

Bases de la Programmation

TP 6 – Démineur

V. Padovani, PPS - IRIF

Le jeu du démineur se joue à un unique joueur sur un plateau rectangulaire divisée en cases. Chaque case est soit vide, soit contenant une mine. Au début du jeu, le contenu de chaque case est dissimulé au joueur. À chaque tour de jeu :

- Le joueur choisit une carte non encore découverte.
- Le contenu de cette case est révélé au joueur. S'il s'agit d'une mine, le joueur perd immédiatement. Sinon, on inscrit dans la case le nombre de mines voisines immédiates de cette case (horizontalement, verticalement et en diagonale).

Le jeu se termine lorsque toutes les cases non minées ont été découvertes par le joueur, avec dans ce cas victoire de celui-ci. Le but de ce TP est d'implémenter le jeu du démineur. Au début de votre code (en un seul fichier) vous définirez quatre constantes :

```
#define HAUTEUR 10
#define LARGEUR 10
#define MINE -1
#define VIDE -2
```

Le plateau de jeu sera représenté par un tableau à deux dimensions de taille `HAUTEUR` × `LARGEUR`. Tout au long du jeu, les contenus des cases du plateau seront encodés par des valeurs entières, avec la convention suivante :

- Une case non encore découverte et minée contiendra la valeur `MINE`.
- Une case non encore découverte non minée contiendra la valeur `VIDE`.
- Toute case non minée découverte contiendra le nombre de mines voisines de cette case (*ie.* un entier positif ou nul, contrairement aux deux constantes précédentes).

Exercice 1 :

Écrire une fonction

```
void afficher_plateau (int plateau[HAUTEUR] [LARGEUR])
```

affichant le contenu courant du plateau. Nous la modifierons un peu plus tard, mais pour le moment, cette fonction affichera la solution, en révélant le contenu des cases non encore découvertes (*eg.* un point pour du vide, une étoile pour une mine) : vous pourrez donc vous en servir pour tester les fonctions des questions qui suivent.

Exercice 2 :

Écrire une fonction

```
void initialiser (int plateau[HAUTEUR] [LARGEUR])
```

initialisant chaque case d'un plateau de jeu à la valeur `VIDE`.

Exercice 3 :

Écrire une fonction

```
void placer_mines (int plateau[HAUTEUR][LARGEUR], int n)
```

plaçant `n` mines aléatoirement sur un plateau vide. Idéalement, `n` devrait pouvoir être compris entre 0 et `HAUTEUR × LARGEUR` inclus, mais implémenter un tel tirage n'est pas immédiat (il faut tirer la position d'une nouvelle mine parmi les positions non encore minées). À défaut, vous pouvez vous contenter de placer des mines en `n` positions tirées au hasard, tant pis si une même position est tirée deux fois.

Exercice 4 :

Noter que le voisinage d'une case est tronqué sur les bords du plateau. Écrire une fonction

```
int dans_plateau (int i, int j)
```

renvoyant 1 si le numéro de ligne `i` et le numéro de colonne `j` sont dans les limites du plateau, et 0 sinon. À partir de cette fonction, écrire

```
int mines_voisines (int plateau[HAUTEUR][LARGEUR], int i, int j)
```

renvoyant le nombre de mines voisines de la case du plateau de numéro de ligne `i` et de numéro de colonne `j`.

Exercice 5 :

Écrire une fonction

```
int jouer_coup (int plateau[HAUTEUR][LARGEUR], int i, int j)
```

découvrant la case de numéro de ligne `i` et de numéro de colonne `j`. La fonction doit renvoyer -1 si la case est déjà découverte ou si les coordonnées sont incorrectes, 0 en cas de défaite immédiate et sinon, renvoyer 1 après avoir mis à jour le plateau.

Exercice 6 :

Modifier la fonction `afficher_plateau` du premier exercice en lui ajoutant un paramètre de plus. Suivant la valeur de ce paramètre, la fonction affichera soit, comme précédemment, le plateau entièrement découvert (la solution), soit le plateau tel qu'il est vu par le joueur (un point pour une case non encore découverte et sinon, l'information numérique inscrite dans cette case).

Exercice 7 : S'il vous reste du temps.

Écrire une fonction `main` combinant les fonctions précédentes pour en faire un jeu de démineur complet : choix du nombre de mines par le joueur, et boucle principale du jeu.

Exercice 8 : Plus difficile.

Lorsqu'une case n'a aucune mine dans son voisinage, on peut sans risque découvrir toutes les cases voisines non encore découvertes. Si certaines d'entre elles sont sans mines voisines, on peut sans risque découvrir toutes leurs voisines non encore découvertes, etc.

Ce traitement, à partir de la position de la case sans mines voisines initiale, peut être implémenté à l'aide d'une fonction récursive (et, bien sûr, de certaines des fonctions précédentes). Il pourra être effectué en complément de chaque coup découvrant une telle case.