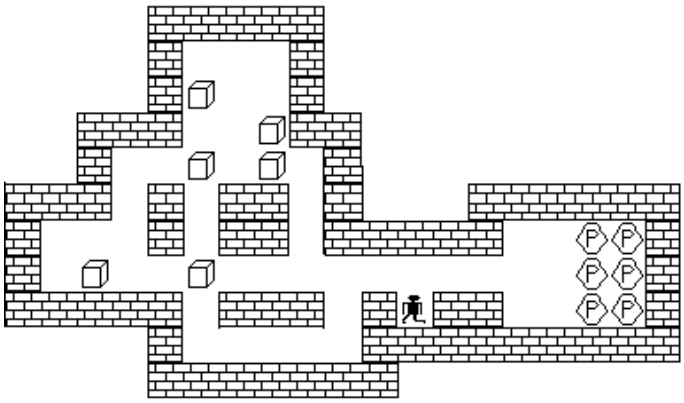
Feuille d'exercices n°19 de l'option d'informatique.
(à réaliser sur 2 séances)

Quelques conseils : lisez soigneusement l'énoncé dans sa totalité avant de commencer votre recherche. Vous devez impérativement respecter les notations de l'énoncé.

L'espace de travail

On se propose d'étudier divers algorithmes utilisés pour la réalisation d'un jeu, fonctionnant selon le principe suivant. Le héros (petit personnage dans le bas du tableau) doit pousser à travers un labyrinthe les six caisses (situées sur la gauche) pour les amener sur les six garages (marqués d'un P cerclé, sur la droite). On notera que l'on peut disposer n'importe quelle caisse sur n'importe quel garage.



Notre héros doit respecter les règles suivantes : il ne peut pousser qu'une caisse à la fois; il ne peut pas tirer une caisse. Il ne peut pas traverser les murailles.

Le labyrinthe sera représenté par un tableau Caml `t` de 19 cases par 11 (observer attentivement le dessin). Chaque case est, soit libre, soit occupée par un mur (inamovible), une caisse à ranger, un garage vide, une caisse rangée sur un garage, ou notre héros lui-même (qui peut d'ailleurs se trouver sur un garage). Les lignes et colonnes étant numérotées à partir de 0, la case située en haut et à gauche a pour coordonnées (0, 0) et la case où se trouve actuellement notre héros a pour coordonnées (8, 11) : le premier indice est celui de la ligne, comme à l'accoutumée.

Les quatre mouvements possibles sont des déplacements d'une case vers le haut, vers le bas, vers la gauche, ou vers la droite. On définit les types suivants :

```
type case = Libre | Mur | Caisse | Garage | Caisse_garée | Héros | Héros_garé;;
type mouv = Nord | Sud | Ouest | Est;;
```

Le tableau `t` sera défini par :

```
let nl = 11 and nc = 19;;
let t = make_matrix nl nc Libre;;
```

et on supposera qu'il a été initialisé de façon à représenter le labyrinthe `t`, par exemple, `t.(5).(2)` a pour valeur `Mur`, et `t.(7).(5)` a pour valeur `Caisse`.

On notera que si le héros, arrivé en case `(7,6)`, fait mouvement vers la gauche, il pousse une caisse, qui se retrouve en case `(7,4)`, et lui-même se retrouve en case `(7,5)`.

Première partie: quelques algorithmes élémentaires

1. Écrire une fonction **laby_valide** qui indique si le labyrinthe proposé est valide, *id est* : il y a un et un seul héros; il y a autant de caisses que de garages.
2. Écrire une fonction **place_héros** qui détermine le couple (i,j) tel que `t.(i).(j)=Héros`.

3. Écrire une fonction **print_laby** qui affiche le tableau en mode «texte»: on utilisera print_char et print_newline. On adoptera la convention suivante pour représenter les différents types de cases

H	le héros
h	le héros sur un garage
C	une caisse
G	un garage inoccupé
c	une caisse sur un garage
M	un mur
espace	une case vide

4. Écrire une fonction **laby_fini** qui indique si notre héros a terminé son travail, *id est* : toutes les caisses sont rangées sur les garages.

Deuxième partie: déroulement de la partie

5. Écrire une fonction **mouv_licite** qui, appliquée à une variable de type `mouv`, rend un résultat de type `bool` indiquant si le mouvement est licite. Par exemple, le premier mouvement licite du héros est nécessairement Nord; ensuite, il a le choix entre Ouest, Est et Sud (ce dernier indiquant une certaine indécision de sa part).

6. Écrire une fonction **liste_mouv_licites** qui construit la liste des mouvements autorisés. Pourquoi cette liste n'est-elle jamais vide?

7. Écrire une fonction **déplace** qui, appliquée à une variable de type **mouv**, calcule le nouvel état du tableau après le mouvement du héros; cette fonction rendra un résultat de type **unit**, puisqu'elle doit procéder par *effets*..

S'il vous reste moins de heure pour terminer, passez directement à la question 14
Le joueur guide le héros dans le labyrinthe avec les touches fléchées du clavier: haut, bas, gauche, droite. Celles-ci sont aussi gravées sur les touches 8, 2, 4 et 6 respectivement du pavé numérique. Il souhaite utiliser les touches 7, 9, 3 et 1 pour des déplacements «en biais»

	=	/	*
7	8	9	-
4	5	6	+
1	2	3	
0		.	

On redéfinit donc le type mouv:
`type mouv = Nord | Sud | Ouest | Est | NO | NE | SE | SO`

8. Écrire une fonction **mouv_gen_licite** qui, appliquée à une variable de type **mouv**, rend un résultat de type **bool** indiquant si le mouvement est licite. On notera bien qu'un mouvement «en biais» doit être décomposé en deux mouvements consécutifs, l'un horizontal, l'autre vertical; et qu'aucun de ces mouvements ne doit nécessiter le déplacement d'une caisse.

9. Justifier la restriction apportée à la fin de la question précédente.

Troisième partie: pilotage du héros à la souris

Vous constatez qu'une souris est reliée à votre ordinateur; vous décidez donc d'en prévoir l'utilisation, selon le principe suivant - en cliquant sur une case du tableau, le héros se déplacera par le plus court chemin menant à cette case (en ne passant que par des cases libres ou des garages inoccupés), si toutefois un tel chemin existe.

10. Écrire une fonction **accès_laby** qui construit un tableau d'entiers `tacc` défini ainsi: `tacc.(l).(c)` vaut n s'il existe un chemin de longueur n menant de la case `(i,j)` où se trouve le héros à la case `(l,c)`, et s'il

n'en existe pas de plus court; s'il n'existe pas de tel chemin, on donnera à `tacc.(l).(c)` la valeur 999999. Indication: `tacc.(i).(j)` vaut 0, les cases libres adjacentes vaudront 1, les cases libres adjacentes à une case marquée 1 et non marquées 0 vaudront 2, et ainsi de suite.

11. Donnez une estimation du coût de l'algorithme que vous venez d'écrire, en fonction du nombre N de cases du tableau.

12. Quelles améliorations *simples* proposeriez-vous d'apporter à l'algorithme précédent pour en diminuer le coût ?

13. Écrire une fonction **chemin** qui, à partir du tableau **tacc** obtenu précédemment, construit (si la chose est possible) une liste de type mouv list, contenant la liste des mouvements «élémentaires» amenant le héros de sa case actuelle `(i,j)` à la case visée `(l,c)`. Remarque: ici encore, le cheminement du héros doit se faire sans qu'il doive pousser une seule caisse.

Quatrième partie: outils complémentaires

14. Le labyrinthe est stocké dans un fichier de nom **laby**, contenant onze chaînes de caractères de longueur 19. Écrire une fonction **lire_laby** qui ouvre ce fichier, lit son contenu, et initialise en conséquence le tableau `t`. La convention de représentation des différentes cases est celle décrite à la question 3 pour l'affichage du labyrinthe. À titre d'exemple, le programme suivant lit le fichier et affiche son contenu

```
let fichier = open_in "laby" in
    for 1=1 to 11 do
      let s = input_line fichier in
      for c=0 to 18 do
           print_char s.[c]
      done;
      print_nevline()
    done; close_in fichier;;
```

15. Assembler les fonctions écrites précédemment pour obtenir une version jouable du programme. Les ordres de déplacement seront lus au clavier avec la fonction **read_key**; le joueur utilisera les touches 8, 2, 4 et 6 comme indiqué à la question 8.

16. Comment faire pour introduire un «droit à l'erreur», *id est* : la possibilité pour le joueur d'annuler le dernier mouvement? Ou, plus généralement, de revenir en arrière autant qu'il le veut ? On ne demande pas d'écrire un programme, mais de donner les éléments nécessaires pour ajouter cette fonctionnalité.

Quelques fonctions de bibliothèque

`read_key` de type `unit -> char` attend la frappe d'un caractère au clavier, et rend ce caractère.
`print_char` de type `char -> unit`, affiche un caractère ; celui-ci doit être encadré par des accents graves. Exemple: `print_char('X')`.
Les chaînes de caractères (type `string`) sont manipulées comme des vecteurs ; en particulier, la numérotation commence à 0. Par exemple, après `let s = "abcd"`, `s.[2]` est égal au caractère 'c' et `string_length s` est égal à 4 (longueur de la chaîne `s`) ; après `s.[3] <- 'x'`, `s` est égale à `"abcx"`.
`print_newline` de type `unit -> unit`, va à la ligne.
`make_matrix` de type `int -> int -> 'a -> 'a vect vect`, construit une matrice (*Id est* : un vecteur de vecteurs). Exemple : `make_matrix 2 3 99` construit la matrice `[|[|99;99;99|];[|99;99;99|]|]`.

Pour information, le contenu du fichier `laby` :

```
     MMMMM
     M   M
     MC  M
   MMM  CMM
   M C  C M
MMM M MM M   MMMMMM
M   M MM MMMMM  GGM
M C C           GGM
MMMMM MMM MHMM  GGM
     M     MMMMMMMMM
     MMMMMMM
```

Ce fichier a été placé dans le répertoire **c:\tmp** pour que vous puissiez y accéder en cas de besoin.