RABIARIVELO Ilo Andrianaly du groupe 103
BOHEC Bryan du groupe 103

PROJET D' INITIATION AU DÉVELOPPEMENT : **LE DÉMINEUR**

Année scolaire : 2021/2022

Sommaire

Introduction au projet :

Qu’est ce qu’un démineur ?

Le Démineur (Minesweeper) est un jeu vidéo de réflexion dont le but est de localiser des mines cachées dans une grille représentant un champ de mines virtuel, avec pour seule indication le nombre de mines dans les zones adjacentes.

Sujet et Objectif :

Le sujet consistait à programmer un démineur fonctionnant avec 5 commandes. Chaque commande prend en paramètre son numéro pour effectuer son appel et la taille de la grille de jeu.

- La première commande prend aussi en entrée le nombre de mines à mettre dans la grille et pose les mines dans des positions aléatoires. Elle renvoie les dimensions de la grille, le nombre de mines et leurs positions.

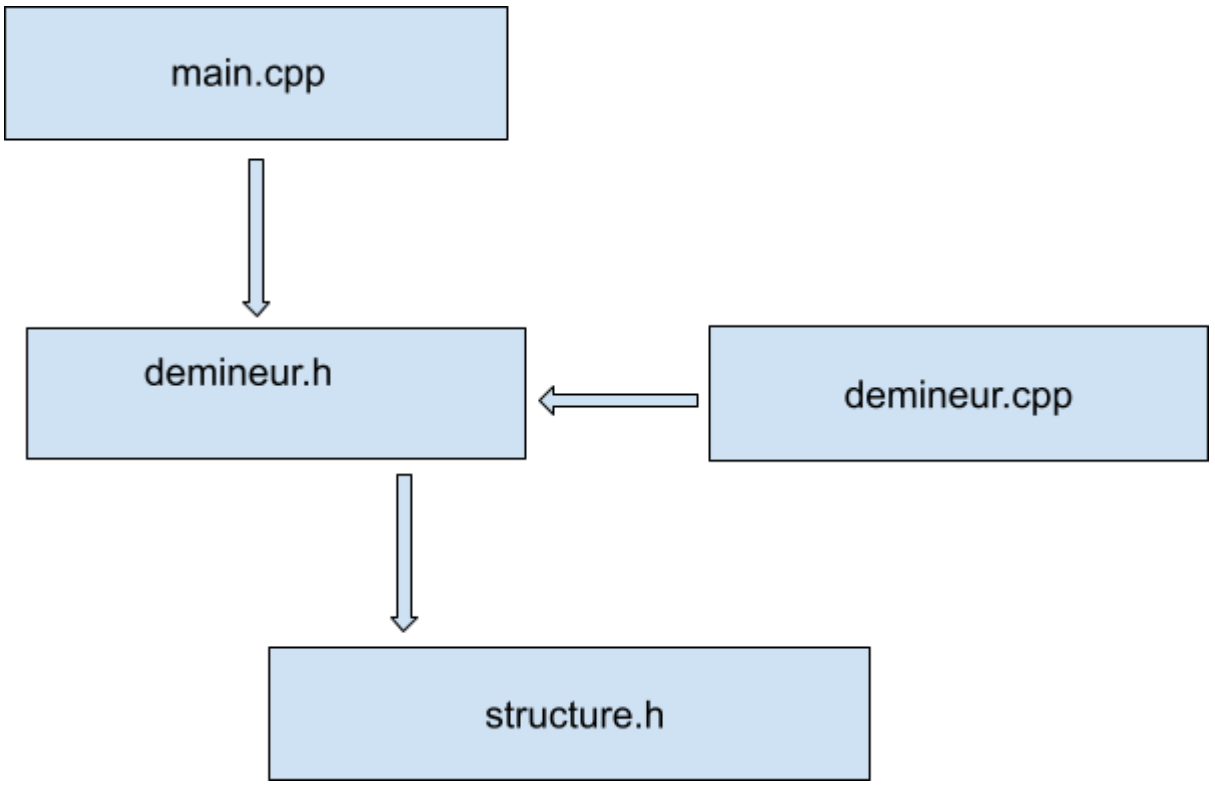
- La deuxième commande prend aussi le nombre de mines à poser mais l’utilisateur doit aussi les placer en indiquant leurs positions dans la grille. Il peut aussi jouer un coup dans la grille en indiquant le nombre de coups qu’il veut faire et ensuite en mettant un ‘D’ suivis de l’endroit où il veut jouer, il peut aussi marquer un endroit qu’il pense être miné de la même manière mais avec un ‘M‘. La commande renvoie l'affichage de la partie en cours si elle n’est pas finie, ou affiche la grille avec les bombes si elle est finie.

- La troisième commande prend les mêmes paramètres d’entrée que la commande deux mais renvoie si la partie est gagnée ou pas gagnée.

- La quatrième commande prend les mêmes paramètres d’entrée que la commande deux mais renvoie si la partie est perdue ou pas perdue.

- La cinquième commande prend en paramètres une grille et renvoie un coup possible dans la grille.

Graphe de dépendance des fichiers sources :



Jeux d’essai et validation des commandes

Chaque commande à été testé par le jeu de test fourni dans un premier temps, nous permettant de vérifier que la commande implémentée fonctionne comme attendue. Ensuite, nous avons vérifié le fonctionnement avec d’autres données, pour s’assurer que les commandes marchent peu importe la taille ou l’emplacement des bombes/cases découvertes. Par ailleurs, nous avons réalisé des fonctions intermédiaires de test, notamment d’affichages, pour vérifier l’état des cases à différents niveaux.

Commande 1 :
On vérifie que la commande fonctionne en changeant la taille du tableau et le nombre de mines, puis en vérifiants que le problème soit différents à chaque fois (que les mines ne sont pas aux mêmes endroits)

Exemple de test :
in :
1 7 7 5
out :
7 7 5 13 14 15 19 34

Commande 2 :
On vérifie que les cases se démasquent et démasquent celles autour si le nombre de bombes sur les cases adjacentes est bien de 0.
On vérifie que les cases marquées se marquent bien.
Enfin, on vérifie que les bombes se révèlent lorsque la partie est terminée (si toutes les cases contenant pas de bombes sont démasquées ou si une case contenant pas de bombes est marquée).

Exemple :
in :
2 7 7 5 7 20 22 33 48 4 D17 M22 D45 M0 (une case non minée est marquée)
out :

```
7 7
 __ __ __ __ __ __ __
| x | 1 |   |   |   |   |
 __ __ __ __ __ __ __
| m | 1 |   |   | 1 | 1 |
 __ __ __ __ __ __ __
| . | 2 | 1 |   | 1 | m |
 __ __ __ __ __ __ __
| . | m | 1 |   | 1 | 2 | . |
 __ __ __ __ __ __ __
| 1 | 1 | 1 |   | 1 | m | . |
 __ __ __ __ __ __ __
|   |   |   |   | 1 | 2 | . |
 __ __ __ __ __ __ __
|   |   |   |   |   | 1 | m |
 __ __ __ __ __ __ __
```

Commande 3 :
On vérifie la sortie “game won” si la partie est gagnée. Si elle est perdue ou en cours : “game not won”

exemple :
in1 :
2 7 7 5 7 20 22 33 48 4 D17 M22 D45 M0 (partie perdue)
out1 :
game not won

in2 :
3 7 7 5 7 20 22 33 48 7 D17 D0 D14 D34 D41 D21 D27
out2 :
game won

Commande 4 :
Même chose avec la partie perdue
in :
2 7 7 5 7 20 22 33 48 4 D17 M22 D45 M0
out :
game lost

Commande 5 :
On vérifie que le coup donné est effectivement possible (la case n’est pas démasquée).

in :

```
5 7 7
 ___ ___ ___ ___ ___ ___ ___
| . | 1 |   |   |   |   |   |
 ___ ___ ___ ___ ___ ___ ___
| . | 1 |   |   |   | 1 | 1 |
 ___ ___ ___ ___ ___ ___ ___
| . | 2 | 1 |   |   | 1 | . |
 ___ ___ ___ ___ ___ ___ ___
| . | . | 1 |   | 1 | 2 | . |
 ___ ___ ___ ___ ___ ___ ___
| 1 | 1 | 1 |   | 1 | . | . |
 ___ ___ ___ ___ ___ ___ ___
|   |   |   |   | 1 | 2 | . |
 ___ ___ ___ ___ ___ ___ ___
|   |   |   |   |   | 1 | . |
  ___ ___ ___ ___ ___ ___ ___
```

out :
D27

Bilan du projet :

Les difficultés rencontrées :

A la première lecture du sujet, le projet nous a paru un peu facile mais c'est lorsque nous nous sommes mis à travailler qu'on a commencé à voir la difficulté. Une des première difficulté était l'initialisation du tableau, en effet, on avait du mal à trouver les structures puis à faire l'allocation dynamique. Ensuite, il a fallu gérer le temps de travail et une autre serait peut-être le travail d'équipe, en plus des cours il fallait trouver le temps de réfléchir au projet et de tenter des algorithmes. Au début on travaillait chacun de notre côté et c'est lorsqu'on a fait une mise en commun de nos codes que nous nous sommes aperçu qu'ils étaient différents. Du coup, il fallait se mettre d'accord sur la meilleure façon de coder les fonctions.

Ce que nous avons réussi :

Nous avons réussi à réutiliser des notions vues en cours (notamment pour la création de la grille avec les allocations dynamiques). Et à utiliser les algorithmes conseillés par le professeur durant les amphis d'algorithmique. Les fonctions sont assez compactées et restent plus ou moins simples.

Ce que nous pouvons améliorer :

Concernant ce que nous pouvons améliorer, il y a la structure des fichiers et la commande 5. En effet, la commande cinq ne fait que donner des coups aléatoires dans les zones non découvertes. Nous n'avons pas réussi à créer un algorithme qui trouve une case non minée en analysant les nombres de bombes autour des cases.
Par ailleurs, nous n'étions pas sûr s' il fallait répartir les structures "case et "grille" dans deux fichier .h différents.

Annexes : sources du projet

Structure.h :

```
/**
 * @file structure.h
 * @author Bryan Bohec, Ilo Rabiarivelo
 * Projet demineur
 */



using namespace std;

#define lg 30 //longueur de tableau
enum ETAT{PERDUE = 1, GAGNEE = 2, ENCOURS = 3};


struct Case { // Chaque case
    bool minee; //mine ou pas mine
    bool demasquee; //démasqué ou pas démasqué
    bool marque; //marqué ou pas marqué
    int bombesautour;
};

struct Grille { //grille de jeu
    int totalcase; // nombre de case total
    int longueur;
    int largeur;
    Case** tab; //allocation dynamique pour stocker la grille
};
```

demineur.h :

```
/**
 * @file demineur.h
 * @author Bryan Bohec, Ilo Rabiarivelo
 * Projet demineur
 */

#include"structure.h"


 /**
  * @brief Initialisation de la grille
  * @param[in-out] g : la grille
  * @param[in] lon : la longueur
  * @param[in] lar : la largeur
  */
void init_probleme(Grille& g, unsigned int lon, unsigned int lar); //initialise grille de jeu


 /**
  * @brief Destruction en mémoire de la grille
  * @param[in-out] g : la grille
  */
void detruire(Grille& g);

/**
 * @brief Initialise les mines dans la grille dans l'ordre
 * @param[in-out] g : la grille
 */
void ini_mines(Grille&, unsigned int mine);


/**
 * @brief Mélange les cases de la grille au hasard
 * @param[in-out] g : la grille
 */
void melanger(Grille& g);


/**
 * @brief Affiche le numéro des cases ou il y a des mines
 * @param[in] g : la grille
 */
void probleme(Grille& g);



/**
 * @brief Ajoute à chaque case le nombre de bombes autour de celle-ci
 * @param[in-out] g : la grille
 */
void ajout_nombres_bombes_autour(Grille& g);

/**
 * @brief Lis les mines et les ajoute a la grille
 * @param[in-out] g : la grille
 * @param[in] nbMines : le nombre de mines
 */
void lecture_mines(Grille& g, unsigned int nbMines);


/**
 * @brief Lis les coups et les ajoute à la grille
 * @param[in-out] g : la grille
 * @param[in] nbCoups : le nombre de coups
 */
void lecture_coups(Grille& g, unsigned int nbCoups);

/**
 * @brief Démasque la case et celles autour récursivement si le nombre de bombes autour est 0
 * @param[in-out] g : la grille
```

```
 * @param[in] x : numéro de la ligne de la case
 * @param[in] y : numéro de la colonne de la case
 */
void demasquer(Grille& g, int x, int y);

/**
 * @brief Renvoie l'état de la partie (gagnée, perdue ou en cours)
 * @param[in] g : la grille
 * @param[in] nbMines : le nombre de mines
 * @return l'état de la partie
 */
int won_or_lost(const Grille& g, int nbMines);

/**
 * @brief Affiche la grille lorsque la partie n'est pas perdue (sans les mines)
 * @param[in] g : la grille
 */
void affiche_grille_entiere(const Grille& g);

/**
 * @brief Affiche la grille lorsque la partie est perdue (avec les mines)
 * @param[in] g : la grille
 */
void affiche_grille_mines(const Grille& g);

/**
 * @brief Affiche un nouveau coup pouvant être joué à partir d'une grille
 */
void coup_possible();
```

main.cpp :

```
/**
 * @file main.cpp
 * @author Bryan Bohec, Ilo Rabiarivelo
 * Projet demineur
 */

#include "demineur.h"
#include <time.h>
#include <cstring>
#include <iostream>
#include <cstdlib>


int main() {

	Grille grille;
	int commande;
	unsigned int longueur;
	unsigned int largeur;
	unsigned int mine;
	unsigned int nbCoups;
	cin >> commande;
	srand(time(NULL));

	switch(commande) {
	case 1: // Commande 1
		cin >> longueur;
		cin >> largeur;
		cin >> mine ;
		init_probleme(grille, longueur, largeur);
		cout << longueur << " " << largeur << " ";
		ini_mines(grille, mine);
		melanger(grille);
		cout << mine << " ";
		probleme(grille);
		cout << endl;
		detruire(grille);
		break;

	case 2: // Commande 2
		cin >> longueur;
		cin >> largeur;
		cin >> mine;
		init_probleme(grille, longueur, largeur);
		cout << longueur << " " << largeur << endl;
		lecture_mines(grille,mine);
		ajout_nombres_bombes_autour(grille);   //ajoute le nombre de bombes autour d'une case
		cin >> nbCoups;
		lecture_coups(grille, nbCoups);   //Met les cases démasquées et marquées
		if (won_or_lost(grille,mine) == ENCOURS) {
			affiche_grille_entiere(grille);
		}
		else {
			affiche_grille_mines(grille);
		}
		detruire(grille);
		break;

	case 3: //Comande 3
		cin >> longueur;
		cin >> largeur;
		cin >> mine; // nombre de mines
		init_probleme(grille, longueur, largeur);
		lecture_mines(grille, mine);
		ajout_nombres_bombes_autour(grille);
		cin >> nbCoups;
		lecture_coups(grille, nbCoups);
		if (won_or_lost(grille, mine) == GAGNEE) {
```

```
                        cout << "game won" << endl;
                }
                else {
                        cout << "game not won" << endl;
                }
                detruire(grille);
                break;

        case 4: //Commande 4
                cin >> longueur;
                cin >> largeur;
                cin >> mine; // nombre de mines
                init_probleme(grille, longueur, largeur);
                lecture_mines(grille, mine);
                ajout_nombres_bombes_autour(grille);
                cin >> nbCoups;
                lecture_coups(grille, nbCoups);
                if (won_or_lost(grille, mine) == PERDUE) {
                        cout << "game lost" << endl;
                }
                else {
                        cout <<  "game not lost" << endl;
                }
                detruire(grille);
                break;

        case 5 : //Commande 5
                coup_possible();
                cout << endl;
        }


}
```

demineur.cpp :

```
/**
 * @file demineur.cpp
 * @author Bryan Bohec, Ilo Rabiarivelo
 * Projet demineur
 */

#include "demineur.h"
#include <cstring>
#include <iostream>
#include <cstdlib>
#include <string>
using namespace std;



/**
 * @brief Initialisation de la grille
 * @param[in-out] g : la grille
 * @param[in] lon : la longueur
 * @param[in] lar : la largeur
 */
void init_probleme(Grille& g, unsigned int lon, unsigned int lar) {
    g.longueur = lon;
    g.largeur = lar;
    g.totalcase = lon * lar;
    g.tab=new Case*[g.longueur];

    for (unsigned int i = 0; i < g.longueur; i ++) {
        g.tab[i] = new Case[g.largeur];
    }

    for (unsigned int x = 0; x < g.longueur; x++) {
        for (unsigned int y = 0; y < g.largeur; y++) {
            g.tab[x][y].minee = false;
            g.tab[x][y].demasquee = false;
            g.tab[x][y].marque = false;
            g.tab[x][y].bombesautour = 0;
        }
    }
}






/**
 * @brief Initialise les mines dans la grille dans l'ordre
 * @param[in-out] g : la grille
 */
void ini_mines(Grille& g, unsigned int mine) { // On met les mines dans l'ordre
  for (int i = 0, j = 0, k=0; k < mine; i++) {

    if (i == g.largeur) {
      j++;
      i = 0;
    }
    g.tab[j][i].minee = true;
    k++;
  }


}


/**
 * @brief Mélange les cases de la grille au hasard
 * @param[in-out] g : la grille
```

```
 */
void melanger(Grille& g) {          // On melange les cases des grilles au hasard
  for (int i = 0; i < 100; i++) {
    int longueur1, largeur1, longueur2, largeur2;
    longueur1 = rand()%(g.longueur-1 - 0 + 1) + 0;
    largeur1 = rand()%(g.largeur-1 - 0 + 1) + 0;
    longueur2 = rand()%(g.longueur-1 - 1 + 1) + 0;
    largeur2 = rand()%(g.largeur-1 - 1 + 1) + 0;
    bool tmp = g.tab[longueur1][largeur1].minee;
    g.tab[longueur1][largeur1].minee = g.tab[longueur2][largeur2].minee;
    g.tab[longueur2][largeur2].minee = tmp;

    //cout << endl << x << " " << y << endl; //test pour voir si les nombres sont bien générés
  }

}


/**
 * @brief Affiche le numéro des cases ou il y a des mines
 * @param[in] g : la grille
 */
void probleme(Grille& g) {
  for (unsigned int x = 0; x < g.longueur; x++) {
        for (unsigned int y = 0; y < g.largeur; y++) {
            if (g.tab[x][y].minee == true) {
                cout << (x * g.largeur) + y << " ";
            }
        }
  }
}




/**
 * @brief Destruction en mémoire de la grille
 * @param[in-out] g : la grille
 */
void detruire(Grille& g) {
    delete[]g.tab;
    g.tab = NULL;
}




/**
 * @brief Ajoute à chaque case le nombre de bombes autour de celle-ci
 * @param[in-out] g : la grille
 */
void ajout_nombres_bombes_autour(Grille& g) {
            for (int x = 0; x < g.longueur; x++) {
                        for (int y = 0; y < g.largeur; y++) {
                                    if (g.tab[x][y].minee == true) {

                                                if ((x + 1 >= 0 && x + 1 < g.longueur) && (y >= 0 && y < g.largeur)) {   // bas    Si les
coordonnées sont dans le tableau uniquement, on incrémente
                                                            g.tab[x + 1][y].bombesautour += 1;
                                                }
                                                if ((x - 1 >= 0 && x - 1 < g.longueur) && (y >= 0 && y < g.largeur)) {   // haut
                                                            g.tab[x - 1][y].bombesautour += 1;
                                                }
                                                if ((x >= 0 && x < g.longueur) && (y + 1 >= 0 && y + 1 < g.largeur)) {   // droite
                                                            g.tab[x][y + 1].bombesautour += 1;
                                                }
                                                if ((x >= 0 && x < g.longueur) && (y - 1 >= 0 && y - 1 < g.largeur)) {   // gauche
                                                            g.tab[x][y - 1].bombesautour += 1;
                                                }
                                                if ((x + 1 >= 0 && x + 1 < g.longueur) && (y + 1 >= 0 && y + 1 < g.largeur)) {   // bas
droit
                                                            g.tab[x + 1][y + 1].bombesautour += 1;
```

```
                                        }
                                        if ((x + 1 >= 0 && x + 1 < g.longueur) && (y - 1 >= 0 && y - 1 < g.largeur)) {   // bas
gauche
                                                g.tab[x + 1][y - 1].bombesautour += 1;
                                        }
                                        if ((x - 1 >= 0 && x - 1 < g.longueur) && (y + 1 >= 0 && y + 1 < g.largeur)) {   // haut
droit
                                                g.tab[x - 1][y + 1].bombesautour += 1;
                                        }
                                        if ((x - 1 >= 0 && x - 1 < g.longueur) && (y - 1 >= 0 && y - 1 < g.largeur)) {   // haut
gauche
                                                g.tab[x - 1][y - 1].bombesautour += 1;
                                        }


                                }
                        }
                }
        }
```

```
/**
 * @brief Lis les mines et les ajoute a la grille
 * @param[in-out] g : la grille
 * @param[in] nbMines : le nombre de mines
 */
void lecture_mines(Grille& g, unsigned int nbMines) {
  for (int i = 0; i < nbMines; i++) {
    int numMine = 0;
    cin >> numMine;
    g.tab[numMine/g.largeur][numMine % g.largeur].minee = true;
  }
}
```

```
/**
 * @brief Lis les coups et les ajoute à la grille
 * @param[in-out] g : la grille
 * @param[in] nbCoups : le nombre de coups
 */
void lecture_coups(Grille& g, unsigned int nbCoups) {
  char coup;
  int numeroCase;

  //lecture des cases démasquées
  for (int i = 0; i < nbCoups; i++) {
    cin >> coup >> numeroCase;
    if (coup == 'D') {
      demasquer(g, numeroCase / g.largeur, numeroCase % g.largeur);
    }

    //lecture des cases marquées
    if (coup == 'M') {
      g.tab[numeroCase/g.largeur][numeroCase % g.largeur].marque = true;
    }
  }
}
```

```
/**
 * @brief Affiche la grille lorsque la partie n'est pas perdue (sans les mines)
 * @param[in] g : la grille
 */
void affiche_grille_entiere(const Grille& g) {
    for (unsigned int i = 0; i < g.longueur; i++) {
        for (unsigned int j = 0; j < g.largeur; j++) {
            cout << " " << "___";
            if (j == g.largeur - 1) {
```

```
            cout << endl;
          }
        }
        for (unsigned int j = 0; j < g.largeur; j++) {

          if (g.tab[i][j].demasquee == true) {
            if (g.tab[i][j].bombesautour == 0) {

              cout<<"|"<<" "<< " " << " ";
            }
            else {

              cout<<"|"<<" "<< g.tab[i][j].bombesautour << " ";
            }
          }


          else if (g.tab[i][j].marque == true) {

              cout<<"|"<<" "<< "x" << " ";
          }
          else {

              cout<<"|"<<" "<< "." << " ";
          }

        }
        cout <<"|"<< endl;
    }
    for (unsigned int j = 0; j < g.largeur; j++) {
        cout << " " << "___";
    }
    cout << endl;
}




/**
 * @brief Démasque la case et celles autour récursivement si le nombre de bombes autour est 0
 * @param[in-out] g : la grille
 * @param[in] x : numéro de la ligne de la case
 * @param[in] y : numéro de la colonne de la case
 */
void demasquer(Grille& g, int x, int y) {

    if (x < 0 || x >= g.longueur || y < 0 || y >= g.largeur)        // ça s'arrete avant de sortir du tableau
        return;

     if (g.tab[x][y].bombesautour != 0)  {                 // ça demasque la case puis ça s'arrete si la case n'a pas bombesautour = 0;
        g.tab[x][y].demasquee = true;
        return;
     }
     if (g.tab[x][y].demasquee == true)                    // ça s'arrete si la case est déja démasquée
         return;


    g.tab[x][y].demasquee = true;


    demasquer(g, x+1, y);
    demasquer(g, x-1, y);
    demasquer(g, x, y+1);
    demasquer(g, x, y-1);

    demasquer(g, x+1, y +1);
    demasquer(g, x+1, y - 1);
    demasquer(g, x-1, y + 1);
    demasquer(g, x-1, y - 1);
```

```
}


/**
 * @brief Renvoie l'état de la partie (gagnée, perdue ou en cours)
 * @param[in] g : la grille
 * @param[in] nbMines : le nombre de mines
 * @return l'état de la partie
 */
int won_or_lost(const Grille& g, int nbMines) {
    //partie perdue
    for (unsigned int x = 0; x < g.longueur; x++) {
        for (unsigned int y = 0; y < g.largeur; y++) {
            if ((g.tab[x][y].marque == true && g.tab[x][y].minee == false) || (g.tab[x][y].minee == true && g.tab[x][y].demasquee ==
true)) {
                return PERDUE;
            }
        }
    }


    //partie gagnée ou partie pas finie

    int nBCasesAdecouvrir = g.totalcase - nbMines;
    int casesDecouvertes = 0;

    for (unsigned int x = 0; x < g.longueur; x++) {
        for (unsigned int y = 0; y < g.largeur; y++) {
            if (g.tab[x][y].demasquee == true) {
                casesDecouvertes++;
            }
        }
    }
    if (nBCasesAdecouvrir == casesDecouvertes) {;
        return GAGNEE;
    }
    if (nBCasesAdecouvrir != casesDecouvertes) {
        return ENCOURS;
    }

}




/**
 * @brief Affiche la grille lorsque la partie est perdue (avec les mines)
 * @param[in] g : la grille
 */
void affiche_grille_mines(const Grille& g) {
    for (unsigned int i = 0; i < g.longueur; i++) {
        for (unsigned int j = 0; j < g.largeur; j++) {
            cout << " " << "___";
            if (j == g.largeur-1) {
                cout << endl;
            }
        }
        for (unsigned int j = 0; j < g.largeur; j++) {
          if (g.tab[i][j].minee == true){
          cout<<"|"<<" "<< "m" << " " ;
         }
         else if (g.tab[i][j].demasquee == true) {

            if (g.tab[i][j].bombesautour == 0) {
             cout<<"|"<<" "<< " " << " " ;
            }
            else {
             cout<<"|"<<" "<< g.tab[i][j].bombesautour << " " ;
            }
         } else if (g.tab[i][j].marque == true) {
          cout<<"|"<<" "<< "x" << " " ;
         }
         else {
```

```
            cout << "|" << " " << "." << " " ;
          }
        }
        cout << "|" << endl;

    }
    for (unsigned int j = 0; j < g.largeur; j++) {
        cout << " " << "___";
    }
    cout << endl;
}



/**
 * @brief Affiche un nouveau coup pouvant être joué à partir d'une grille
 */
void coup_possible() {
    int longueur, largeur;
    cin >> longueur >> largeur;
    string ligne;

    getline(cin, ligne);

    int index = 0;
    int* tableau;
    tableau = new int[largeur * longueur];
    int indextableau = 0;

    for (int i = 0; i < longueur; i++) {


        getline(cin, ligne);

        getline(cin, ligne);

        for (int j = 2; j < (largeur * 4) - 1; j += 4) {
            if (ligne[j] == '.') {
                tableau[indextableau] = index;
                indextableau++;
                index++;


            }
            else {
                index++;
            }
        }


    }


    cout << "D" << tableau[rand() % (indextableau - 1 - 0 + 1) + 0];

    delete []tableau;
}
```