Projet : Détection de collision en 2D à l'aide d'un QuadTree

Consignes :

- Lisez bien tout le sujet avant de commencer à coder.
- Envoyez une archive, contenant votre code (fichiers `.ml`, `.mli`, etc.) et un rapport au format `PDF`, par email avant le mercredi **15 avril à 23h59** à `christophe.mouilleron@ensiie.fr` .
- Pensez à commentez votre code. Au minimum, chaque déclaration de fonction devra être précédée d'un commentaire expliquant ce que la fonction est censée faire.

L'objectif de ce projet est de simuler le déplacement d'un objet dans un plan, et de déterminer d'éventuelles collisions avec d'autres objets. Pour se faire, nous allons définir et utiliser une structure de donnée qui s'appelle le QuadTree.

1 Échauffement sur les rectangles

On considère les types OCaml suivants :

```
type coord = float * float ;;
type rect  = coord * coord ;;
```

Le type `rect` sera utilisé pour représenter les rectangles dont les cotés sont soit horizontaux, soit verticaux. On impose en plus que la première coordonnée (accessible grâce à `fst`) correspond au coin supérieur gauche. Ainsi, la seconde coordonnée (accessible grâce à `snd`) est celle du coin inférieur droit.

Question 1. Écrire une fonction `make_rect: coord -> coord -> rect` qui, sur la donnée des coordonnées de deux points A et B, retourne le rectangle qui admet A et B comme coins opposés.

Par exemple, `make_rect (0.0, 0.0) (1.0, 1.0)` devra retourner `( (0.0, 1.0), (1.0, 0.0) )`.

Question 2. Écrire les fonctions `rect_length` et `rect_height`, renvoyant respectivement la longueur et la hauteur du rectangle passé en argument.

Question 3. Écrire une fonction `rect_mem: rect -> coord -> bool` qui, sur la donnée d'un rectangle et des coordonnées d'un point A, renvoie `true` si A est à l'intérieur du rectangle et `false` sinon.

Question 4. Écrire une fonction `rect_split: rect -> coord -> rect * rect * rect * rect` qui, sur la donnée d'un rectangle $ABCD$ et des coordonnées d'un point P, renvoie les quatre sous-rectangles obtenus à partir de P et des points A, B, C et D.

2 QuadTree – Première époque

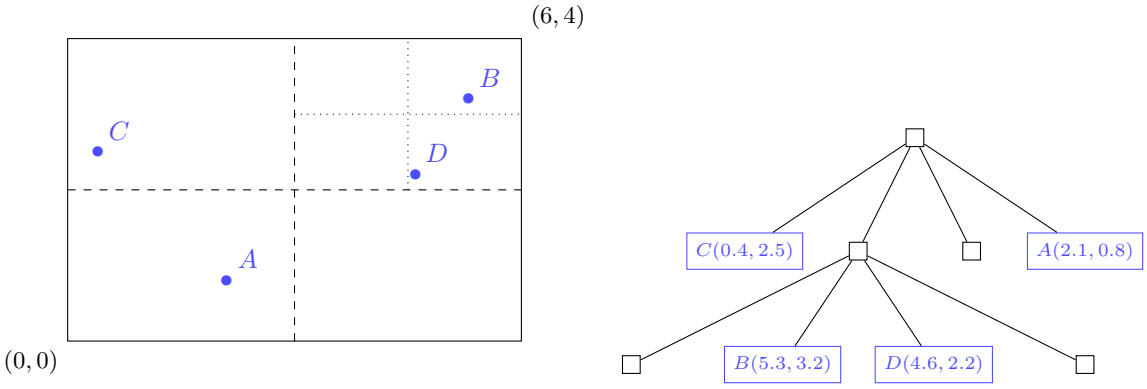
Dans cette partie, on va se concentrer sur un cas simple : le déplacement d'un palet sur une planche avec des clous [1]. La scène est donc un ensemble de clous (objets ponctuels immobiles) dans lequel on fait se déplacer le palet (disque). L'objectif est de représenter cette scène à l'aide d'une structure adaptée, puis d'utiliser cette représentation pour détecter l'éventuelle collision entre le palet et un des clous.

2.1 La structure de données QuadTree

Un QuadTree est une structure de données permettant de faire de l'indexation spaciale. L'idée générale est de paver le plan de rectangles, de telle sorte que chaque rectangle contienne 0 ou 1 objet ponctuel. En l'absence d'objets, la scène est vue comme un grand rectangle vide. On va ensuite ajouter les clous un par un. Si deux clous se retrouvent dans le même rectangle, ce dernier est divisé en quatre rectangles de même taille, et on continue ainsi tant qu'il y a plus qu'un clou dans un des rectangles.

1. Penser au jeu du Fakir.

Le QuadTree est alors un arbre dont les feuilles sont les rectangles pavant le plan, et dont les nœuds internes correspondent aux rectangles qu'on a du diviser en quatre. Les nœuds internes du QuadTree ont donc toujours quatre fils. Un exemple de scène et du QuadTree associé est présenté à la figure 1.



(a) Scène 2D contenant 4 objets ponctuels.

(b) QuadTree représentant la scène.

FIGURE 1 – Exemple de l'utilisation d'un QuadTree pour représenter une scène 2D.

Pour la suite, nous allons utiliser la définition de type suivante :

```
type 'a quadtree = rect * 'a cell
and 'a cell =
    | Empty
    | Leaf of coord * 'a
    | Node of 'a quadtree * 'a quadtree * 'a quadtree * 'a quadtree
;;
```

Ce type ressemble au type vu en cours pour les arbres binaires, à ceci près que les nœuds interne ont cette fois quatre fils et que nous allons conserver à chaque étage de l'arbre le rectangle correspondant. Concernant les fils, on prendra soin d'ordonner les quatre sous-rectangles de Nord-Ouest à Sud-Ouest, dans le sens des aiguilles d'une montre, comme cela a été fait dans l'exemple de la figure 1.

Question 5. Discuter des avantages et des inconvénients d'utiliser la structure de QuadTree pour représenter la scène, plutôt que d'autres structures comme celle d'ensemble.

Question 6. Écrire une fonction `cardinal` qui, sur la donnée d'un QuadTree, retourne le nombre d'objets présents.

Question 7. Écrire une fonction `list_of_quadtree: 'a quadtree -> ('a, coord) list` qui prend un QuadTree en argument et retourne la liste des objets qu'il contient.

Question 8. Écrire une fonction `insert: 'a quadtree -> coord -> 'a -> 'a quadtree` qui, sur la donnée d'un QuadTree, des coordonnées, et du nom d'un objet, renvoie un nouveau QuadTree dans lequel l'objet a été ajouté.

Question 9. Écrire une fonction `quadtree_of_list: ('a, coord) list -> 'a quadtree`, permettant de construire le QuadTree à partir de la liste des objets d'une scène.

Question 10. Écrire une fonction `remove: 'a quadtree -> coord -> 'a quadtree` qui, sur la donnée d'un QuadTree et des coordonnées d'un objet, retourne un nouveau QuadTree privé de cet objet.

note : Enlever un objet peut rendre une subdivision en quatre sous-rectangle inutile. Il convient dans ce cas de simplifier au maximum le QuadTree pour des raisons évidentes d'efficacité.

2.2 Représentation graphique d'un QuadTree et tests

Question 11.

Question 12.

2.3 Déplacement du disque et détection de collision

Question 13.

Question 14.

Question 15.

Question 16.

3 QuadTree – Deuxième époque

Pour cette troisième partie, on va s'intéresser au cas plus général où les objets de la scène ne sont plus des points mais des rectangles (dont les cotés sont toujours horizontaux ou verticaux).

Pour ce faire, ...

Question 17. Proposer une version modifiée du code de la partie précédente afin de gérer le cas d'objets rectangulaires. Fournir un jeu de test pour valider le nouveau code.

Question 18. Proposer une généralisation au cas d'objets de forme quelconque. Expliquer sous quelle(s) condition(s) votre approche est correcte et efficace.