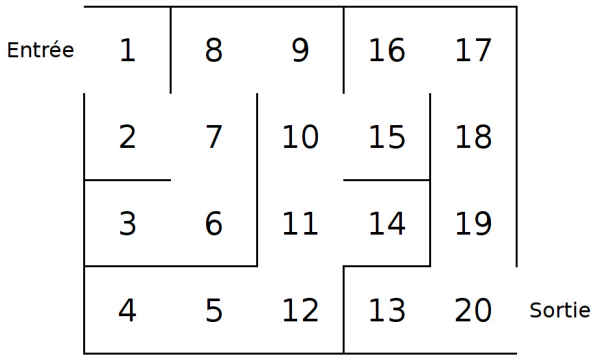
Medioni Johan TC03
Julien Maris GI01

Compte-rendu de TD-IA01

On dispose du labyrinthe suivant :



On souhaite programmer un robot pour qu'il puisse le traverser. On va utiliser la notion d'espace états dans ce but.

Partie 1 : Etude du problème

Ensemble d'états : Q = {1, 2, …, 20, e, s} où e et s sont respectivement l'entrée et la sortie du labyrinthe.

Ensemble des états initiaux : S={E}.

Ensemble des états solutions/finaux : G={S}.

Ensemble d'actions : A={(e 1) ; (1 2) ; (2 1) ; (2 7) ; (3 6) ; (4 5) ; (5 4) ; (5 12) ; (6 3) ; (6 7) ; (7 2) ; (7 6) ; (7 8) ; (8 7) ; (8 9) ; (9 8) ; (9 10) ; (10 9) ; (10 11) ; (10 15) ; (11 10) ; (11 12) ; (11 14) ; (12 5) ; (13 20) ; (14 11) ; (15 10) ; (15 16) ; (16 15) ; (16 17) ; (17 16) ; (17 18) ; (18 17) ; (18 19) ; (19 18) ; (19 20) ; (20 19) ; (20 13) ; (20 S)} où $\forall (i, j) \in \mathbb{N}^2$, i est le point de départ et j l'arrivée du déplacement.

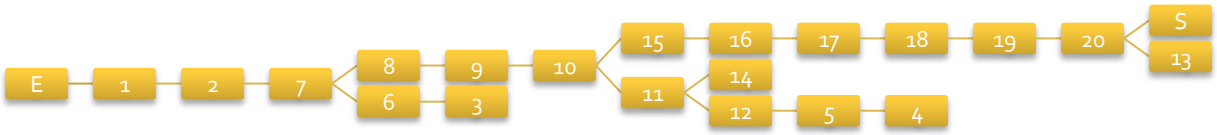
Partie 2 : Représentation du labyrinthe

On peut déduire à partir de là une représentation du labyrinthe sous forme d'une grande liste constituée de sous-listes. Ces sous-listes auraient en premier élément un état et en éléments suivants les états vers lesquels il est possible d'évoluer au moyen d'une action de A.

```
(setq lab '((e 1)(1 2)(2 1 7)(3 6)(4 5)(5 4 12)(6 3 7)(7 2 6 8)(8 7 9)(9 8
10)(10 9 11 15)(11 10 12 14)(12 5)(13 20)(14 11)(15 10 16)(16 15 17)(17 16
18)(18 17 19)(19 18 20)(20 19 13 S)))
```

Medioni Johan TC03
Julien Maris GI01

Partie 3 : Un arbre de recherche



Partie 4 : Exploration manuelle

Exploration en profondeur d'abord : on explore linéairement en allant aussi loin que possible à chaque nœud, puis en revenant à ce nœud une fois la branche choisie entièrement parcourue.

Ici, [E 1 2 7 6 3 6 7 8 9 10 11 12 5 4 5 12 11 14 11 10 15 16 17 18 19 20 13 20 S] est la plus longue exploration en profondeur possible (en revenant sur nos pas aussi peu que possible).

Exploration en largeur d'abord : on parcourt l'axe en suivant un axe avec une ligne perpendiculaire. On explore tout ce qui se trouve sur cette ligne avant de passer à l'étape suivante.

Ici, en explorant de gauche à droite et en lisant de bas en haut, on a [E 1 2 7 6 8 3 9 10 11 15 12 14 16 5 17 4 18 19 20 13 S] comme exploration en largeur d'abord.

Partie 5 : Fonction explore

On s'intéresse à l'exploration en profondeur d'abord. Définissons quelques fonctions de service :

CG-USER(6): (defun succ (etat lab) (cdr (assoc etat lab)))	Etablie la liste des successeurs d'un état quelconque.
CG-USER(59): (defun dep_1s (etat lab) (if (member (car (succ etat lab)) L) (pop back) (progn (push (car (succ etat lab)) L) (car (succ etat lab)))))	Gère la situation où il y a un successeur. Si le successeur est dans L ... ... la valeur prise est (pop back) ... ... sinon ... ... on met le successeur dans L ... ... et on le retourne.
CG-USER(65): (defun dep_2S (etat lab) (if (member (car (succ etat lab)) L) (if (member (cadr (succ etat lab)) L) (princ "On tourne en rond !") (progn (push (cadr (succ etat lab)) L) (cadr (succ etat lab))	Gère la situation où il y a 2 successeurs. Si (1) le premier successeur est dans L ... ... et si (2) le deuxième l'est aussi ... ... on considère qu'il y a un problème(1)... ... sinon (2) ... ... on met le 2° successeur dans L ... ... et on le retourne (fin de l'exécution de la

)) (progn (push etat back) (push (car (succ etat lab)) L) (car (succ etat lab)))))	fonction). … sinon (1) … … on mémorise l'état actuel, qui est un nœud, dans back … … on met le 1° successeur dans L … … et on le retourne.
CG-USER(66): (defun dep_3S (etat lab) (if (member (car (succ etat lab)) L) (if (member (cadr (succ etat lab)) L) (if (member (caddr (succ etat lab)) L) (pop back) (progn (push (caddr (succ etat lab)) L) (caddr (succ etat lab)))) (progn (push etat back) (push (cadr (succ etat lab)) L) (cadr (succ etat lab)))) (progn (push etat back) (push (car (succ etat lab)) L) (car (succ etat lab)))))	Gère la situation où il y a 3 successeurs. Cette fonction utilise le même principe que les précédentes avec quelques variantes. Cf. note 2
CG-USER(138): (defun dep_4S (etat lab) (if (member (car (succ etat lab)) L) (if (member (cadr (succ etat lab)) L) (if (member (caddr (succ etat lab)) L) (if (member (cadddr (succ etat lab)) L) (pop back) (progn (push (cadddr (succ etat lab) L)) (caddr (succ etat lab)))) (progn (push etat back) (push (caddr (succ etat lab) L)) (caddr (succ etat lab)))	Gère la situation où il y a 4 successeurs (pas présent dans ce labyrinthe).

```
 )
 (progn
  (push (etat back)
  (push (cadr (succ etat lab) L))
  (cadr (succ etat lab))
 )
)
(progn
 (push etat back)
 (push (car (succ etat lab)) L)
 (car (succ etat lab))
)
)
)
```

Note 1 : la liste L permet de stocker les positions visitées. On peut utiliser pushnew pour qu'il n'y ait pas de doublon, mais utiliser push permet aussi de voir à quel moment on est revenu en arrière puisque que les nœuds revisités sont mémoriser encore une fois.

Note 2 : la liste back permet de mémoriser les nœuds sur lesquels on **doit** revenir plus tard. On utilise le (pop back) pour retourner la valeur de ce nœud quand on arrive au bout d'une branche, c'est-à-dire quand on arrive sur un état avec un seul successeur déjà visité (= qui est dans L). On utilise également le (pop back) quand on arrive à nœud dont on a visité toutes les alternatives (= dont tous les successeurs sont dans L) (Exemple : 10 11 12 5 4 11 14 11 10 15 …).

Maintenant que nous avons ces 5 fonctions de service, nous pouvons les mettre dans la fonction explore qui prend simplement la forme suivante :

```
CG-USER(133): (defun explore (etat lab)
(case (length(succ etat lab))
 (0 (print "Arrivé !"))
 (1 (explore (dep_1S etat lab) lab))
 (2 (explore (dep_2S etat lab) lab))
 (3 (explore (dep_3S etat lab) lab))
)
)
```

N.B. : les listes L et back ne sont pas des variables internes aux fonctions. Il est impératif de les définir en dehors avant d'utiliser les fonctions. Il faut de plus les initialiser à NIL (surtout L) avant chaque réutilisation de la fonction explore. (setq L NIL) (setq back NIL)