Séance 19 : Jeudi 25/04/2019

Le démineur

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.

Construction du jeu :

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

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

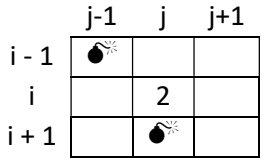
      - un entier représentant le contenu de cette case (-1 si mine, 0..8 pour le nombre de mines dans les cases adjacentes)
      - un booléen permettant de savoir si cette case a déjà été choisie par l'utilisateur. Cette valeur indiquera si la case doit être dévoilée à l'utilisateur lors de l'affichage.

   3. 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.

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".
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".

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 :



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

Jeu

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

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

3. 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.

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

const int TGX = 3 ; // petite grille pour tester le jeu jusqu'au bout
```

```
const int TGY = 3 ;
const int MINES = 4 ;

struct case_grille{
    int val;
    bool visible;
};
void init_demineur (struct case_grille jeu[TGX+2][TGY+2]){
    int i,j;
    for (i=0;i<=TGX+1;i++)
    {
        for (j=0;j<=TGY+1;j++)
        {
            jeu[i][j].val = 0;
            jeu[i][j].visible = false ;
        }
    }
}
void affiche_tout_demineur (struct case_grille jeu[TGX+2][TGY+2]){
    int i,j;
    for (i=0;i<=TGX+1;i++)
    {
        for (j=0;j<=TGY+1;j++)
        {
            cout<<jeu[i][j].val<<"/"<<jeu[i][j].visible<<" ";
        }
        cout<<endl;
    }
}
void affiche_visible_demineur (struct case_grille jeu[TGX+2][TGY+2]){
    int i,j;
    for (i=0;i<=TGX+1;i++)
    {
        for (j=0;j<=TGY+1;j++)
        {
            if (jeu[i][j].visible==true)
                cout<<jeu[i][j].val<<" ";
            else cout<<"- ";
        }
        cout<<endl;
    }
}
void affiche_zone_jeu_visible_demineur (struct case_grille
jeu[TGX+2][TGY+2]){
    int i,j;
    for (i=1;i<=TGX;i++)
    {
        for (j=1;j<=TGY;j++)
        {
            if (jeu[i][j].visible==true)
                cout<<jeu[i][j].val<<" ";
            else cout<<"- ";
        }
        cout<<endl;
    }
}
void pose_bombes(struct case_grille jeu[TGX+2][TGY+2]){
    int i,x,y;
    for (i=0;i<MINES;i++)
    {
        do
        {
            x = rand () % TGX + 1;
            y = rand () % TGY + 1;
        } while (jeu[x][y].val!=0);
        jeu[x][y].val = -1 ;
```

```
    }
}
void remplit_demineur(struct case_grille jeu[TGX+2][TGY+2]){
    int i,j,k,l;
    for (i=1;i<=TGX;i++)
    {
        for(j=1;j<=TGY;j++)
        {
            if (jeu[i][j].val!=-1)
            {
                for(k=i-1 ; k<=i+1 ; k++)
                {
                    for (l=j-1;l<=j+1;l++)
                    {
                        if (jeu[k][l].val==-1)
                            jeu[i][j].val++;
                    }
                }
            }
        }
    }
}
int choix_case (struct case_grille jeu[TGX+2][TGY+2]){
    int l,c;
    do
    {
        cout<<"Donnez les coordonnées de la case a sonder : "<<endl;
        cin>>l>>c;
    }
    while (l<1 || l>TGX || c<1 || c>TGY || jeu[l][c].visible != false);
    jeu[l][c].visible = true ;
    return jeu[l][c].val;
}
int main (void)
{
    srand(time(NULL));
    struct case_grille demineur[TGX+2][TGY+2];
    int cases_restantes = TGX * TGY - MINES ;
    int coup ;
    init_demineur(demineur);
    pose_bombes(demineur);
    remplit_demineur(demineur);
    affiche_tout_demineur(demineur);//pour pouvoir tricher !
    do
    {
        coup = choix_case(demineur);
        affiche_zone_jeu_visible_demineur(demineur);
        if (coup!=-1) cases_restantes--;
        cout<<endl<<"Il reste "<<cases_restantes<<" cases à trouver"<<endl;
    }while (cases_restantes!=0 && coup!=-1);
    if (cases_restantes==0)
        cout<<"gagne !"<<endl;
    else cout<<"perdu !"<<endl;
    return 0;
}
```

Une autre version des palindromes.

Nous avons conçu en TD un algorithme permettant de dire si un mot est un palindrome. Cet algorithme exploitait le sous-programme `miroir`. Nous allons à présent définir une autre version de ce programme `palindrome` en suivant les étapes suivantes :

1. Écrire l'algorithme d'un sous-programme **`premder`** permettant de constituer une chaîne de caractères en regroupant les caractères de la manière suivante : le premier et le dernier caractère, puis le deuxième et l'avant-dernier, etc.
   Exemple : abcdefg → agbfced
2. On constate que si on applique le sous-programme précédent à un palindrome (exemple : laval → llaav), la chaîne résultat est composée de paires de caractères identiques (sauf le dernier dans le cas impair). Écrire une fonction booléenne **`estpalindrome`** permettant de vérifier si une chaîne passée en paramètre est un palindrome, en utilisant **`premder`**.
3. Écrire le programme principal permettant de saisir une chaîne de caractères, et de vérifier à l'aide des questions précédentes si c'est un palindrome ou non.

```
#include <iostream>
#include <string.h>
const int CHMAX = 64 ;
using namespace std;

void premder (char chaine[CHMAX], char res[CHMAX]){
    int lg = strlen (chaine);
    int i,j=0;
    for (i=0;i<lg/2;i++)
    {
        res[j] = chaine[i];
        res[j+1] = chaine[lg-i-1];
        j=j+2;
    }
    res[j] = '\0';

}
bool estpalinrome (char chaine[CHMAX])
{
    char res[CHMAX];
    int i, lg;
    premder(chaine,res);
    lg = strlen(res);
    for (i=0;i<lg;i+=2)
    {
        if (res[i]!=res[i+1])
            return false;
    }
    return true;
}
int main (void)
{
    char ch[CHMAX];
    cout<<"Donnez une chaine de caracteres"<<endl;
    cin>>ch;
    if (estpalinrome(ch)) cout<<ch<<" est un palindrome"<<endl;
    else cout<<ch<<" n'est pas un palindrome"<<endl;
    return 0;
}
```