Fonction et Procédure

Jeu du Labyrinthe

1. Premiers pas en graphisme

1.1 Ecrire un programme qui réalise l'affichage d'une grille sur la totalité de l'écran. Les couleurs utilisées devront être paramétrables. Le programme devra permettre la saisie des paramètres liés à la couleur et c'est une procédure qui réalisera l'affichage

1.2 Ecrire un programme qui réalise l'affichage d'une grille sur une certaine partie de l'écran. Les couleurs utilisées devront être paramétrables. La zone d'affichage ainsi que la taille des cases devront également l'être. Le programme devra permettre la saisie des paramètres liés à la couleur, à la zone d'affichage et à la taille des cases, et c'est une procédure qui réalisera l'affichage.

2. Gestion d'un curseur graphique

2.1 Ecrire un programme qui permet de déplacer un curseur graphique à l'écran. Le curseur sera représenté par un petit disque coloré et ce sont les touches « flèches » du clavier (→, ←, ↓ et ↑, ou les chiffres 6, 4, 2 et 8 du pavé numérique).

2.2 Ecrire un programme qui permet de déplacer le curseur graphique à l'écran à l'intérieur d'une grille (obtenue à l'aide de la procédure programmée dans la question 1.2). Le curseur sera positionné à l'intérieur des cases et se déplacera maintenant de case en case.

On se propose de programmer un jeu (interactif) qui doit permettre tout d'abord de créer un labyrinthe graphiquement, puis à un utilisateur (le joueur) de se déplacer dans ce labyrinthe avec comme objectif d'en trouver la sortie. Le programme se décompose en deux parties distinctes que nous détaillons :

3. Jeu interactif du labyrinthe

On se propose de programmer un jeu interactif graphique de labyrinthe. Ce jeu se déroulera en deux étapes. Tout d'abord un étape de création d'un labyrinthe, puis dans un second temps, une phase de jeu effectif. Dans une première étape, un utilisateur a face à lui une grille (nXn) correspondant aux limites du labyrinthe (figure 1). Initialement, un curseur est positionné sur la case de départ. La définition du labyrinthe va se faire en déplaçant le curseur de case en case à l'aide des 4 flèches du clavier (→, ←, ↓ et ↑, ou des chiffres pris sur le pavé numérique), en indiquant ainsi quels seront les points de passage entre les cases (figure 2). Une fois le labyrinthe défini (figure 3), celui-ci est mémorisé, puis effacé de l'écran. La phase de jeu peut alors débuter.

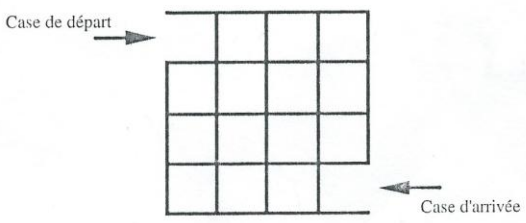

Figure 1 : vue de la grille associée au labyrinthe pendant sa conception.

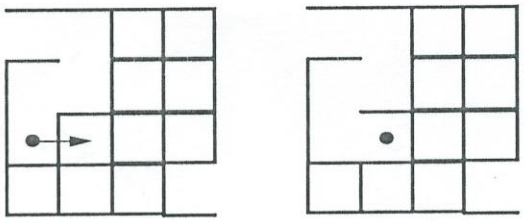

Figure 2 : définition d'un point de passage entre deux cases du labyrinthe : le curseur est matérialisé par un disque.

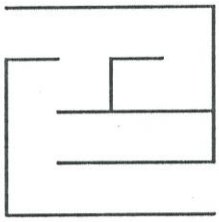

Figure 3 : vue d'un labyrinthe après sa conception.

Lors de la seconde phase, seules les limites du labyrinthe sont visualisées. Le joueur (son curseur) est alors positionné sur la case de départ. Le joueur va essayer de se déplacer à l'aide des 4 flèches du clavier afin d'arriver à la sortie. Il va donc essayer de passer d'une case à l'autre, mais ceci, sans disposer de la vue du labyrinthe. Si une tentative de passage d'une case à une autre est autorisée (pour le cas où il n'y a pas de mur de séparation), le curseur se positionne alors sur la case d'arrivée (figure 4), sinon, le mur interdisant ce passage est matérialisé par un trait (figure 5). Par ailleurs, un compteur doit dénombrer les transitions tentées par le joueur, ce total fournissant son score au jeu. On peut compliquer le jeu en ne matérialisant pas les différents murs rencontrés pendant la recherche ; on dispose ainsi de deux niveaux de jeu qui peuvent être proposés au joueur en début de partie.

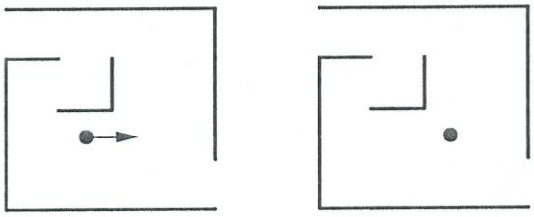

Figure 4 : déplacement: seuls certains murs du labyrinthe sont visualisés. Ici la transition essayée par le joueur (flèche →) est permise ; le curseur se positionne donc sur la case demandée.

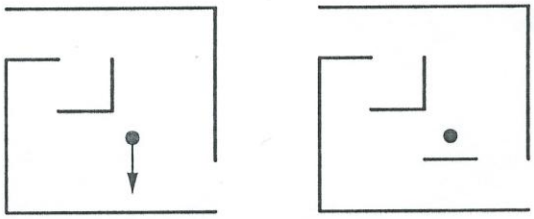

Figure 5 : déplacement interdit : la transition demandée par le joueur (flèche ↓) est impossible ; le curseur reste donc sur la même case, mais le mur interdisant cette transition est matérialisé.

Pour coder le labyrinthe, nous utiliserons le type C suivant :

```
typedef struct {
                  int mv[n][n]; /* Murs_Verticaux ; n prédéfini */
                  int mh[n][n]; /* Murs_Horizontaux; n prédéfini */
} labyrinthe ;
```

Etant donnée une variable `lab` de type `labyrinthe`, on aura `lab.mv[i][j]` = 1 si et seulement s'il existe un mur entre la case de la rangée i colonne j et la case de la rangée i colonne j+1, en numérotant les rangées et les colonnes de 0 à n-1 ; sinon la valeur de `lab.mv[i][j]` vaudra 0. Ainsi, la case d'entrée située en haut à gauche, de rangée 0 et de colonne 0 sera séparée de la case située sur la rangée 0 et de colonne 1 si `lab.mv[0][0]` a pour valeur 1. Il faut noter que s'il y a n rangées de murs verticaux, il n'y a par contre que n-1 colonnes de murs verticaux. De même, s'il n'y a que n-1 rangées murs horizontaux, il y a par contre n colonnes de murs horizontaux.

Un certain nombre de procédures sont à définir, parmi lesquelles :

```
void affichegrillepleine ()
/* specif : affiche la grille complète pour définir le labyrinthe */
```

```
void initlabyr(labyrinthe lab)
/* specif : initialisation avec tous les murs du labyrinthe lab */
```

Cette procédure va ainsi affecter à 1 tous les murs potentiels du labyrinthe représenté dans la variable de type `labyrinthe` passée en paramètre.

```
void affichelabyr(labyrinthe lab)
/* specif : affiche la grille du labyrinthe lab */
```

Cette procédure va ainsi afficher, outre les limites extérieures du labyrinthe, tous les murs présents. C'est-à-dire qu'un segment de droite associé à un mur vertical sera affiché si bien sûr la valeur associée à ce mur dans `lab.mv` vaut 1. Sinon, elle n'affichera pas ce mur. Ainsi, si par exemple `lab.mv[0][0]` a pour valeur 1, alors le mur correspondant à la séparation entre la case de rangée 0 et de colonne 0 et la case située sur la rangée 0 et la colonne 1 sera affiché.

```
void saisielabyr(labyrinthe lab)
/* specif : construit interactivement le labyrinthe lab */
```

```
void jeulabyr(labyrinthe lab)
/* specif : met en oeuvre le jeu en utilisant le labyrinthe lab */
```