Labyrinthus
Projet de langage C — IFI1

Élise Vareilles
Frédérick Benaben

2007-2008

Tous les fichiers évoqués dans ce document sont présents dans le répertoire `/home/vareille/Cours/PUBLIC/mini-projet/2007-2008/`.

Introduction

Le terme **labyrinthe** vient du grec $\lambda\alpha\beta\nu\varrho\iota\nu\theta o\varsigma$. Il désigne dans la mythologie grecque une série complexe de galeries construites par Dédale pour enfermer le Minotaure.

Par extension, un labyrinthe est un tracé sinueux, muni ou non d'embranchements, d'impasses et de fausses pistes, destiné à perdre ou à ralentir celui qui cherche à s'y déplacer. Ce motif apparu à la préhistoire se retrouve dans de très nombreuses civilisations sous des formes diverses [1].

Objectifs de ce mini-projet

Les objectifs de ce mini-projet sont :
- de lire la description du labyrinthe fournie dans un fichier LAB1.TXT (par exemple) (cf. sectionDB),
- de décrypter celle-ci et de la stocker dans la structure de données la mieux adaptée,
- d'afficher le labyrinthe en utilisant la bibliothèque graphique *PLplot*[2] et ses fonctions associées (cf. section 3.3),
- enfin de guider Thésée pour l'aider à sortir du labyrinthe.

1 Description du labyrinthe

Un labyrinthe est décrit par un fichier .txt contenant :
- sur sa première ligne la taille du labyrinthe et le nombre d'éléments de type mur coulissant de bas en haut : nombre de lignes, nombre de colonnes, nombre de murs coulissants,
- sur les lignes suivantes les descriptions ligne par ligne du labyrinthe.

Chaque ligne du labyrinthe est décrite par une succession de couples (nombre, lettre) indiquant :
- pour le nombre, le nombre d'éléments se succédant de même type,
- pour la lettre, le type d'élément du labyrinthe.

Il existe 6 types d'élèvent composant un labyrinthe :
- la lettre **E** indique la position de l'entrée du labyrinthe ainsi que la position initiale de Thésée,
- la lettre **S** indique la position de la sortie du labyrinthe,
- la lettre **P** indique un élément de type passage sur lequel Thésée va pouvoir se déplacer,
- la lettre **M** indique un élément de type mur au travers duquel Thésée ne peut pas passer,

[1] tiré de Wikipédia
[2] `http://plplot.sourceforge.net/`

- la lettre **C** indique un mur coulissant de bas en haut sur lequel Thésée peut se déplacer lorsque le mur est en position haute,
- la lettre **T** indique un élément de type téléporteur qui va téléporter Thésée à une position tirée aléatoirement dans l'ensemble des passages.

Seuls les éléments de type entrée, sortie, passage, mur et téléporteur, codés respectivement par les lettres **E**, **S**, **P**, **M** et **T**, sont positionnés dans la description d'un labyrinthe.

Les éléments de type mur coulissant de bas en haut, codés par la lettre **C**, sont positionnés de manière aléatoire dans l'ensemble des passages, leur nombre étant donné dans la première ligne du fichier de description et leur état (levé ou baissé) étant tiré aléatoirement. Les murs coulissant de bas en haut se lèvent ou se baissent à chaque tour de jeu selon leur état au tour précédent. Un mur coulissant en position haute au tour courant se baissera au tour suivant alors qu'un mur coulissant en position basse au tour courant se lèvera au tour suivant.

Par exemple, voilà les 3 premières lignes de description du labyrinthe codé dans le fichier LAB1.TXT :

L1 20 20 5

L2 20M

L3 1E1P1T7P2M7P1M

L4 .....

La première ligne indique la taille du labyrinthe : 20 lignes par 20 colonnes, ainsi que le nombre d'éléments

La deuxième ligne est une ligne composée uniquement de mur :

M M M M M M M M M M M M M M M M M M M M

La troisième ligne est une ligne composée d'une entrée suivie d'un passage, suivi d'un téléporteur, suivi de 7 passages, suivis de 3 passages, suivis de 2 murs, suivis de 7 passages, suivis d'un mur :

E P T P P P P P P P M M P P P P P P P M

Thésée est par défaut positionné sur l'entrée du labyrinthe.

Les deux premières lignes du labyrinthe sont donc les suivantes :

M M M M M M M M M M M M M M M M M M M M
E P T P P P P P P P M M P P P P P P P M

2 Description du comportement de Thésée

Thésée peut se déplacer suivant quatre directions :

- Nord,
- Est,
- Sud,
- Ouest.

Pour aider Thésée à sortir du labyrinthe, vous devez le guider à travers celui-ci en lui indiquant la direction dans laquelle ce dernier doit avancer. Cette aide s'effectue de manière interactive.

Thésée ne comprend qu'une seule instruction à la fois. Pour le guider, vous devrez donc lui indiquer son itinéraire case par case.

Par exemple, posons que les lettres **N**, **E**, **S** et **O** correspondant respectivement aux instructions se déplacer au Nord, à l'Est, au Sud et à l'Ouest d'une case. Dans le cas précédent du labyrinthe LAB1.TXT, pour indiquer à Thésée de se déplacer de deux cases vers l'est, vous devrez taper 2 fois l'instruction : E ↩ dans votre console xterm.

3 Approche informatique

3.1 Decryptage du labyrinthe, procédure strcpy

Le décryptage d'une ligne décrivant le labyrinthe peut nécessiter l'emploi d'une procédure, nommée **strcpy**, relative aux chaînes de caractères. Cette procédure est accessible via l'inclusion de la bibliothèques standard *string.h*.

La procédure **strcpy** permet de recopier la valeur d'une chaîne de caractères dans un tableau de caractères. Elle prend deux paramètres :

- le premier paramètre est un tableau de caractères, pointeur sur caractère, dans lequel la chaîne va être recopiée,
- le second paramètre est la chaîne de caractères elle-même.

```
void strcpy(char * copie, char * a_copier);
```

Remarques : Attention, les éléments lus dans le fichier de description sont de type caractère. Chaque caractère est codé par un code ASCII, selon la norme ANSI. Le code ASCII du caractère '2' est 50 en hexadécimal. La conversion des caractères représentant des chiffes dans la description d'une ligne d'un labyrinthe n'est donc pas immédiate.

3.2 Génération de nombre aléatoire

La génération de nombre aléatoire par un ordinateur est un problème beaucoup plus complexe qu'il n'y parait. De nombreux articles et livres ont déjà été publiés sur le sujet. Nous ne rentrerons pas dans les détails techniques dans ce document. Quelques points à retenir :

- un générateur réellement aléatoire n'est pas réalisable avec un ordinateur seul. Pour que le générateur soit réellement aléatoire, il lui faut impérativement un capteur externe lui donnant une information aléatoire. Par exemple, un capteur lui indiquant la hauteur actuelle d'un jet d'eau.
- en l'absence de cette information externe, le générateur ne peut être que pseudo-aléatoire. Dans ce cas, on utilise des suites numériques pseudo-aléatoires dont on fournit la valeur initiale. Pour une valeur initiale donnée, la suite sera toujours la même. Les programmes voulant simuler une véritable suite aléatoire utilise par exemple comme graine l'heure actuelle.

Nous vous proposons dans les fichiers `alea.c` et `alea.h` quatre fonctions permettant d'utiliser un générateur pseudo-aléatoire.

3.2.1 La fonction `init_alea_graine`

Cette fonction permet d'initialiser la suite pseudo-aléatoire en lui donnant la graine initiale. Elle prend comme seul paramètre la valeur de la graine (un `long`). Pour une même graine, la suite fournie sera toujours la même.

```
void init_alea_graine(long graine);
```

3.2.2 La fonction `init_alea`

La fonction `init_alea` permet d'initialiser le générateur aléatoire avec une graine presque aléatoire. Dans notre bibliothèque, c'est l'heure (à la micro-seconde) qui est utilisée. Pour en savoir plus, vous pouvez regarder les sources. Une fois initialisé par cette fonction, le générateur semblera donner une suite de valeur vraiment aléatoire.

```
void init_alea();
```

3.2.3 La fonction `alea_double`

La fonction `alea_double` ne prend aucun paramètre. Elle retourne comme résultat le nombre aléatoire suivant de la suite choisie. Chaque appel retourne une valeur différente. La valeur retournée est un double contenu dans $[0, 1[$.

```
double alea_double();
```

3.2.4 La fonction `alea_int`

La fonction `alea_int` ne prend qu'un seul paramètre : la valeur maximale. Elle retourne comme résultat le nombre aléatoire suivant de la suite choisie. Chaque appel retourne une valeur différente. La valeur retournée est un entier contenu dans $[0, maximum[$.

```
int alea_int(int maximum);
```

3.3 L'affichage graphique

Pour la visualisation de la simulation, nous avons choisi de travailler avec la bibliothèque *PLplot*[3]. Dans le cadre du projet, l'utilisation des fonctions de cette bibliothèque restera restreinte. Cependant il nous a semblé nécessaire de vous aider en vous fournissant trois fonctions à utiliser pour la simulation.

Ces trois fonctions sont dans les fichiers `visu.c` et `visu.h`.

3.3.1 La fonction `init_visu`

```
void init_visu(int xmax, int ymax);
```

La fonction `init_visu` doit être appelée *une* fois avant tout autre routine de la bibliothèque *PLplot*. Elle permet d'initialiser la bibliothèque graphique. Les deux paramètres sont la largeur (`xmax`) et la hauteur (`ymax`) de la grille à dessiner. En modifiant cette fonction, il est possible de choisir d'autres couleurs.

3.3.2 La fonction `trace_case`

```
void trace_case(int couleur, int x, int y);
```

Cette seconde fonction, `trace_case`, permet de dessiner un carré de couleur correspondant à une case de la grille. Le premier paramètre est le numéro d'une couleur (0 - fond (blanc) ; 1 - rouge ; 2 - vert ; 3 - noir ; 4 - bleu). Les deux autres paramètres sont les coordonnées de la case à dessiner (compris respectivement entre $[0, xmax[$ et $[0, ymax[$). On peut, en s'inspirant du code de cette fonction, imaginer d'autres possibilités pour le choix de la forme.

3.3.3 La fonction `trace_bord`

```
void trace_bord(int xmax, int ymax);
```

La fonction `trace_bord` permet de tracer un contour à la grille afin de mieux voir les limites de la zone du couvain. Les deux paramètres sont la taille du contour.

[3] `http://plplot.sourceforge.net/`

3.3.4 Autres fonctions utiles de *PLplot*

Vous aurez aussi besoin d'appeler de temps à autre la fonction `plbop()`. Cette fonction demande à votre serveur d'affichage (votre terminal X-Window par exemple) d'oublier tous les ordres graphiques précédemment reçus. Sinon vous saturerez la mémoire de votre serveur d'affichage. Accessoirement, elle efface la fenêtre d'affichage. Après un appel à cette fonction, il vous faudra redessiner entièrement l'état du couvain (fourmis, œufs et bordure). Nous vous conseillons d'appeler cette fonction environ tous les 100000 tracés. Cette fonction n'a pas d'argument.

Une dernière fonction de la bibliothèque *PLplot* vous sera utile : la fonction `plflush()`. L'appel de cette fonction vous garantit que tous les ordres envoyés au serveur d'affichage sont effectivement réalisés et ne sont pas restés en attente de traitement. À appeler avant la saisie d'informations via la fenêtre de commande.

4 Compilation

Pour compiler votre programme et faire l'édition de liens avec les fonctions de la bibliothèque *PLplot*, il faut indiquer au compilateur l'endroit où est rangée cette bibliothèque dans l'arborescence générale.

Comme les fonctions de *PLplot* utilisent elles-mêmes des fonctions provenant d'autres bibliothèques (gd, png, X11 et mathématique), il faut aussi le lui signaler.

L'instruction de compilation peut prendre alors la forme suivante :

```
gcc -Wall -o lab -I/usr/local/gnu/include/ lab.c visu.c alea.c \
 -L/usr/local/gnu/lib -lplplotd -lgd -lpng -lX -lm -liconv
```

Attention : la bibliothèque *PLplot* n'est pas installée sur les stations Linbox. Vous devrez vous connecter (via la commande `xon`) sur l'une des stations Sun pour compiler et tester votre programme.

5 Quelques conseils pour débuter

Tout d'abord, vous pouvez essayer le programme `lab` qui sera fourni en début de deuxième séance de mini-projet encadré. Cela vous permettra de mieux comprendre le fonctionnement général du programme à écrire.

Puis, dans un premier temps, nous vous conseillons de réaliser un premier programme permettant de lire la description du labyrinthe dans le fichier de description.

Après cela, ajoutez une fonction de décryptage des lignes du labyrinthe, puis la fonction d'affichage de celui-ci.

Dans un premier temps, ne prenez pas en compte la téléportation, ni le comportement daes murs coulissants. Réalisez la fonction de déplacement de Thésée.

Enfin, terminez votre programme par l'ajout de la téléportation et du comportement des murs coulissants.

Conseils : dans un premier temps, utilisez une structure statique de données, sachant que la taille d'un labyrinthe est limitée à 100x100. Dans un second temps, lorsque votre version statique fonctionnera, vous pourrez passer à une version dynamique de vos structures de données (allocation dynamique de la mémoire en fonction des paramètres lus).

6 Idées de variantes

Pour ceux qui veulent aller plus loin, voici différentes idées de variantes possibles :

– les instructions de directions données à Thésée pourraient être un peu plus évoluées et similaires à la description des labyrinthe. Par exemple, l'instruction 2O4S3E5N indique à Thésée de se déplacer de 2 cases vers l'Ouest, puis 4 cases vers le Sud, puis 3 cases vers l'Est puis 5 cases vers le Nord ;

- Thésée pourrait aussi avoir une orientation et regarder dans la direction de son déplacement. Il serait alors orienté vers l'Est dés le départ. Dans ce cas, les instructions à suivre pourraient être : GAD5A pour lui indiquer de tourner vers la gauche, puis d'avancer d'une case, puis de tourner vers la droite et d'avancer de 5 cases ;
- il pourrait être tout à fait intéressant de vérifier avant de commencer à guider Thésée qu'il existe bien au moins un chemin entre l'entrée et la sortie du labyrinthe (prouver la connexité du labyrinthe, cf. cours de recherche opérationnelle d'IFI2) sans tenir compte des murs coulissants de bas en haut, ni des téléporteurs ;
- il pourrait être aussi intéressant de trouver le plus court chemin dans un labyrinthe connexe sans tenir compte des murs coulissants de bas en haut, ni des téléporteurs ;
- vous pourriez aussi ajouter de nouveaux pièges afin de rendre plus difficile la progression de Thésée (trous noirs qui ramènent Thésée sur l'entrée ou bien le Minautore qui cherche à tuer Thésée, par exemple) ;

7 Travail à rendre

Ce travail constituera la seconde partie de l'évaluation du cours d'algorithmique et programmation. Pour cela, il vous faudra rendre par binôme un dossier comprenant au moins les pièces suivantes :

- une analyse du problème à résoudre et toutes les explications nécessaires à la compréhension des solutions que vous aurez choisies dans votre programme ;
- les algorithmes issus de cette analyse fournis sous forme de schéma bulle précisant leurs entrées et leurs sorties, respectant les 4 flux d'entrée-sortie vus en algorithmique ;
- les liens entre vos différents algorithmes illustrés par une graphique mettant en relation les bulles algorithmiques ;
- la description précise des algorithmes que vous jugez pertinents pour résoudre le problème.

Vous pouvez fournir d'autres informations si vous l'estimez nécessaire (en particulier si vous avez essayé différentes variantes).

Votre travail doit être impérativement rendu le **vendredi 27 juin 2008 midi**. Votre rapport devra être remis à M. Massat et vos fichiers sources devront être déposés dans le répertoire `/home/vareille/Cours/PUBLIC/mini-projet/2007-2008/Depot/.` (un seul dépot autorisé par binôme). Vos fichiers sources devront porter un nom correspondant aux noms des étudiants du binome rangés par ordre alphabétique. Par exemple : Benaben_Vareilles.c.