

Rapport SAé 1.02

Comparaison d'approches algorithmiques



Xingtong Lin
Vanessa Pham
Groupe 104

Novembre 2021 Décembre 2022

Table des matières

Introduction	3
Graphes de dépendance des fichiers	4
Les jeux d'essai	5
Bilan du Projet	8
Annexe	9
TestDémineur.cpp	9
DémineurOperations.cpp	9
DemineurOperations.h	17
DémineurSousF.cpp	18
DemineurSousF.h	21

Introduction

Le projet demandé consiste à programmer un démineur sous console, il doit pouvoir exécuter une suite d'instruction après l'appel d'une commande numéroté de 1 à 5:

1. Problème, il doit permettre à l'utilisateur de donner les dimensions de la grille de jeu et le nombre de mines présentes, la commande doit enregistrer les données et donner les positions des n mines aléatoirement.

On doit pouvoir rentrer par exemple :

1 4 6 5

et obtenir :

4 6 5 1 5 12 7 19

en rouge : l'appel de l'opération

en vert : le nombre de lignes puis de colonnes

en orange : la position de mines donnés au hasard

Pour l'historique de coup joué le programme doit pouvoir enregistrer les coups composés d'une lettre 'D' pour démasquer et 'M' marqué, et de la position ou le coup est produit.

exemple :

4 D15 M5 D0 M13

en bleu : le nombre de coups prévu puis des coups.

2.Grille, le programme doit pouvoir afficher le nombre de lignes et de colonne puis afficher une grille structurée de "I" et de "---", et des dimensions rentrées en indiquant les marques résultant des coups dans l'historique de coup.

par exemple :

2 4 6 5 1 5 12 7 19 4 D15 M5 D0 M13

en rouge : l'appel de l'opération

en vert : le nombre de lignes puis de colonnes

en orange : la position de mines donnés au clavier

en bleu : le nombre de coups prévu puis des coups en rose.

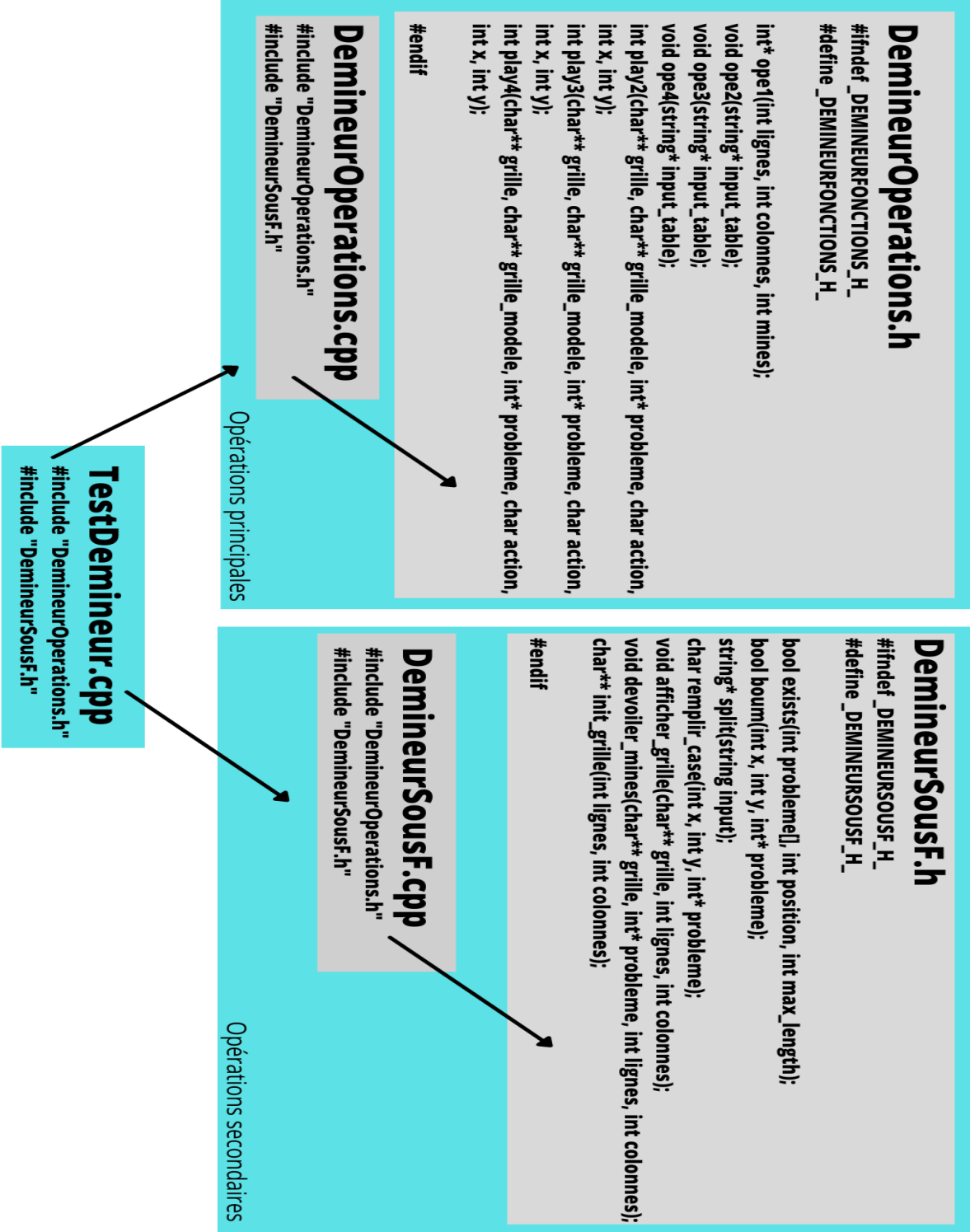
Il ne faut pas oublier une fonction pour connaître le nombre de mines autour d'une case démasquée, et libérer les cases autour d'une case démasquée, on peut s'inspirer de la fonction "Flood Fill".

Ensuite des fonctions conditionnelle en commande 3 et 4, pour savoir si un jeu, un problème est gagné ou non selon les positions des coups.

et 5.un nouveau coup aléatoire donné par la machine mais elle doit s'assurer que le nouveau coup qu'elle donne est valide, c'est-à-dire que la case est non démasquée ou déjà marquée.

Nous avons atteint l'opération 4.

Graphes de dépendance des fichiers





Les jeux d'essai

Opération 1







<div data-bbox="252 353 563 398">in1.txt - Bloc-notes</div> <div data-bbox="252 421 577 465">Fichier Edition Forma</div> <div data-bbox="252 465 391 510">1 4 6 5</div>	<div data-bbox="798 331 1185 376">monout1.txt - Bloc-notes</div> <div data-bbox="798 398 1219 443">Fichier Edition Format Affic</div> <div data-bbox="798 443 1157 488">4 6 5 18 12 23 5 4</div>
In1	Out1

Opération 2







<div data-bbox="228 734 488 779">in2a.txt - Bloc-notes</div> <div data-bbox="228 801 699 846">Fichier Edition Format Affichage Aide</div> <div data-bbox="228 846 730 891">2 4 6 5 1 5 12 7 19 3 D15 M5 D0</div>	<div data-bbox="826 734 1075 779">in2b.txt - Bloc-notes</div> <div data-bbox="826 801 1273 846">Fichier Edition Format Affichage Aide</div> <div data-bbox="826 846 1369 891">2 4 6 5 1 5 12 7 19 4 D15 M5 D0 D19</div>
<div data-bbox="228 1003 596 1048">monout2a.txt - Bloc-notes</div> <div data-bbox="228 1070 762 1115">Fichier Edition Format Affichage Aide</div> <div data-bbox="228 1115 683 1541"> <div data-bbox="228 1115 284 1160">4 6</div> <div data-bbox="228 1182 683 1541"> <div data-bbox="228 1182 683 1227">-----</div> <div data-bbox="228 1227 683 1272"> 2 . 2 1 x </div> <div data-bbox="228 1272 683 1317">-----</div> <div data-bbox="228 1317 683 1361"> . . 2 1 1 </div> <div data-bbox="228 1361 683 1406">-----</div> <div data-bbox="228 1406 683 1451"> . . 2 </div> <div data-bbox="228 1451 683 1496">-----</div> <div data-bbox="228 1496 683 1541"> . . 1 </div> <div data-bbox="228 1541 683 1585">-----</div> </div> </div>	<div data-bbox="826 1003 1211 1048">monout2b.txt - Bloc-notes</div> <div data-bbox="826 1070 1377 1115">Fichier Edition Format Affichage Aide</div> <div data-bbox="826 1115 1297 1518"> <div data-bbox="826 1115 882 1160">4 6</div> <div data-bbox="826 1182 1297 1518"> <div data-bbox="826 1182 1297 1227">-----</div> <div data-bbox="826 1227 1297 1272"> 2 m 2 1 m </div> <div data-bbox="826 1272 1297 1317">-----</div> <div data-bbox="826 1317 1297 1361"> . m 2 1 1 </div> <div data-bbox="826 1361 1297 1406">-----</div> <div data-bbox="826 1406 1297 1451"> m . 2 </div> <div data-bbox="826 1451 1297 1496">-----</div> <div data-bbox="826 1496 1297 1541"> . m 1 </div> <div data-bbox="826 1541 1297 1585">-----</div> </div> </div>
<div data-bbox="228 1641 475 1686">in2c.txt - Bloc-notes</div> <div data-bbox="228 1709 675 1753">Fichier Edition Format Affichage Aide</div> <div data-bbox="228 1753 762 1798">2 4 6 5 1 5 12 7 19 4 D15 M5 D0 M13</div>	<div data-bbox="826 1641 1107 1686">in2d.txt - Bloc-notes</div> <div data-bbox="826 1709 1329 1753">Fichier Edition Format Affichage Aide</div> <div data-bbox="826 1753 1297 1798">2 6 6 3 30 31 35 3 M0 M5 D2</div>

 monout2c.txt - Bloc-notes Fichier Edition Format Affichage Aide 4 6 ----- 2 m 2 1 m ----- . m 2 1 1 ----- m x 2 ----- . m 1 -----	 monout2d.txt - Bloc-notes Fichier Edition Format Affichage Aide 6 6 ----- x x ----- ----- ----- ----- 2 2 1 1 1 ----- m m 1 1 m -----
--	--

Opération 3

 in3a.txt - Bloc-notes Fichier Edition Format Affichage Aide 3 4 6 5 1 5 12 7 19 3 D15 M5 D0	 in3b.txt - Bloc-notes Fichier Edition Format Affichage Aide 3 4 6 5 1 5 12 7 19 4 D15 M5 D0 D19
 monout3a.txt - Bloc-notes Fichier Edition Format Affichage Aide Game not won	 monout3b.txt - Bloc-notes Fichier Edition Format Affichage Aide Game not won Game not won
 in3c.txt - Bloc-notes Fichier Edition Format Affichage Aide 3 4 6 5 1 5 12 7 19 4 D15 M5 D0 M13	
 monout3c.txt - Bloc-notes Fichier Edition Format Affichage Aide Game not won Game not won	

Opération 4

 in4a.txt - Bloc-notes Fichier Edition Format Affichage Aide 4 4 6 5 1 5 12 7 19 3 D15 M5 D0	 in4b.txt - Bloc-notes Fichier Edition Format Affichage Aide 4 4 6 5 1 5 12 7 19 4 D15 M5 D0 D19
 monout4a.txt - Bloc-notes Fichier Edition Format Affichage Aide Game not lost Game not lost	 monout4b.txt - Bloc-notes Fichier Edition Format Affichage Aide Game not lost Game not lost Game lost
 in4c.txt - Bloc-notes Fichier Edition Format Affichage Aide 4 4 6 5 1 5 12 7 19 4 D15 M5 D0 M13	
 monout4c.txt - Bloc-notes Fichier Edition Format Affichage Aide Game not lost Game not lost Game lost	

Nous n'avons pas atteint l'Opération 5.

Pour les opérations 3 et 4, nous rencontrons des doublons, et plusieurs sorties pour une même entrée, c'est un problème que nous n'arrivons pas à corriger, cependant l'affichage le plus récent (tout en bas) est un résultat correct.

Bilan du Projet

Nous avons eu énormément de difficulté, tant dans la partie logique et écriture du code du à des lacunes personnelles.

Nous avons eu des difficultés par rapport aux termes définition de “Game won, game not won, Game lost, game not lost”. Peut-être que nous aurions dû nous familiariser avec ce jeu en jouant davantage, ce jeu qui est classique mais qui a été une découverte pour nous 2 membres du binôme.

Comme constaté plus haut dans “jeux d’essai” nous avons rencontré des problèmes pour l’affichage des résultats, peut-être aurait-il fallu utiliser une structure “pile” afin d’enregistrer les résultats sorties mais en n’affichant que le dernier empiler.

Pour améliorer le code, il faut corriger les doublons et atteindre l’Opération 5. L’opération 5 peut se faire en trouvant une structure qui attribuera un État à une case, si une certaine case a un état “inconnu”, un coup aléatoire pourra se faire sur une des cases avec l’état “Inconnu, ce qui sera un coup Légal.

Annexe

TestDémineur.cpp

```
/**
 * @TestDémineur.cpp
 * Projet SDA, Démineur
 * @author : Vanessa Pham Xingtong Lin groupe 104
 * @IUT de Paris - Rives de Seine
 * @version 09/01/22
 * @brief Tester le demineur
 */

#include <iostream>
#include <string>
#include <sstream>
#include "DemineurOperations.h"
#include "DemineurSousF.h"
using namespace std;

int main(int argc, char* argv[]) {
    // codeope, lignes, colonnes, nb_mines, <nombre de mines>, nb_pas, <nombre de
    pas>
    string input;
    getline(cin, input);
    string* input_table = split(input);

    if (input_table[0] == "1") {
        ope1(stoi(input_table[1]), stoi(input_table[2]), stoi(input_table[3]));
    }
    else if (input_table[0] == "2") {
        ope2(input_table);
    }
    else if (input_table[0] == "3") {
        ope3(input_table);
    }
    else if (input_table[0] == "4") {
        ope4(input_table);
    }
    return 0;
}
```

DémineurOperations.cpp

```
/**
 * @DémineurOperations.cpp
 * Projet SDA, Démmineur
 * @author : Vanessa Pham Xingtong Lin groupe 104
 * @IUT de Paris - Rives de Seine
 * @version 09/01/22
 * @brief les Operations du demineur
```

```

*/

#include <iostream>
#include <string>
#include <sstream>
#include "DemineurOperations.h"
#include "DemineurSousF.h"
using namespace std;

//OPERATION 1
int* ope1(int lignes, int colonnes, int mines) {
    int* probleme = new int[mines + 3];
    probleme[0] = lignes;
    probleme[1] = colonnes;
    probleme[2] = mines;
    int tmp; //random les mines
    for (int i = 3; i < mines + 3; i++) {
        tmp = (rand() % (lignes * colonnes) + 1);
        while (exists(probleme, tmp, lignes * colonnes)) {
            tmp = (rand() % (lignes * colonnes));
        }
        probleme[i] = tmp;
    };
    string c = "";
    c += to_string(lignes) + " " + to_string(colonnes) + " " + to_string(mines) + " ";
    for (int i = 3; i < mines + 3; i++) {
        c += to_string(probleme[i]) + " ";
    }
    cout << c << endl;
    return probleme;
}

//OPERATION 2
void ope2(string* input_table) {
    int ope = stoi(input_table[0]); //transforme string en int
    int lignes = stoi(input_table[1]);
    int colonnes = stoi(input_table[2]);
    int mines = stoi(input_table[3]);
    int* probleme = new int[3 + mines];
    for (int i = 1; i <= (mines + 3); i++) {
        probleme[i - 1] = stoi(input_table[i]);
    }
    int pas = stoi(input_table[3 + mines + 1]);

    int index_move = 3 + mines + 1 + 1;
    int max_index_move = 3 + mines + 1 + pas;
    // Initialisation de la grille
    char** grille = init_grille(lignes, colonnes);
    char** grille_modele = init_grille(lignes, colonnes);
    // Construction de la grille
    for (int x = 0; x < colonnes; x++) {
        for (int y = 0; y < lignes; y++) {
            grille_modele[x][y] = remplir_case(x, y, probleme);
            grille[x][y] = '.';
        }
    }
}

```

```

    }
}

while (index_move <= max_index_move) {
    string move = input_table[index_move];
    char action = move[0];
    int index = stoi(move.substr(1, move.length() - 1));
    int move_y = index / colonnes;
    int move_x = index % colonnes;
    int result = play2(grille, grille_modele, probleme, action, move_x, move_y);
    index_move++;
}afficher_grille(grille, lignes, colonnes);
cout << endl;
}

//OPERATION 3
void ope3(string* input_table) {
    int ope = stoi(input_table[0]);
    int lignes = stoi(input_table[1]);
    int colonnes = stoi(input_table[2]);
    int mines = stoi(input_table[3]);
    int* probleme = new int[3 + mines];
    for (int i = 1; i <= (mines + 3); i++) {
        probleme[i - 1] = stoi(input_table[i]);
    }
    int pas = stoi(input_table[3 + mines + 1]);

    int index_move = 3 + mines + 1 + 1;
    int max_index_move = 3 + mines + 1 + pas;
    // Initialisation de la grille
    char** grille = init_grille(lignes, colonnes);
    char** grille_modele = init_grille(lignes, colonnes);
    // Construction de la grille
    for (int x = 0; x < colonnes; x++) {
        for (int y = 0; y < lignes; y++) {
            grille_modele[x][y] = remplir_case(x, y, probleme);
            grille[x][y] = '.';
        }
    }

    while (index_move <= max_index_move) {
        string move = input_table[index_move];
        char action = move[0];
        int index = stoi(move.substr(1, move.length() - 1));
        int move_y = index / colonnes;
        int move_x = index % colonnes;
        int result = play3(grille, grille_modele, probleme, action, move_x, move_y);
        index_move++;
    }
}

//OPERATION 4
void ope4(string* input_table) {
    int ope = stoi(input_table[0]);

```

```

    int lignes = stoi(input_table[1]);
    int colonnes = stoi(input_table[2]);
    int mines = stoi(input_table[3]);
    int* probleme = new int[3 + mines];
    for (int i = 1; i <= (mines + 3); i++) {
        probleme[i - 1] = stoi(input_table[i]);
    }
    int pas = stoi(input_table[3 + mines + 1]);

    int index_move = 3 + mines + 1 + 1;
    int max_index_move = 3 + mines + 1 + pas;
    // Initialisation de la grille
    char** grille = init_grille(lignes, colonnes);
    char** grille_modele = init_grille(lignes, colonnes);
    // Construction de la grille
    for (int x = 0; x < colonnes; x++) {
        for (int y = 0; y < lignes; y++) {
            grille_modele[x][y] = remplir_case(x, y, probleme);
            grille[x][y] = '.';
        }
    }

    while (index_move <= max_index_move) {
        string move = input_table[index_move];
        char action = move[0];
        int index = stoi(move.substr(1, move.length() - 1));
        int move_y = index / colonnes;
        int move_x = index % colonnes;
        int result = play4(grille, grille_modele, probleme, action, move_x, move_y);
        index_move++;
    }
}

//play2 pour l'OPERATION 2
int play2(char** grille, char** grille_modele, int* probleme, char action, int x, int y) {
    int lignes = probleme[0];
    int colonnes = probleme[1];
    int mines = probleme[2];
    if (action == 'D') {
        if (boum(x, y, probleme)) {
            dévoiler_mines(grille, probleme, lignes, colonnes);
            return 0;
        }
        else if (grille_modele[x][y] >= '1' && grille_modele[x][y] <= '8' && grille[x][y]
== '.') {
            grille[x][y] = grille_modele[x][y];
        }
        // récursif
        else if (grille_modele[x][y] == ' ' && grille[x][y] == '.') {
            grille[x][y] = ' ';
            if (x > 0) { //éviter les erreur de depassement; pas soyez plus grand
que grille
                play2(grille, grille_modele, probleme, action, x - 1, y);
            }
        }
    }
}

```

```

        if (x < colonnes - 1) { //eviter les erreur de depassement; pas soyez
plus grand que grille
            play2(grille, grille_modele, probleme, action, x + 1, y);
        }
        if (y > 0) { //eviter les erreur de depassement; pas soyez plus grand
que grille
            play2(grille, grille_modele, probleme, action, x, y - 1);
        }
        if (y < colonnes - 1) { //eviter les erreur de depassement; pas soyez
plus grand que grille
            play2(grille, grille_modele, probleme, action, x, y + 1);
        }
        if (x > 0 && y > 0) { //eviter les erreur de depassement; pas soyez
plus grand que grille
            play2(grille, grille_modele, probleme, action, x - 1, y - 1);
        }
        if (x > 0 && y < colonnes - 1) { //eviter les erreur de depassement;
pas soyez plus grand que grille
            play2(grille, grille_modele, probleme, action, x - 1, y + 1);
        }
        if (x < colonnes - 1 && y > 0) { //eviter les erreur de depassement;
pas soyez plus grand que grille
            play2(grille, grille_modele, probleme, action, x + 1, y - 1);
        }
        if (x < colonnes - 1 && y < colonnes - 1) { //eviter les erreur de
depassement; pas soyez plus grand que grille
            play2(grille, grille_modele, probleme, action, x + 1, y + 1);
        }
    }
}
else if (action == 'M') {
    if (grille_modele[x][y] != 'm' && grille[x][y] == '.') {
        grille[x][y] = 'x';
        dévoiler_mines(grille, probleme, lignes, colonnes);
        return 0;
    }
    else if (grille[x][y] == '.') {
        grille[x][y] = 'x';
    }
}
return 1;
}

//play3 pour l'OPERATION 3
int play3(char** grille, char** grille_modele, int* probleme, char action, int x, int y) {
    int lignes = probleme[0];
    int colonnes = probleme[1];
    int mines = probleme[2];
    if (action == 'D') {
        if (boum(x, y, probleme)) {
            dévoiler_mines(grille, probleme, lignes, colonnes);
            cout << "Game not won" << endl;
            return 0;
        }
    }
}

```

```

else if (grille_modele[x][y] >= '1' && grille_modele[x][y] <= '8' && grille[x][y]
== '.') {
    grille[x][y] = grille_modele[x][y];
}
// récursif
else if (grille_modele[x][y] == ' ' && grille[x][y] == '.') {
    grille[x][y] = ' ';
    if (x > 0) {//eviter les erreur de depassement; pas soyez plus grand
que grille
        play3(grille, grille_modele, probleme, action, x - 1, y);
    }
    if (x < colonnes - 1) {//eviter les erreur de depassement; pas soyez
plus grand que grille
        play3(grille, grille_modele, probleme, action, x + 1, y);
    }
    if (y > 0) {//eviter les erreur de depassement; pas soyez plus grand
que grille
        play3(grille, grille_modele, probleme, action, x, y - 1);
    }
    if (y < colonnes - 1) {//eviter les erreur de depassement; pas soyez
plus grand que grille
        play3(grille, grille_modele, probleme, action, x, y + 1);
    }
    if (x > 0 && y > 0) {//eviter les erreur de depassement; pas soyez
plus grand que grille
        play3(grille, grille_modele, probleme, action, x - 1, y - 1);
    }
    if (x > 0 && y < colonnes - 1) {//eviter les erreur de depassement;
pas soyez plus grand que grille
        play3(grille, grille_modele, probleme, action, x - 1, y + 1);
    }
    if (x < colonnes - 1 && y > 0) {//eviter les erreur de depassement;
pas soyez plus grand que grille
        play3(grille, grille_modele, probleme, action, x + 1, y - 1);
    }
    if (x < colonnes - 1 && y < colonnes - 1) {//eviter les erreur de
depassement; pas soyez plus grand que grille
        play3(grille, grille_modele, probleme, action, x + 1, y + 1);
    }
}
}
else if (action == 'M') {
    if (grille_modele[x][y] != 'm' && grille[x][y] == '.') {
        grille[x][y] = 'x';
        dévoiler_mines(grille, probleme, lignes, colonnes);
        cout << "Game not won" << endl;
        return 0;
    }
    else if (boum(x, y, probleme)) {
        dévoiler_mines(grille, probleme, lignes, colonnes);
        cout << "Game not won" << endl;
        return 0;
    }
    else if (grille[x][y] == '.') {

```

```

        grille[x][y] = 'x';
    }
}
int minefind = 0;
for (int x = 0; x < colonnes; x++) {
    for (int y = 0; y < lignes; y++) {
        if ((grille[x][y] == '.') || (grille[x][y] == 'x' && grille_modele[x][y] == 'm'))
        {
            grille_modele[x][y] = remplir_case(x, y, probleme);
            minefind++;
        }
    }
}
if (minefind == mines) {
    cout << "Game Won" << endl;
}

return 1;
}

//play4 pour l'OPERATION 4
int play4(char** grille, char** grille_modele, int* probleme, char action, int x, int y) {
    int lignes = probleme[0];
    int colonnes = probleme[1];
    int mines = probleme[2];
    if (action == 'D') {
        if (boum(x, y, probleme) == true) {
            dévoiler_mines(grille, probleme, lignes, colonnes);
            cout << "Game lost" << endl;
            return 0;
        }
        else if (boum(x, y, probleme) == false) {
            cout << "Game not lost" << endl;
        }
        else if (grille_modele[x][y] >= '1' && grille_modele[x][y] <= '8' && grille[x][y]
== '.') {
            grille[x][y] = grille_modele[x][y];
        }
        // récursif
        else if (grille_modele[x][y] == ' ' && grille[x][y] == '.') {
            grille[x][y] = ' ';
            if (x > 0) {//eviter les erreur de depassement; pas soyez plus grand
que grille
                play4(grille, grille_modele, probleme, action, x - 1, y);
            }
            if (x < colonnes - 1) {//eviter les erreur de depassement; pas soyez
plus grand que grille
                play4(grille, grille_modele, probleme, action, x + 1, y);
            }
            if (y > 0) {//eviter les erreur de depassement; pas soyez plus grand
que grille
                play4(grille, grille_modele, probleme, action, x, y - 1);
            }
            if (y < colonnes - 1) {//eviter les erreur de depassement; pas soyez

```

```

plus grand que grille
        play4(grille, grille_modele, probleme, action, x, y + 1);
    }
    if (x > 0 && y > 0) {//eviter les erreur de depassement; pas soyez
plus grand que grille
        play4(grille, grille_modele, probleme, action, x - 1, y - 1);
    }
    if (x > 0 && y < colonnes - 1) {//eviter les erreur de depassement;
pas soyez plus grand que grille
        play4(grille, grille_modele, probleme, action, x - 1, y + 1);
    }
    if (x < colonnes - 1 && y > 0) {//eviter les erreur de depassement;
pas soyez plus grand que grille
        play4(grille, grille_modele, probleme, action, x + 1, y - 1);
    }
    if (x < colonnes - 1 && y < colonnes - 1) {//eviter les erreur de
depassement; pas soyez plus grand que grille
        play4(grille, grille_modele, probleme, action, x + 1, y + 1);
    }
}
}
else if (action == 'M') {
    if (grille_modele[x][y] != 'm' && grille[x][y] == '.') {
        grille[x][y] = 'x';
        dévoiler_mines(grille, probleme, lignes, colonnes);
        if (grille_modele[x][y] == 'm' && grille[x][y] == 'x')
            cout << "Game lost" << endl;
        return 0;
    }
    else if (grille[x][y] == '.') {
        grille[x][y] = 'x';
    }
}

int minefind = 0;
for (int x = 0; x < colonnes; x++) {
    for (int y = 0; y < lignes; y++) {
        if ((grille[x][y] == '.') || (grille[x][y] == 'x' && grille_modele[x][y] == 'm'))
        {
            grille_modele[x][y] = remplir_case(x, y, probleme);
            minefind++;
        }
    }
}
if (minefind == mines) {
    cout << "Game not lost" << endl;
}

return 1;
}

```



```

#ifndef _DEMINEURFONCTIONS_H_
#define _DEMINEURFONCTIONS_H_

#include <iostream>
#include <string>
#include <sstream>
using namespace std;

/**
 * OPERATION 1
 * @brief Rentrer et afficher un probleme
 * @param[in] int : le nombre de lignes.
 * @param[in] int : le nombre de colonnes.
 * @param[in] int : le nombre de mines.
 * @param[out] int : la liste des positions des mines donnés aléatoirement.
 * @pre le nombre de mines doit être entre lignes*colonnes.
 */
int* ope1(int lignes, int colonnes, int mines);

/**
 * OPERATION 2
 * @brief Rentrer un probleme, la position des mines,
 * @brief rentrer un historique de coup
 * @brief créer et affiche une grille selon le probleme et l'historique de coup.
 * @param[in] string : le probleme, les positions des mines, les coups.
 * @param[out] string : Le nombre de colonnes et lignes.
 * @param[out] string : une grille.
 */
void ope2(string* input_table);

/**
 * OPERATION 3
 * @brief similaire à l'opération 2 MAIS n'affiche QUE le resultat
 * @param[in] string : le probleme, les positions des mines, les coups.
 * @param[out] string : Resulat : "Game won" ou "Game not won"
 */
void ope3(string* input_table);

/**
 * OPERATION 4
 * @brief similaire à l'opération 2 MAIS n'affiche QUE le resultat
 * @param[in] string : le probleme, les positions des mines, les coups.
 * @param[out] string : Resulat : "Game lost" ou "Game not lost"
 */
void ope4(string* input_table);

/**
 * "PLAY2" pour l'OPERATION 2
 * @brief Est appelé par l'ope2
 * @brief remplit la grille selon le probleme et l'historique de coup
 * @param[in-out] char** : grille : la 1ere couche visible
 * @param[in-out] char** : grille_modele : la couche revelé après les coups

```

```

* @param[in-out] int* : le probleme
* @param[in] char : l'action: les coups joués
* @param[in] int : la position sur la grille en abscisse x
* @param[in] int : la position sur la grille en ordonnée y
*/
int play2(char** grille, char** grille_modele, int* probleme, char action, int x, int y);

/**
* "PLAY3" pour l'OPERATION 3
* @brief Est appelé par l'ope3
* @brief remplit la grille selon le probleme et l'historique de coup
* @brief calcule le résultat "Game won" ou "Game not won"
* @param[in-out] char** : grille : la 1ere couche visible
* @param[in-out] char** : grille_modele : la couche revelé après les coups
* @param[in-out] int* : le probleme
* @param[in] char : l'action: les coups joués
* @param[in] int : la position sur la grille en abscisse x
* @param[in] int : la position sur la grille en ordonnée y
*/
int play3(char** grille, char** grille_modele, int* probleme, char action, int x, int y);

/**
* "PLAY4" pour l'OPERATION 4
* @brief Est appelé par l'ope4
* @brief remplit la grille selon le probleme et l'historique de coup
* @brief calcule le résultat "Game lost" ou "Game not lost"
* @param[in-out] char** : grille : la 1ere couche visible
* @param[in-out] char** : grille_modele : la couche revelé après les coups
* @param[in-out] int* : le probleme
* @param[in] char : l'action: les coups joués
* @param[in] int : la position sur la grille en abscisse x
* @param[in] int : la position sur la grille en ordonnée y
*/
int play4(char** grille, char** grille_modele, int* probleme, char action, int x, int y);

#endif

```

DémineurSousF.cpp

```

/**
* @DémineurSousF.cpp
* Projet SDA, Démmineur
* @author : Vanessa Pham Xingtong Lin groupe 104
* @IUT de Paris - Rives de Seine
* @version 09/01/22
* @brief les Sous Fonctions utilisés par les Operations du demineur
*/

#include <iostream>
#include <string>
#include <sstream>
#include "DemineurOperations.h"

```

```

#include "DemineurSousF.h"
using namespace std;

//éviter depassement de grille
bool exists(int probleme[], int position, int max_length) {
    for (int i = 3; i < max_length; i++) {
        if (probleme[i] == position) {
            return true;
        }
    }
    return false;
}

// Vérifier si c'est la position de la mine
bool boum(int x, int y, int* probleme) {
    int colonnes = probleme[1];
    int mines = probleme[2];
    for (int i = 3; i < (mines + 3); i++) {
        if ((y * colonnes + x) == probleme[i])
            return true;
    }
    return false;
}

// split pour separer le cin(entree) type string
string* split(string input) {
    int nb_parameters = 1;
    for (int i = 0; input[i]; i++)
        if (input[i] == ' ')
            nb_parameters++;
    string* result = new string[nb_parameters];
    stringstream ssin(input);
    int i = 0;
    while (ssin.good() && i < nb_parameters) {
        ssin >> result[i];
        i++;
    }
    return result;
}

// Remplir une case de la grille
char remplir_case(int x, int y, int* probleme) {
    if (boum(x, y, probleme)) { return 'm'; }
    int index_max_lignes = probleme[0] - 1;
    int index_max_colonnes = probleme[1] - 1;
    int somme = 0;
    for (int i = -1; i <= 1; i++) {
        for (int j = -1; j <= 1; j++) {
            if (!(x + i) < 0 || (y + j) < 0 || (x + i) > index_max_colonnes || (y + j) >
index_max_lignes || (i == 0 && j == 0))
                if (boum(x + i, y + j, probleme))
                    somme++;
        }
    }
}

```

```

    }
    if (somme > 0) {
        char chiffre = '0' + somme; //les chiffres sont affichés en char
        return chiffre;
    }
    else {
        return ' '; //si autour de lui il y a pas de mine, afficher ' '
    }
}

//affiche grille
void afficher_grille(char** grille, int lignes, int colonnes) {
    cout << to_string(lignes) + " " << to_string(colonnes) << endl;
    for (int x = 0; x < colonnes; x++) {
        cout << " ---";
    }
    cout << endl;
    for (int y = 0; y < lignes; y++) {
        for (int x = 0; x < colonnes; x++) {
            cout << "|" << " " << grille[x][y] << " ";
        }
        cout << '|' << endl;
        for (int x = 0; x < colonnes; x++) {
            cout << " ---";
        }
        cout << endl;
    }
}

//si la case a mine affiche m
void dévoiler_mines(char** grille, int* probleme, int lignes, int colonnes) {
    for (int x = 0; x < colonnes; x++) {
        for (int y = 0; y < lignes; y++) {
            if (boum(x, y, probleme)) {
                grille[x][y] = 'm';
            }
        }
    }
}

//initialiser grille
char** init_grille(int lignes, int colonnes) {
    char** grille = new char* [colonnes];
    for (int x = 0; x < colonnes; x++) {
        grille[x] = new char[lignes];
    }
    return grille;
}

```

```

#ifndef _DEMINEURSOUSF_H_
#define _DEMINEURSOUSF_H_

#include <iostream>
#include <string>
#include <sstream>
using namespace std;

/**
 * @brief type Booléen, s'assurer que la position d'une mine ne dépasse pas la limite
 * @param[in] int : le probleme
 * @param[in] int : la position de mine
 * @param[in] int : la limite
 * @param[return] : true ou false
 */
bool exists(int probleme[], int position, int max_length);

/**
 * @brief type Booléen, Vérifier si il y a une mine à une position
 * @param[in-out] int* : le probleme
 * @param[in] int : une variable de calcule x
 * @param[in] int : une variable de calcule y
 * @param[return] : true ou false
 */
bool boum(int x, int y, int* probleme);

/**
 * @brief Séparer les int et char d'une entrée en string
 * @param[in] string : input une entrée.
 * @param[return] string : le int OU le char
 */
string* split(string input);

/**
 * @brief Remplir les cases de la grille selon le probleme et l'historique de coup
 * @param[in] int : la position en abscisse x sur la grille
 * @param[in] int : la position en ordonnée y sur la grille
 * @param[return] : un contenu : '.', ' ', 'm', 'x'.
 */
char remplir_case(int x, int y, int* probleme);

/**
 * @brief afficher la structure grille
 * @param[out] char : la structure grille
 * @param[in] int : le nombre de lignes
 * @param[in] int : le nombre de colonnes
 */
void afficher_grille(char** grille, int lignes, int colonnes);

/**
 * @brief Revele la position des mines avec un 'm' sur la grille apres le dernier coup
 * @param[out] char : la structure de la grille : la couche revelé après les coups

```

```
* @param[in] int : le nombre de lignes
* @param[in] int : le nombre de colonnes
*/
void dévoiler_mines(char** grille, int* probleme, int lignes, int colonnes);

/**
 * @brief Initialise une grille
 * @param[in] int : le nombre de lignes
 * @param[in] int : le nombre de colonnes
 */
char** init_grille(int lignes, int colonnes);

#endif
```