Algorithme A* sur une grille

On considère une grille rectangulaire de $n \times m$ cases qui sera représentée en `OCaml` par une matrice d'entiers :

```
type grille = int array array;;
```

Cette grille peut contenir plusieurs type de cases :

- Une case contenant 0 est une case vide.
- Une case contenant 9 est une case infranchissable.
- Une case contenant 1 servira à représenter un sommet ouvert dans l'algorithme A^*.
- Une case contenant 2 servira à représenter un sommet fermé dans l'algorithme A^*.
- Une case contenant 5 servira à indiquer le chemin solution en fin d'algorithme A^*.

La case $(0, 0)$ sera notre case de départ et le but est d'atteindre la case $(n-1, m-1)$.

1 La grille

Question 1

Écrire une fonction
`creer_grille : int -> int -> int -> grille`
où `creer_grille n m k` créé une grille de taille $n \times m$ contenant k cases infranchissables réparties aléatoirement. Les cases de départ et d'arrivée seront vides.

INDICATIONS

Un entier aléatoire entre 0 et $n-1$ peut être obtenu avec l'expression `Random.int n`. Il faudra en début de fichier initialiser le générateur aléatoire avec `Random.self_init ();;`.

Dans le TP on commencera par tester les fonctions sur des petites grilles avec peu d'obstacles.

On pourra afficher la grille en mode texte avec une fonction telle que :

```
let print_grille g =
    let n = Array.length g in
    let m = Array.length g.(0) in
    for i = 0 to n-1 do
        for j = 0 to m-1 do
            match g.(i).(j) with
```

```
            | 0 -> print_char '.'
            | 9 -> print_char 'X'
            | _ -> ()
        done;
        print_newline ()
    done;;
```

Question 2

Écrire une fonction
`voisins : grille -> int * int -> (int * int) list`
retournant la liste des cases voisines d'une case. Les déplacements possibles à partir d'une case sont : haut, bas, gauche, droite. Les cases infranchissables ne comptent pas comme voisins.

2 File de priorité

Par simplicité et gain de temps, la file de priorité sera codée sous forme de liste modifiable de couples $(element, valeur)$ triée par ordre croissant de valeur.

```
type 'a fileprio = ('a * int) list ref;;
```

Question 3

Écrire les fonctions d'interface de la file de priorité :

```
creer_file : unit -> 'a fileprio
est_vide : 'a fileprio -> bool
inserer : 'a -> int -> 'a fileprio -> unit
extraire : 'a fileprio -> ('a * int)
diminuer : 'a -> int -> 'a fileprio -> unit
```

3 Heuristique

Pour la fonction heuristique, on utilisera la distance de Manhattan :

$$d(\ (x,y),\ (a,b)\ ) = |x-a| + |y-b|$$

Question 4

Écrire la fonction distance de Manhattan :
`manhattan : int * int -> int * int -> int`

Dans la suite l'heuristique utilisée sera donc $h(x, y) = d((x, y), (n-1, m-1))$.

4 Algorithme A*

On introduit le type suivant qui sera expliqué dans la suite :

```
type predecesseur = Gauche | Droite | Haut | Bas | Aucun;;
```

Question 5

Écrire une fonction
`astar : grille -> ((predecesseur array array)`
exécutant l'algorithme A* sur la grille en la modifiant par effet de bord à chaque itération. Cette fonction ne calculera pas tout de suite le chemin optimal. Elle retourne une matrice indiquant pour chaque case, comment se déplacer pour atteindre la classe qui la précède dans l'arborescence construite par $A*$.

Question 6

Écrire une fonction
`reconstruire : grille -> predecesseur array array -> unit`
qui modifie la grille par effet de bord et qui reconstruit un chemin optimal menant de la case de départ à la case d'arrivée.

5 Affichage graphique

Pour les affichages graphiques, on n'hésitera pas à construire des grilles intéressantes.

Représenter graphiquement la solution obtenue à l'aide du module `OCaml Graphics`. Voici un exemple d'utilisation de ce module :

```
#load "graphics.cma";;
(* ouverture de la fenetre *)
Graphics.open_graph " 700x400+50-0";;
```

```
Graphics.set_window_title "A star";;

(* format couleur  0xRRGGBB *)
let rouge = 0xFF0000;;
Graphics.set_color rouge;;

(* Tracer un point avec la couleur actuelle *)
Graphics.plot x y;;

(* Tracer une ligne *)
Graphics.moveto xa ya;;
Graphics.lineto xb yb;;

(* Pour fermer la fenetre a la fin *)
Graphics.close_graph ();;
```

En s'y prenant correctement vous pouvez avec ces commandes produire un programme qui affiche en temps réel le calcul de A^*.