendu ?).
- La propreté du code (notamment favoriser le fonctionnel au style impératif).
- La documentation du code que vous avez implémenté.

Correction du code : J'attends de vous que vous me donniez un code qui *compile*. Un code qui ne compile pas ne sera pas lu. Il faut aussi que votre code soit robuste. S'il se met à crasher sans raison ou à renvoyer une exception parce que l'utilisateur a fait un mouvement non valide ce n'est pas normal !

Propreté du code : Un code non commenté sera sévèrement sanctionné. Moins de temps je passerai à corriger votre code, meilleure sera votre note normalement (les plus audacieux pourront tenter de donner un code vide et de voir ce qui se passera...).

Documentation du code : Pensez toujours qu'une ligne de code est écrite 1 fois et lu 10 fois. Pour les personnes qui liront votre code ensuite, il est <u>nécessaire</u> que vous commentiez/documentiez votre code ! Même s'il est peu probable que 10 personnes liront votre code source pour ce mini-projet, je vous impose de faire comme si. Ne tombez pas non plus dans l'excès inverse. Trop de commentaires peut-être pire que pas assez de commentaires. À vous de trouver le juste milieu.

7 Modalités de rendu

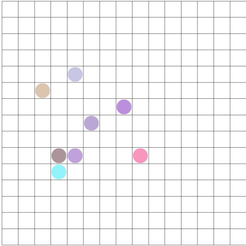
La date de rendu sera fixé ultérieument. Comptez environ 2 semaines. Votre rendu se composera d'une archive au format `tar.gz` qui une fois détarée doit créer un dossier `<NOM>_<PRENOM>` qui contiendra à l'intérieur le dossier `fling` contenant votre code ainsi qu'un fichier `README`.

7.1 Le fichier README

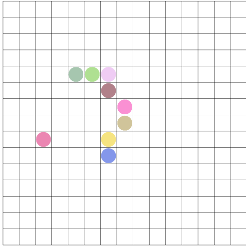
Ce fichier doit résumer en quelques lignes ce que vous avez fait (et ce que vous n'avez pas fait si vous n'avez pas fini). En particulier, précisez les améliorations implémentées. De plus, ce fichier doit expliquer succinctement vos choix d'implémentation (par exemple justifier les définitions de types). N'hésitez pas à parler des difficultés que vous avez rencontrées (une question mal posée/une question trop difficile, une ambiguïté dans le code, etc.). Vous pouvez aussi profiter de ce fichier pour me donner votre avis sur le projet et comment il serait possible de l'améliorer. Il me reste à vous souhaiter

Have fun !

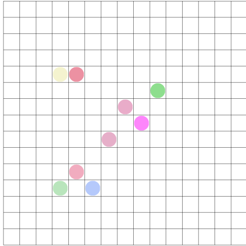
8 Annexe : Exemples de configurations ayant une solution



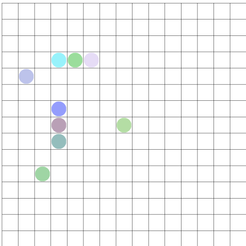
(k) Exemple 1



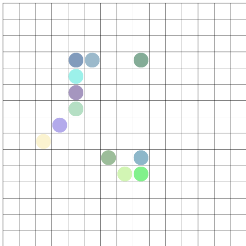
(l) Exemple 2



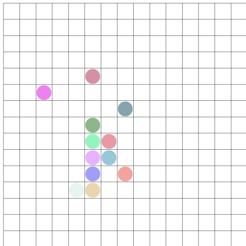
(m) Exemple 3



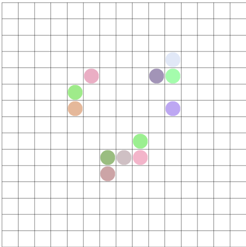
(n) Exemple 4



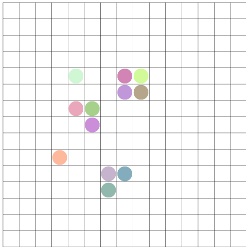
(o) Exemple 5



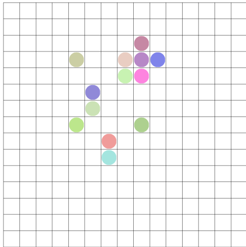
(p) Exemple 6



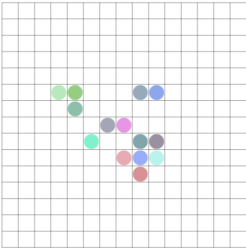
(q) Exemple 7



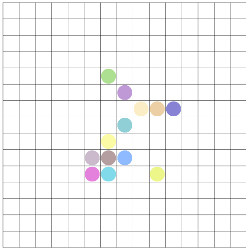
(r) Exemple 8



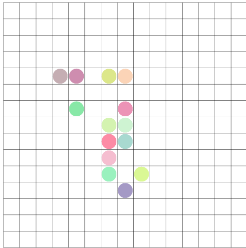
(s) Exemple 9



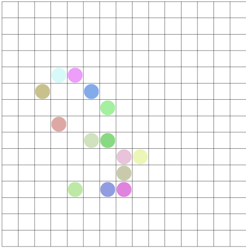
(t) Exemple 10



(u) Exemple 11



(v) Exemple 12



(w) Exemple 13