LIFAPI – TP 12 - 13 : Démineur

Objectifs : Approfondir les notions vues dans le TD précédent (structures)
Réviser toutes les notions abordées jusqu'à présent dans les travaux dirigés et travaux pratiques
Au travers d'un exemple de conception, analyser un problème et concevoir sa solution pas à pas

Le démineur

Il s'agit lors des TD/TP de réaliser un jeu de démineur. Dans le TD, les déclarations de types et les sous-programmes demandés devront être écrits en langage C/C++.

Principe du jeu :

L'objectif est de trouver les mines qui sont cachées aléatoirement par l'ordinateur dans les cases du tableau.

Si la case choisie contient une mine, la partie est perdue. Si la case choisie ne contient pas de mine alors apparaîtra un chiffre indiquant le nombre de mines qui se trouvent dans les 8 cases qui touchent directement la case sélectionnée.

Par exemple si le numéro découvert est un 2, cela indique qu'il y a 2 mines cachées parmi les 8 cases qui touchent directement celle choisie.

Attention : seul un affichage en **mode texte** sera demandé ici.

Construction du jeu :

1. Structures de données :
   a. Définir 3 constantes *TGX*, *TGY* et *Mines* permettant de fixer les paramètres généraux du jeu. Dans l'exemple précédent, on aura comme valeurs respectives (9, 9 et 10).

```
#define NB_MINES 10 // définition du nombre de mines à trouver
#define TAILLE_X 9 // définition de la taille de la grille
#define TAILLE_Y 9 // nombre de lignes * nombre de colonnes
```

   b. Définir une structure *case_grille* comportant deux informations :

- un entier représentant le contenu de cette case (-1 si mine, 0..8 pour le nombre de mines dans les cases adjacentes)
- un booléen permettant de savoir si cette case a déjà été choisie par l'utilisateur. Cette valeur indiquera si la case doit être dévoilée à l'utilisateur lors de l'affichage.

```
struct case_grille
{
  int valeur;
  bool decouverte;
};
```

c. Définir la grille de jeu comme étant un tableau 2D de *case_grille*. Pour des besoins ultérieurs, nous allons volontairement surdimensionner la grille de jeu en ajoutant une ligne supplémentaire en haut et en bas et une colonne supplémentaire à droite et à gauche.

```
struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2]
```

2. Initialisation de la grille de jeu : écrire un sous-programme permettant d'initialiser la grille de jeu. Pour chacune des cases de la grille du jeu, la valeur numérique sera initialisée à 0 et le booléen découvert à la valeur "false".

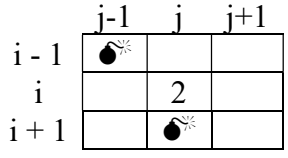
```
void init_grille (struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2])
{
 int i,j;
 for (i=0;i<TAILLE_X+2;i++)
 {
    for (j=0;j<TAILLE_Y+2;j++)
    {
       grille_jeu[i][j].valeur=0;
       grille_jeu[i][j].decouverte=false;
    }
 }
}
```

Pour rappel, un tableau est toujours en donnée/résultat.

3. **Positionnement aléatoire des bombes sur la grille de jeu** : écrire un sous-programme permettant de positionner aléatoirement *Mines* bombes sur la grille de jeu. Attention, avant de placer une mine, on vérifiera que la case est "libre".

```
void pose_bombes (struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2])
{
    int num_bombe;
    int x,y;
    for (num_bombe=0;num_bombe<NB_MINES;num_bombe++)
    {
       do
       {
         x=rand()%TAILLE_X+1;  // le +1 pour se décaler de la 1ère colonne
         y=rand()%TAILLE_Y+1;
       }
       while (grille_jeu[x][y].valeur!=0);
       grille_jeu[x][y].valeur=(-1);
    }
}
```

4. **Remplissage complet de la grille de jeu.** Dans ce sous-programme, nous allons pour chacune des cases (hors mine) compter le nombre de mines dans toutes les cases adjacentes. Pour éviter une gestion trop complexe des cases du pourtour, nous avons, dans la déclaration, pris soin de rajouter des cellules supplémentaires. Écrire le sous-programme permettant de remplir la grille avec ces valeurs.
   Exemple pour le calcul de la case i, j :




```
void compte_bombes(struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2])
{
 int i,j,k,l;
  for (i=1;i<TAILLE_X+1;i++) // commence à 1 pour ne remplir que les cases ”utiles”
```

```
{
   for (j=1;j<TAILLE_Y+1;j++)
   {
      if (grille_jeu[i][j].valeur!=-1) // si on a une bombe (-1), on ne change rien
      {
         for (k=i-1;k<=i+1;k++)    // double boucle pour parcourir les cases adjacentes
         {
            for (l=j-1;l<=j+1;l++)
            {
               if (grille_jeu[k][l].valeur==-1)
                  grille_jeu[i][j].valeur++;
            }
         }
      }
   }
}
}
```

5. **Affichage de la grille de jeu.** La grille de jeu est stockée en interne sous forme d'un tableau de structures *case_grille* contenant une valeur numérique (nombre de mines ou -1) et un booléen indiquant si la case doit être affichée ou non. Pour rendre la grille plus lisible pour l'utilisateur, les cases non dévoilées seront affichées avec le symbole '-', les cases contenant une mine (-1) avec le caractère 'M' et les autres cases avec la valeur numérique correspondant au nombre de mines dans les cases adjacentes. Écrire un sous-programme permettant d'afficher le contenu de la grille de jeu.

Exemple pour une grille de taille 5*5 avec 4 mines.

0/f	0/f	0/f	0/f	0/f	0/f	0/f
0/f	1/f	1/f	1/t	1/f	-1/t	0/f
0/f	1/f	-1/f	1/f	1/f	1/f	0/f
0/f	2/f	3/t	2/t	1/t	0/t	0/f
0/f	-1/f	2/f	-1/f	1/t	0/t	0/f
0/f	1/f	2/t	1/t	1/f	0/t	0/f
0/f	0/f	0/f	0/f	0/f	0/f	0/f

Grille stockée

-	-	1	-	M
-	-	-	-	-
-	3	2	1	0
-	-	-	1	0
-	2	1	-	0

Grille affichée

```
void affiche_grille (struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2])
{
 int i,j;
 cout<<"  ";

// affichage du n° de colonne :
 for (i=1;i<TAILLE_Y+1;i++)
 {
    cout<<i<<" ";
 }
 cout<<endl;

// affichage de la grille :
 for (i=1;i<TAILLE_X+1;i++)
 {
    cout<<i<<" ";  /// affichage du n° de ligne
    for (j=1;j<TAILLE_Y+1;j++)
    {
       if (grille_jeu[i][j].decouverte)
          if(grille_jeu[i][j].valeur ==-1)
            {
             putchar(207); // permet d'afficher un caractere qui ressemble à une bombe...
on peut toujours mettre un M a la place !!!
```

```
            cout <<" ";
            }
          else cout<<grille_jeu[i][j].valeur<<" ";
        else cout<<"- ";
    }
    cout<<endl;
 }
}

// sous-programme permettant d'afficher tout le contenu de la grille sous forme
d'entiers (pour debug)

void affiche_grille_entiere (struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2])
{
 int i,j;
 for (i=1;i<TAILLE_X+1;i++)
 {
    for (j=1;j<TAILLE_Y+1;j++)
    {
          cout<<grille_jeu[i][j].valeur<<" ";
    }
    cout<<endl;
 }
}
```

6. **Choix d'une case :** écrire un sous-programme qui demande à l'utilisateur les coordonnées de la case à sonder.

```
int decouvrir_une_case (struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2], int
&x, int &y)
{
    int code_retour; // pour le jeu complet, si code_retour= 0 on a touché une bombe
donc la partie est terminée

    do
    {
     cout<<"données les coordonnées de la case à sonder"<<endl;
     cin>>x>>y;
    } while (grille_jeu[x][y].decouverte==true);

    if (grille_jeu[x][y].valeur==-1)
       code_retour = 0;
       else code_retour=1;
    return code_retour;
}
```

7. **Jeu :** écrire le sous-programme permettant de recommencer le choix d'une case jusqu'à la fin de la partie.

```
int jouer (struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2], int &cases_restantes)
{
    bool bombe=false;
    int x,y,retour;
     // on joue tant que l'on a encore des cases non découvertes et que l'on n'est pas
   tombé sur une bombe
     while ((cases_restantes!=0)&&(bombe==false))
    {
         retour=decouvrir_une_case(grille_jeu,x,y);
         if (retour==0)
         {
             bombe=true;
             grille_jeu[x][y].decouverte=true;
         }
         else if (grille_jeu[x][y].decouverte==false)
```

```
                {
                  grille_jeu[x][y].decouverte=true;
                  cases_restantes--;
                }
                else
                   cout<<"case déja jouée"<<endl;

    affiche_grille(grille_jeu);
    cout<<"Nombre de cases à trouver : "<<cases_restantes<<endl;
    }
    if (cases_restantes ==0)
       return 0;  // partie finie et gagnée
    else return 1; // partie finie et perdue
}
```

8. **Fin de partie :** la partie s'arrête quand toutes les cases ont été découvertes sauf celles contenant des mines (partie gagnée) ou bien lorsqu'une mine est touchée (partie perdue). Écrire le programme principal permettant de jouer au démineur.

```
int main (void)
{
   struct case_grille jeu[TAILLE_X+2][TAILLE_Y+2];
   int nb_cases=(TAILLE_X*TAILLE_Y)- NB_MINES;
   int fin_partie;
   cout<<"Nombre de cases à trouver : "<<nb_cases<<endl;
   srand(time(NULL));
   init_grille_vide(jeu);
   affiche_grille(jeu);
   pose_bombes(jeu);
   compte_bombes_voisinage(jeu);

   fin_partie=jouer(jeu,nb_cases);
   if (fin_partie==0)
      cout<<"BRAVO vous avez gagné"<<endl;
   else cout<<"PERDU : vous avez touche une mine !!!"<<endl;

   system("PAUSE");
   return 0;
}
```