L2 Informatique

TP Algorithmique – Labyrinthe

Ce TP consiste à écrire un programme permettant à une souris de trouver un morceau de fromage dans un labyrinthe.

Le labyrinthe contient :

1. un emplacement de départ (la souris, repérée par le symbole `'S'`),
2. un emplacement d'arrivée (le fromage, repéré par le symbole `'F'`),
3. des emplacements vides (repérés par un espace `' '`),
4. des obstacles infranchissables (représentés par `'*'`).

Voici un exemple de labyrinthe :

```
*****************************
**       *        * * * *   **
* * **       ****    ***     *
*     *   * *     * *   *   **
* *   **  **      ** *     *S*
*   * *    * * **   *        *
*    *  *** *   *     *      *
* F *    *   **        ** * **
*     * *   * *   **     *   *
*****************************
```

La souris part d'une position initiale spécifiée dans le labyrinthe et se déplace d'une case à la fois dans l'une des quatre directions cardinales (nord, sud, est, ouest). Elle ne dispose pas d'une vue globale du labyrinthe : elle ne *voit* que les cases immédiatement adjacentes à sa position.

Structures de données

1. Le labyrinthe est représenté par un tableau à deux dimensions de caractères. Le nombre de lignes et de colonnes peut être spécifié par des constantes. Chaque case (ou *position*) est identifiée par un couple de coordonnées (ligne, colonne).
   → Définir les types `Labyrinthe` et `Position` en conséquence.
2. Le chemin parcouru par la souris sera mémorisé dans une *pile de positions*.
   → Définir un type *PilePosition* permettant de représenter une pile de positions, ainsi que les primitives associées : `Initialiser`, `EstVide`, `Empiler`, `Sommet`, `Depiler` et `Vider`.

Sous-programmes à implémenter

3. Initialiser un labyrinthe vide entouré d'obstacles (`'*'`) et contenant des emplacements vides (`' '`) à l'intérieur.
4. Ajouter aléatoirement des obstacles dans les emplacements vides du labyrinthe, en fonction d'un pourcentage donné en paramètre.
5. Placer aléatoirement la souris (`'S'`) et le fromage (`'F'`) sur des emplacements libres.
6. Afficher le labyrinthe.

7. Retourner la position actuelle de la souris (ses coordonnées dans le labyrinthe).

8. À partir d'une position donnée, retourner une position libre contiguë (orthogonalement) si elle existe. Une position est considérée libre si elle contient `' '` ou `'F'`.
   → Si aucune position libre n'est trouvée, retourner une position spéciale (par exemple `(0,0)`, en supposant qu'elle ne pourra jamais être libre).

9. **Résolution du labyrinthe** : déplacer la souris jusqu'à ce qu'elle trouve le fromage ou qu'elle conclue qu'il n'existe pas de chemin possible.

   La stratégie est la suivante :

   - Tant que la souris n'a pas trouvé le fromage, elle tente de se déplacer vers une case contiguë non encore visitée.
   - Lorsqu'elle quitte une case, elle y laisse une *marque* (`'.'`) pour indiquer qu'elle y est déjà passée, et mémorise cette position au moyen d'une pile.
   - Si aucune case contiguë n'est accessible (cul-de-sac ou cases déjà visitées), elle revient en arrière en utilisant la pile des positions précédentes.

   → À la fin de la recherche, afficher l'état final du labyrinthe, en indiquant les positions initiales de `'S'` et `'F'`. Les `'.'` indiqueront les cases explorées.
   → Indiquer si la souris a trouvé ou non le fromage.

10. Si la souris a trouvé le fromage, afficher à nouveau le labyrinthe en faisant apparaître uniquement le chemin direct parcouru entre la position de départ et celle du fromage, c'est-à-dire sans les retours arrière.