Rapport de projet

Marco Freire, Clément Legrand-Duchesne

26 septembre 2017

Résumé

Dans ce rapport seront expliqués les codes du pavage de Penrose et des tours de Hanoi, ainsi que leurs extensions ; et les choix d'implémentation que nous avons été menés à faire.

1 Penrose

1.1 Principe : algorithme de base

Le pavage de Penrose implémenté utilise deux tuiles de base, des triangles d'or. Les dimensions de leurs côtés sont donc des des nombres flottants. Afin d'éviter des erreurs d'arrondi suceptibles de ce cumuler, et d'alléger le code, nous avons choisi de ne travailler qu'avec des flottants, avant des les arrondir en entiers juste avant le tracé.

Par commodité, nous avons suivi la convention suivante : les triangles seront représentés par un tableau contenant les coordonnées des trois sommets, le premier étant celui duquel sont issus les deux cotés de longeurs égales, le second est le suivant en tournant dans le sens horaire.

Inserer image !

Afin de differentier les triangles obtus des triangles aigus, nous avons créé un type `triangle` :

```
type triangle = Obtuse | Acute;;
```

Une des principales difficultés rencontrées lors de l'implémentation de ce pavage est le fait la fonction `fill_poly` du module `Graphics.cma` remplit le polygone avec une couleur donnée, mais recouvre et donc efface les bords de celui ci si ceux ci ont été tracés avant. Cela n'est pas encore trop contraignant ici, puisqu'il suffit d'écrire la fonction `draw` de la sorte :

```
let draw points triangle =
    (if triangle = Obtuse then set_color red
    else set_color blue);
    fill_poly (iof_array points);
    move points.(0);
    set_color black;
    line points.(1);
    line points.(2);
    line points.(0)
;;
```

1.2 Améliorations

Le premier problème du code précédent est que les côtés des triangles sont tous tracés deux fois. Pour y remédier, il suffit, lors de chaque subdivision d'un triangle, de ne tracer que les séparations entre les nouveaux triangles obtenus (et de ne pas oublier les cotés du triangle initial).

Inserer une image ! (+ peuve ?)

Il apparaît alors qu'il est necéssaire de tracer ces lignes après que les triangles soient remplis, afin que celles ci ne soient pas effacées. La fonction `divide` est en réalité un simple parcours en profondeur de l'arbre des divisions des triangles. Plutôt que d'afficher directement le pavage, il semble plus interressant de montrer la division succéssive des triangles pendant la construction du pavage. Survient alors une nouvelle difficulté : l'algorithme de parcours en profondeur décrit précédemment ne convient plus. Une première solution est d'implémenter un parcours largeur et de marquer une pause avant d'entamer chaque nouvelle profondeur de l'arbre. il faut pour cela maintenir une structure de file En revanche, il est alors nettement plus difficile d'afficher les couleurs des triangles dessinés à chaque génération et de ne tracer chaque coté qu'une seule fois. En effet, à chaque nouvelle génération, la coloration des triangles efface les cotés de ceux ci si ils ont été précédemment tracés.

2 Hanoi

2.1 Principe : algorithme de base

Le problème des tours de Hanoi est essentiellement récursif. Pour déplacer n disques du premier poteau jusqu'au troisième il suffit de savoir en déplacer n-1 sur le deuxième poteau (étape 1), puis de déplacer le n-ième disque sur le troisième poteau (étape 2) et enfin de déplacer encore une fois les n-1 disques sur le troisième poteau (étape 3).

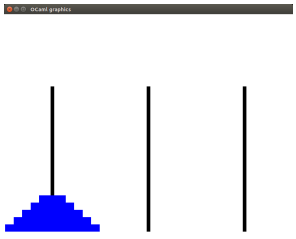

FIGURE 1 – État initial

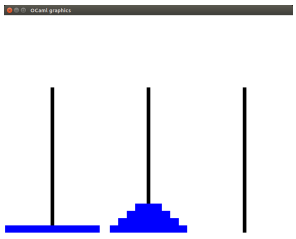

FIGURE 2 – Fin étape 1

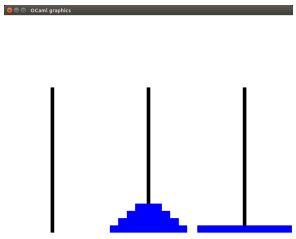

FIGURE 3 – Fin étape 2

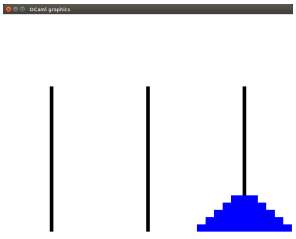

FIGURE 4 – État final

Nous avons choisi pour représenter la situation un tableau de piles `rods` : chaque pile du tableau représente chacun des poteaux et contient les disques qui y sont empilés.

La seule complication du code est le choix du poteau temporaire utilisé pour chaque déplacement de disques. La première implémentation réalisée repose sur la fonction suivante `choose` :

```
let choose a b =
  if a = b then failwith "a and b are equal"
  else if a = 0 then
        if b = 1 then 2
        else 1
```

```
  else if a = 1 then
          if b = 0 then 2
          else 0
  else if a = 2 then
          if b = 0 then 1
          else 0
  else failwith "a or b are not between 0 and 2"
;;
```

Cette fonction prend en argument deux éléments distincts de 0, 1, 2 et renvoie le troisième et est utilisée pour choisir automatiquement dans la fonction `move` le poteau temporaire à utiliser :

```
let rec move rods num_discs orig_rod dest_rod =
  if num_discs = 1 then
          move_disc rods orig_rod dest_rod
  else
          (
          let temp_rod = choose orig_rod dest_rod in
          move rods (num_discs - 1) orig_rod temp_rod;

          move_disc rods orig_rod dest_rod;

          move rods (num_discs - 1) temp_rod dest_rod;
          )
;;
```

Lors de l'implémentation de la fonction résolvant le problème des tours de Hanoi à n poteaux, nous nous sommes rendu compte que la fonction `choose` était difficilement généralisable. Nous avons donc choisi de passer le poteau intermédiaire en argument à la fonction `move` :

```
let rec move rods num_discs orig_rod dest_rod temp_rod=
  if num_discs = 1 then
          move_disc rods orig_rod dest_rod
  else
          (
          move rods (num_discs - 1) orig_rod temp_rod dest_rod;

          move_disc rods orig_rod dest_rod;

          move rods (num_discs - 1) temp_rod dest_rod orig_rod;
          )
;;
```

Cette modification permet de choisir comme l'on veut le poteau intermédiaire, ce qui est nécessaire pour la version généralisée de l'algorithme.

2.2 Principe : algorithme généralisé

Pour ce problème, l'implémentation choisie est la deuxième présentée.

Cette fois il s'agit d'utiliser au mieux les poteaux supplémentaires. Nous avons calculé le nombre de disques pouvant être sur les poteaux intermédiaires, pour répartir ces disques sur chacun des poteaux intermédiaires en utilisant comme poteau temporaire le dernier (étape 1), ensuite il faut déplacer le disque le plus grand sur le dernier poteau (étape 2), et finalement il

suffit de réaliser le parcours inverse de celui effectué dans l'étape 1, et de regrouper les disques intermédiaires sur le dernier poteau (étape 3).

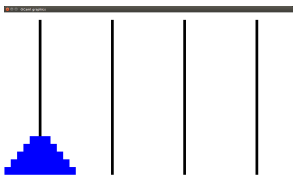

FIGURE 5 – État initial

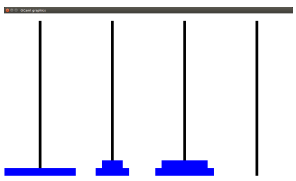

FIGURE 6 – Fin étape 1

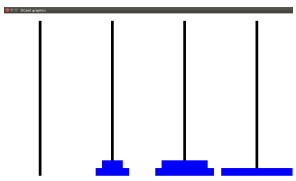

FIGURE 7 – Fin étape 2

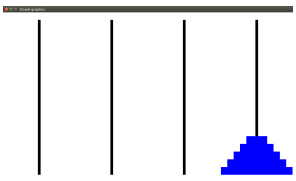

FIGURE 8 – État final

Ainsi beaucoup moins de déplacements sont nécessaires pour la résolution du problème.