RAPPORT DE PROJET

MINESWEEPER

MOUELHI Ranim & BENYETTOU Imane

Groupe 105

Table des matières

Introduction

L'objectif de ce projet était de programmer le jeu du démineur, suivant les règles classiques, à partir d'un problème, qui décrit le nombre de lignes et colonnes d'une grille du démineur. On utilisera comme notation : '.' pour une case masquée non marquée, 'x' pour une case masquée marquée. Les cases démasquées contiendraient alors le nombre de mines présentes dans les cases adjacentes, ou rien si aucune mine n'est présente dans celles-ci.

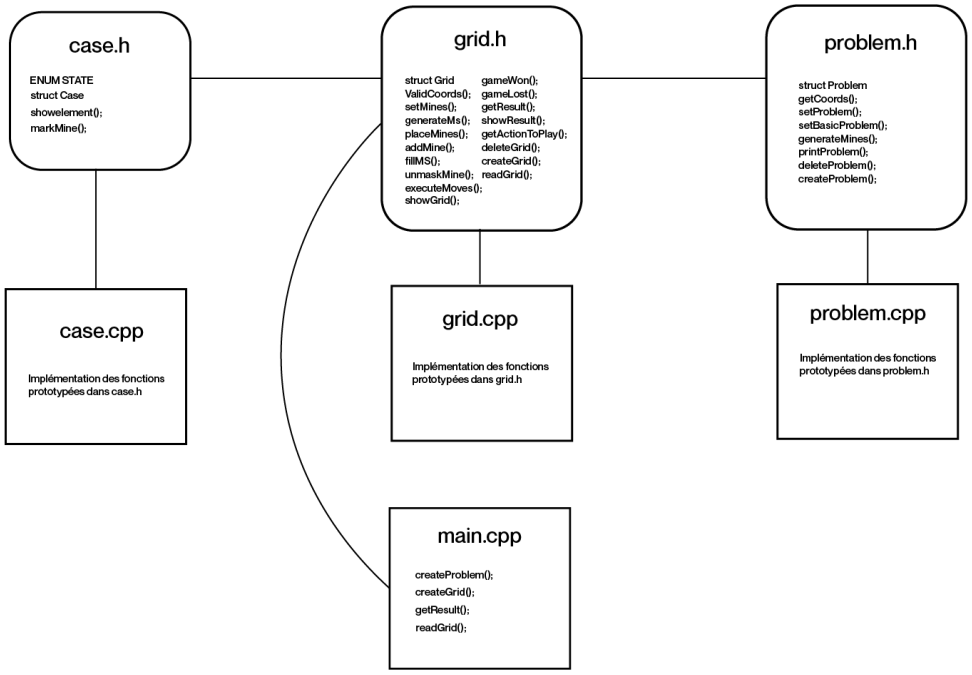
Notre programme comporte alors les 5 opérations élémentaires :

- Produire un problème à partir du nombre de lignes et colonnes
- Produire une grille à partir d'un problème et d'un historique de coup
- Déterminer si la partie est gagnée ou non à partir d'un problème et d'un historique de coups
- Déterminer si la partie est perdue à partir d'un problème et d'un historique de coups
- Produire un nouveau coup à partir d'une grille.

A noter que ce projet ne nous permettait pas d'utiliser les bibliothèques standards C++ (stack, vector, queue…) : nous avons donc dû créer nos propres types et structures.

Nous avons séparé le code de ce projet en plusieurs fichiers sources et d'en-tête, pour une lecture plus facile du code. (voir graphe des dépendances des fichiers ci-dessous).

Graphe de dépendance des fichiers sources



Bilan du projet

Difficultés rencontrés

Bien que nous pensions avoir réussi de manière générale ce projet, nous avons rencontré beaucoup de difficultés de tous types : premièrement, la compréhension du sujet et des informations données dans celui-ci, mais aussi la prise en compte des contraintes, voulant dire des restrictions au niveau de la programmation de ce jeu. Nous avons aussi eu des difficultés, comme dans la plupart des projets de code pour 'traduire' l'énoncé (ici les commandes) en implémentation concrètes : il y à eu beaucoup d'essai au niveau du prototypage et de la planification des fonctions/types avant d'obtenir quelque chose qui fonctionnait correctement.

Une autre difficulté n'étant pas des moindre, à été l'organisation en groupe : même en binôme, travailler sur un projet d'une telle ampleur (et où nous avions encore jamais été confronté) pendant la période de cours, puis en distanciel pendant les fêtes et vacances était très difficile.

Réussites

Malgré les difficultés rencontrées, techniques ou organisationnelles, nous avons pu à travers la réalisation de ce projet, implémenter et concrètement utiliser les notions acquises en cours, et en séances de TP/TD, ce qui nous à aidé à renforcer nos connaissances et notre aisance en programmation. De plus, malgré les difficultés à travailler en période chargée et à distance, nous avons tout de même maintenu un rythme assez régulier grâce à une bonne organisation.

Ce qui pourrait être amélioré

Nous pensons pouvoir améliorer la présentation de notre code en ajoutant plus de commentaires et en les détaillant encore plus, ce que nous pensions assez important notamment lors de collaboration sur un même projet. Cependant, dans ce cas de figure, nous pensions que les prototypes et implémentations étaient assez explicites, mais il serait bon que nous prenions l'habitude de bien documenter notre code.

Annexe

Code source

case.h

```
#ifndef I_MINESWEEPER_CASE_H
#define I_MINESWEEPER_CASE_H

// type énumeré pour indiquuer l'état de la mine : visible, marquée, ou cachée
enum STATE {
   VISIBLE, HIDDEN, MARKED
};

// 9 mines
enum {
   MINE = 9
};

// type struct d'une case
struct Case {
   STATE state;
   unsigned content; //9 == MINE
};

/**
* @brief montre l'élement au sein d'une case donnée
* @param[in] bloc la case à dévoiler
* @return char à montrer
*/
char showElement(Case &bloc);

/**
* @brief marque la mine pour une case donnée
* @param[inout] bloc
*/
void markMine(Case &bloc);
```

```
#endif //I_MINESWEEPER_CASE_H
```

case.cpp

```
#include "case.h"

// controle visibilite des mines sur la grille
char showElement(Case &bloc) {
   switch (bloc.state) {

       case VISIBLE: // mine visible
           if (bloc.content > 8) return 'm';
           if (bloc.content == 0) return ' ';
           return (char) (48 + bloc.content);
       case HIDDEN: // mine cachee
           return '.';
       case MARKED: // mine barree
           return 'x';
   }
}

/**
* @brief fonction marquant les mines
* @param bloc
*/

void markMine(Case &bloc) {
   bloc.state = STATE::MARKED;
}
```

problem.h

```
#ifndef I_MINESWEEPER_PROBLEM_H
#define I_MINESWEEPER_PROBLEM_H

struct Problem {
```

```
    unsigned lines, columns, mines;
    unsigned *minePos;
};

/**
* @brief retourne les coordonnées du problème
* @param[in] pb probleme donnée
* @param[in] pos position à traduire
* @param[out] x coord 'x'
* @param[out] y coord 'y'
*/
void getCoords(Problem &pb, unsigned pos, unsigned &x, unsigned &y);

/**
* @brief initialisation des colonnes, lignes et mines pour le problème
* @param[out] pb problème à sauvegarder
*/
void setProblem(Problem &pb);

/**
* @brief initialisation d'un problème sans mines
* @param[out] pb problème à sauvegarder
*/
void setBasicProblem(Problem &pb);

/**
* @brief génére des mines de manière aléatoire
* @param[in] pb probleme à sauvegarder
*/
void generateMines(Problem &pb);

/**
* @brief dévoile le probleme
* @param[in] pb probleme a montrer
*/
void printProblem(Problem &pb);

/**
```

```
* @brief suppression de l'allocation dynamique en mémoire
* @param pb probleme dont l'allocation soit etre effacée
*/
void deleteProblem(Problem &pb);

/**
* @brief création d'un probleme
*/
void createProblem();

#endif //I_MINESWEEPER_PROBLEM_H
```

problem.cpp

```
#include <iostream>
#include "problem.h"

void getCoords(Problem &pb, unsigned pos, unsigned &x, unsigned &y) {
    x = pos / pb.columns;
    y = pos % pb.columns;
}

void setProblem(Problem &pb) {
    unsigned lines, columns, mines;
    std::cin >> lines >> columns >> mines;
    pb.lines = lines;
    pb.columns = columns;
    pb.mines = mines;
    pb.minePos = new unsigned[pb.mines];
}

void setBasicProblem(Problem &pb) {
    unsigned lines, columns, mines;
    std::cin >> lines >> columns;
    pb.lines = lines;
    pb.columns = columns;
}
```

```
void generateMines(Problem &pb) {
    unsigned limit = pb.lines * pb.columns - 1;
    for (unsigned i = 0; i < pb.mines; i++) {
        unsigned minePos = rand() % limit;
        pb.minePos[i] = minePos;
    }
}

void printProblem(Problem &pb) {
    std::cout << pb.lines << " "
              << pb.columns << " "
              << pb.mines << " ";
    for (unsigned i = 0; i < pb.mines; i++) {
        std::cout << pb.minePos[i] << " ";
    }
    std::cout << '\n';
}

void deleteProblem(Problem &pb) {
    delete[] pb.minePos;
}

void createProblem() {
    Problem pb{};
    setProblem(pb);

    generateMines(pb);

    printProblem(pb);

    deleteProblem(pb);
}
```

grid.h

```
//
// Created by Imane & Ranim;
```

```
//

#ifndef I_MINESWEEPER_GRID_H
#define I_MINESWEEPER_GRID_H

#include "case.h"
#include "problem.h"

struct Grid {
   Problem pb;
   Case **ms; // MineSweeper
};

/**
* @brief Vérifie les coordonées x et y
* @param[in] grid la grille
* @param[in] x à vérifier
* @param[in] y à vérifier
* @return true si les coordonées sont valides, false sinon
*/

bool validCoords(Grid &grid, unsigned x, unsigned y);

/**
* @brief set mine from buffer
* @param[inout] grid
*/
void setMines(Grid &grid);

/**
* @brief génération du démineur
* @param[inout] grid la grille du démineur
*/
void generateMS(Grid &grid);

/**
* @brief place les mines dans le jeu du démineur
* @param[inout] gridla grille du jeu
```

```
*/
void placeMines(Grid &grid);

/**
* @brief ajoute +1 à la case adjacente à une mine
* @param[inout] grid grille du jeu
* @param[in] x coordonée x de la mine
* @param[in] y coordonée y de la mine
*/
void addMineAdj(Grid &grid, unsigned &x, unsigned &y);

/**
* @brief place les valeurs dans le démineur
* @param[inout] grid la grille du jeu
*/
void fillMS(Grid &grid);

/**
* @brief démasquer une case
* @param[inout] grid grille du jeu
* @param x coordonée x de la case
* @param y coordonée y de la case
*/

void unmaskMine(Grid &grid, unsigned x, unsigned y);

/**
* @brief execution d'un mouvement
* @param[in] grid grille du jeu
*/
void executeMoves(Grid &grid);

/**
* @brief montrer la grille
* @param[in] grid grille du jeu
*/
void showGrid(Grid &grid);
```

```
/**
* @brief annonce si le jeu est gagné
* @param[in] grid grille du jeu
* @return true si utilisateur gagnant, false sinon
*/
bool gameWon(Grid &grid);

/**
* @brief annonce si jeu perdu
* @param[in] grid grille du jeu
* @return true si jeu perdu, false sinon
*/
bool gameLost(Grid &grid);

/**
* @brief montre le résultat
* @param[in] grid grille du jeu
* @param cmd obtenir la commande d'execution
*/
void showResult(Grid &grid, unsigned cmd);

/**
* @brief retourne la prochaine action
* @param[in] grid grille
*/
void getActionToPlay(Grid &grid);

/**
* @brief supprime toute l'allocation dynamique en mémoire de la grille
* @param grid grille dont la mémoire doit etre supprimée
*/
void deleteGrid(Grid &grid);

/**
* @brief crée une grille
*/
void createGrid();
```

```
/**
* @brief retourne l'état du jeu
* @param cmd commande à executer
*/
void getResult(unsigned cmd);

/**
* @brief lectire de la grille
*/
void readGrid();

#endif //I_MINESWEEPER_GRID_H
```

grid.cpp

```
#include <iostream>
#include "grid.h"


bool validCoords(Grid &grid, unsigned x, unsigned y) {
   return x < grid.pb.lines && y < grid.pb.columns;
}


void setMines(Grid &grid) {
   for (unsigned i = 0; i < grid.pb.mines; i++) {
       unsigned mP;
       std::cin >> mP;
       grid.pb.minePos[i] = mP;
   }
}


void generateMS(Grid &grid) {
   grid.ms = new Case *[grid.pb.lines];
   for (unsigned x = 0; x < grid.pb.lines; x++) {
       grid.ms[x] = new Case[grid.pb.columns];
```

```
        for (unsigned y = 0; y < grid.pb.columns; y++) {
            grid.ms[x][y].content = 0;
            grid.ms[x][y].state = STATE::HIDDEN;
        }
    }
}

void placeMines(Grid &grid) {
    for (unsigned m = 0; m < grid.pb.mines; m++) {
        unsigned x, y;
        getCoords(grid.pb, grid.pb.minePos[m], x, y);
        grid.ms[x][y].content = MINE;
    }
}

void addMineAdj(Grid &grid, unsigned &x, unsigned &y) {
    enum {
        COUNT_TEST = 8
    };
    unsigned int coords[COUNT_TEST][2] =
            {
                    {x,     y + 1},
                    {x,     y - 1},
                    {x + 1, y},
                    {x + 1, y + 1},
                    {x + 1, y - 1},
                    {x - 1, y},
                    {x - 1, y + 1},
                    {x - 1, y - 1},
            };

    for (unsigned int *coord: coords) {
        if (validCoords(grid, coord[0], coord[1])) {
            if (grid.ms[coord[0]][coord[1]].content < 8)
grid.ms[coord[0]][coord[1]].content += 1;
        }
    }
}
```

```
void fillMS(Grid &grid) {
    for (unsigned x = 0; x < grid.pb.lines; x++) {
        for (unsigned y = 0; y < grid.pb.columns; y++) {
            if (grid.ms[x][y].content == MINE) {
                addMineAdj(grid, x, y);
            }
        }
    }
}

void unmaskMine(Grid &grid, unsigned x, unsigned y) {
    enum {
        COUNT_TEST = 8
    };
    unsigned int coords[COUNT_TEST][2] =
            {
                    {x + 0, y + 1},
                    {x + 0, y - 1},
                    {x + 1, y + 0},
                    {x + 1, y + 1},
                    {x + 1, y - 1},
                    {x - 1, y + 0},
                    {x - 1, y + 1},
                    {x - 1, y - 1},
            };

    if (!validCoords(grid, x, y)) return;
    Case &c = grid.ms[x][y];

    if (c.state == STATE::VISIBLE) return;
    c.state = STATE::VISIBLE;

    if (c.content == MINE) {
        for (unsigned m = 0; grid.pb.mines > m; m++) {
            unsigned mx, my;
            getCoords(grid.pb, grid.pb.minePos[m], mx, my);
            grid.ms[mx][my].state = STATE::VISIBLE;
```

```
        }
        return;
    }

    if (c.content != 0) return;

    for (unsigned t = 0; t < COUNT_TEST; t++) {
        unmaskMine(grid, coords[t][0], coords[t][1]);
    }
}

void executeMoves(Grid &grid) {
    unsigned moveCount;
    std::cin >> moveCount;

    for (unsigned m = 0; m < moveCount; m++) {
        char action;
        unsigned position, x, y;
        std::cin >> action >> position;
        getCoords(grid.pb, position, x, y);
        if (action == 'D') {
            unmaskMine(grid, x, y);
        }

        if (action == 'M') {
            markMine(grid.ms[x][y]);
            if(grid.ms[x][y].content == MINE) {
                unsigned xm, ym;
                getCoords(grid.pb, grid.pb.minePos[0], xm, ym);
                unmaskMine(grid, xm, ym);
            }
        }
    }
}

bool gameWon(Grid &grid) {
    for (unsigned int x = 0; x < grid.pb.lines; x++) {
        for (unsigned y = 0; y < grid.pb.columns; y++) {
```

```
            Case &c = grid.ms[x][y];
            if (c.content == MINE && c.state != STATE::MARKED) return false;
        }
    }
    return true;
}

bool gameLost(Grid &grid) {
    for (unsigned int x = 0; x < grid.pb.lines; x++) {
        for (unsigned y = 0; y < grid.pb.columns; y++) {
            Case &c = grid.ms[x][y];

            if ((c.content == MINE && c.state == STATE::VISIBLE) ||
                (c.content != MINE && c.state == STATE::MARKED))
                return true;
        }
    }
    return false;
}

void showResult(Grid &grid, unsigned cmd) {
    if (cmd == 4) std::cout << "game " << (gameLost(grid) ? "lost" : "not
lost");
    else std::cout << "game " << (gameWon(grid) ? "won" : "not won");
}

void getActionToPlay(Grid &grid) {
    unsigned underscoreCount = grid.pb.columns * 3,
            lineCount = (grid.pb.columns * 4) + 1;

    for (unsigned x = 0; x < grid.pb.lines; x++) {
        char useless, line[lineCount + 1];
        for (unsigned _ = 0; _ < underscoreCount; _++) std::cin >> useless;
        std::cin.read(line, sizeof(line));
        //If we start with 2 and get all 4 values, it's ok !
        for (unsigned c = 3; c < lineCount + 1; c += 4 /*c++ :p*/) {
            unsigned getPos = grid.pb.columns * x + (c / 4);
            if(line[c] == '.') {
```

```
                std::cout << 'D' << getPos;
                return;
            }
        }
    }
}

void showGrid(Grid &grid) {

    std::cout << grid.pb.lines << " " << grid.pb.columns << '\n';
    for (unsigned int x = 0; x < grid.pb.lines; x++) {
        for (unsigned line = 0; line < grid.pb.columns; line++) {
            std::cout << " ___";
        }
        std::cout << "\n";
        for (unsigned int y = 0; y < grid.pb.columns; y++) {

            if (y == 0) std::cout << "| ";
            else std::cout << ' ';

            std::cout << showElement(grid.ms[x][y])
                      << " |";
        }
        std::cout << '\n';
    }
    for (unsigned line = 0; line < grid.pb.columns; line++) {
        std::cout << " ___";
    }
    std::cout << '\n';
}

void deleteGrid(Grid &grid) {
    delete[] grid.pb.minePos;
    for (unsigned i = 0; i < grid.pb.lines; i++) {
        delete[] grid.ms[i];
    }
    delete[] grid.ms;
}
```

```
void createGrid() {
   Grid grid{};
   setProblem(grid.pb);

   setMines(grid);

   generateMS(grid);

   placeMines(grid);

   fillMS(grid);

   executeMoves(grid);

   showGrid(grid);

   deleteGrid(grid);
}

void getResult(unsigned cmd) {
   Grid grid{};
   setProblem(grid.pb);

   setMines(grid);

   generateMS(grid);

   placeMines(grid);

   fillMS(grid);

   executeMoves(grid);

   showResult(grid, cmd);

   deleteGrid(grid);
}
```

```
void readGrid() {
   Grid grid{};
   setBasicProblem(grid.pb);

   getActionToPlay(grid);
}
```

main.cpp

```
#include <iostream>
#include <random>
#include "grid.h"

int main() {
   srand((unsigned) time(nullptr));

   unsigned command;
   std::cin >> command;
   switch (command) {
       // création du problème
       case 1:
           createProblem();
           break;
       // création de la grille
       case 2:
           createGrid();
           break;
       case 3:
       // retourne le résultat pour une commande donnée
       case 4:
           getResult(command);
           break;
       // lecture de la grille
       case 5:
           readGrid();
           break;
       // dans le cas ou commande non reconnue
```

```
        default:
            std::cout << "Commande inconnue...";
            break;
    }
}
```