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. Nous allons nous 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). Dans un premier temps, il va falloir représenter cette scène à l'aide d'une structure adaptée appelée QuadTree. Ensuite, nous utiliserons cette représentation pour détecter l'éventuelle collision entre le palet et les clous.

1 Échauffement sur les rectangles

On considère les types OCaml suivants :

```
type coord = float * float ;;
type rect  = R of 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 correspond au coin supérieur gauche. Ainsi, la seconde coordonnée 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 `R ((0.0, 1.0), (1.0, 0.0))`.

Question 2. Écrire les fonctions de projection suivantes :
1. `rect_left` renvoyant l'abscisse du coté gauche d'un rectangle,
2. `rect_right` renvoyant l'abscisse du coté droit d'un rectangle,
3. `rect_bottom` renvoyant l'ordonnée du coté inférieur d'un rectangle,
4. `rect_top` renvoyant l'ordonnée du coté supérieur d'un rectangle.

Question 3. Écrire les fonctions `rect_length` et `rect_height`, renvoyant respectivement la longueur et la hauteur du rectangle passé en argument.

Question 4. 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 5. Écrire une fonction `rect_intersect: rect -> rect -> bool` qui, sur la donnée de deux rectangles r_1 et r_2, renvoie `false` si l'intersection de r_1 et r_2 est vide et `true` sinon.

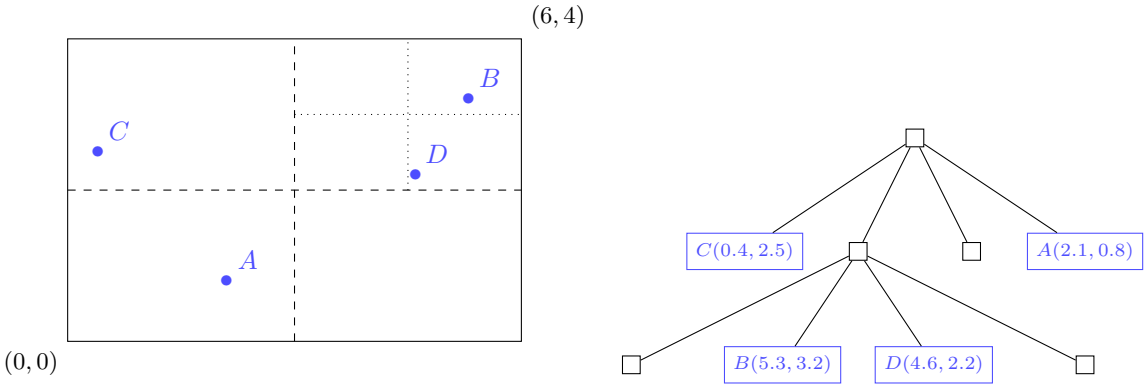
Question 6. Écrire une fonction `rect_split: rect -> rect * rect * rect * rect` qui, sur la donnée d'un rectangle `r`, renvoie les quatre sous-rectangles de `r` formés à partir du centre de `r`.

2 La structure de données QuadTree

Un QuadTree est une structure de données permettant de faire de l'indexation spatiale. 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 =
    | Q of 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. La valeur de type `'a` dans les feuilles sera utilisée pour donner un nom aux différents objets. Enfin, 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 7. 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 8. Écrire une fonction `boundary` qui, sur la donnée d'un QuadTree, retourne le rectangle dans lequel s'inscrit ce QuadTree.

Question 9. Écrire une fonction `cardinal` qui, sur la donnée d'un QuadTree, retourne le nombre d'objets présents.

Question 10. Écrire une fonction `list_of_quadtree: 'a quadtree -> (coord, 'a) list` qui prend un QuadTree en argument et retourne la liste des objets qu'il contient.

Question 11. 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 12. Écrire une fonction `quadtree_of_list: (coord, 'a) list -> 'a quadtree`, permettant de construire le QuadTree à partir de la liste des objets d'une scène.

Question 13. 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-rectangles inutile. Il convient dans ce cas de simplifier au maximum le QuadTree pour des raisons évidentes d'efficacité.

3 Représentation graphique d'un QuadTree et tests

Nous allons à présent utiliser le code fourni dans le fichier [2] `display.ml` afin d'obtenir une représentation graphique de QuadTree. En supposant que `my_qt` est de type `int quadtree`, l'affichage se fait en ajoutant le bloc suivant à la fin de votre code :

```
#use "display.ml" ;;
simple_test my_qt string_of_int ;;
```

Pour que cela fonctionne, vous devez charger explicitement le module `Graphics`, ce qui peut être fait :
- soit dans `Emacs` en mettant `ocaml graphics.cma` au lancement de l'interpréteur,
- soit en ajoutant l'argument `graphics.cma` à votre ligne de commande si vous utilisez `ocaml` ou `ocamlc`.

Question 14. Dans votre rapport, expliquez avec le plus de détails possible ce que font les différentes fonctions présentes dans le fichier `display.ml`, et comment elles le font.

note : N'hésitez pas à consulter la documentation en ligne du module `Graphics` si besoin.

Question 15. Adapter la fonction `simple_test` afin d'illustrer le bon fonctionnement des fonctions `insert`, `quadtree_of_list` et `remove` codées précédemment. On veillera à tester plusieurs cas de figures dont la pertinence devra être discutée dans le rapport.

bonus : Pour tester plus intensivement la fonction `insert`, on pourra écrire une fonction reposant sur `Random.float` et générant aléatoirement un QuadTree avec une centaine d'objets.

Question 16. On aurait pu faire appel aux fonctions `rect_length` et `rect_height` dans le code de la fonction `draw_quadtree`. Essayez de modifier ce code afin et de faire un test afin de constater que ceci n'est pas une bonne idée. Quel est le problème ?

4 Placement du disque

Nous disposons maintenant de la structure de QuadTree pour représenter la scène. Cette partie est consacrée au placement d'un disque (immobile pour le moment) et à la vérification que celui-ci n'est pas déjà en collision avec les objets présents sur la scène.

Question 17. Écrire une fonction `collision_disk_point: coord * float -> coord -> bool` qui, sur la donnée du centre (x_c, y_c) et du rayon r d'un disque, et des coordonnées (x, y) d'un point, retourne `true` si le point est contenu dans le disque et `false` sinon.

rappel : Une équation pour le disque est $(x - x_c)^2 + (y - y_c)^2 \leq r^2$.

Question 18. Écrire une fonction récursive `clip: 'a quadtree -> rect -> 'a quadtree` basée sur l'algorithme 1. Un exemple de fonctionnement de cette fonction est présenté à la figure 2.

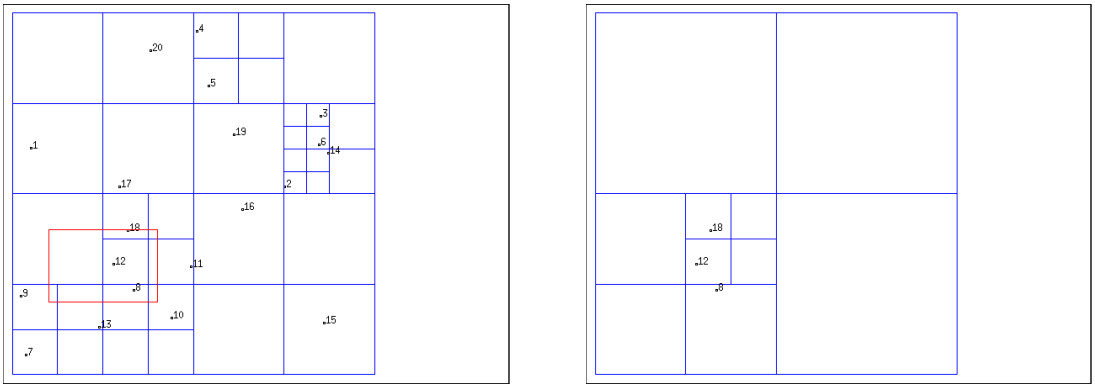
Algorithme 1 : `clip`

Entrée : un QuadTree $q = Q(r_q, c_q)$ et un rectangle r
Sortie : un QuadTree correspondant aux éléments de q contenu dans r

si $r_q \cap r = \emptyset$ **alors retourner** $Q(r_q, \texttt{Empty})$
sinon si $c_q = \texttt{Empty}$ *ou* $c_q = \texttt{Leaf}((x, y), e)$ *avec* $(x, y) \in r$ **alors retourner** q
sinon si $c = \texttt{Leaf}((x, y), e)$ *avec* $(x, y) \notin r$ **alors retourner** $Q(r_q, \texttt{Empty})$
sinon
    Extraire les quatre fils de $c_q = \texttt{Node}(nw, ne, se, sw)$
    $nw' \leftarrow \texttt{clip}(nw)$; $ne' \leftarrow \texttt{clip}(ne)$; $se' \leftarrow \texttt{clip}(se)$; $sw' \leftarrow \texttt{clip}(sw)$
    $q' \leftarrow Q(r, \texttt{Node}(nw', ne', se', sw'))$
    retourner q' simplifié au maximum

Question 19. Écrire une fonction `collision_disk: 'a quadtree -> (coord * float) -> (coord * 'a) list` qui, sur la donnée d'un QuadTree représentant une scène et d'un disque, retourne la liste des points de la scène appartenant au disque.

2. Téléchargeable depuis la page `http://chadok.info/~cmouille/teaching/IPF_FIP/projet_2014/`



(a) Scène dont on veut la restriction au rectangle en rouge. (b) Résultat obtenu grâce à un appel à `clip`.

FIGURE 2 – Exemple d'appel à la fonction `clip`.

note : Commencer par appeler `clip` sur un carré bien choisi contenant l'intégralité du disque.

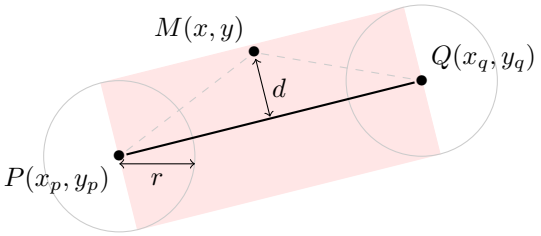
Question 20. Le fichier[3] `simulation1.ml` fournit une fonction `simulation_placement` permettant de tester le code des questions précédentes. Décrire le fonctionnement de cette fonction et des fonctions auxiliaires qui sont appelées, puis vérifier que le placement du disque s'effectue correctement.

5 Déplacement du disque et détection de collision

Dans cette dernière partie, nous allons simuler le déplacement du disque, que l'on suppose initialement à une position sans collision. La détection d'une collision lors du déplacement demande deux choses :

1. Vérifier qu'il n'y a pas de collision à l'arrivée, ce qui est faisable avec le code de la partie 4.
2. Vérifier qu'il n'y a pas d'objet dans la surface balayée lors du déplacement.

La situation pour le deuxième point est illustrée par la figure 3.



$$d = \frac{|(y_q - y_p)\,x - (x_q - x_p)\,y + x_q y_p - y_q x_p|}{\sqrt{(x_q - x_p)^2 + (y_q - y_p)^2}} \overset{?}{\leq} r$$

$$\overrightarrow{PM} \cdot \overrightarrow{PQ} \overset{?}{\geq} 0 \qquad \overrightarrow{QM} \cdot \overrightarrow{QP} \overset{?}{\geq} 0$$

FIGURE 3 – Surface balayée par le disque lors d'un déplacement de P à Q et test de collision.

Question 21. Écrire une fonction `collision_trail_point: coord -> coord -> float -> coord -> bool` qui, sur la donnée de deux points P et Q, d'une distance r, et d'un point M, retourne `true` si M est dans la bande de rayon r centrée sur le segment $[PQ]$ et `false` sinon.

Question 22. Écrire une fonction `collision_trail: 'a quadtree -> coord -> coord -> float -> (coord * 'a) list` qui, sur la donnée d'un QuadTree, des coordonnées de départ et d'arrivée, et du rayon du disque, retourne la liste des objets du QuadTree appartenant à la surface balayée lors du déplacement du disque.

Question 23. Testez votre code à l'aide de la fonction `simulation_move` gracieusement fournie par le fichier[3] `simulation2.ml`. Pensez aussi à décrire le contenu de ce fichier dans votre rapport.

bonus : On pourra aussi s'inspirer de `simulation_move` afin d'écrire une fonction récursive qui redemande une nouvelle destination et effectue le déplacement tant qu'il n'y a pas eu de collision.

3. Disponible au même endroit que `display.ml`.