DOSSIER DE DÉVELOPPEMENT EN LANGAGE C++

Objet : Minesweeper

08 JANVIER 2022

GROUPE 107, 111
Anxian Zhang, Vick Ye

Table des matières

I. Présentation de l'application

❖ Introduction du sujet

Le premier projet consistait à développer un interpréteur de commande pour l'organisation d'un tournoi. Cette fois-ci nous avons dû en programmer un de démineur, celui-ci a été décomposé en 5 parties (en 5 commandes différentes). Les 5 commandes réunies nous permettent de simuler un jeu de démineur de bout en bout. De plus, ce démineur a une règle particulière : marquer une case vide induit systématiquement à la perte de la partie.

❖ Précision des commandes

Commande **1** : Génération d'un problème

En donnant (en entrée standard) le nombre de lignes, de colonnes et de bombes, notre programme devra générer le nombre de bombes demandé en entrée avec leurs positions. Ces positions devront se trouver dans le tableau à deux dimensions qui devra être généré antérieurement grâce au nombre de lignes et de colonnes fournies en entrée.

Commande **2** : Création d'une grille

Une grille sera affichée avec l'aide d'un problème créé notamment grâce à la commande 1 et d'un historique de coups. Donc cette commande nous donnera un visuel sur la partie en cours.

Commande **3** : Déterminer si la partie est gagnée

La détermination d'une « partie gagnée ou non » s'effectuera avec la même entrée que celle de la commande 2. Soit, avec les dimensions du tableau, le nombre de bombes suivi de leurs positions, le nombre de coups suivi des positions des coups.

Commande **4** : Détermine si la partie est perdue

Même principe que la commande 3, mais, cette fois-ci, pour déterminer si une partie est perdue ou non.

Commande **5** : Production d'un nouveau coup

Afin de produire un nouveau coup nous devons entrer une grille (que l'on pourra notamment générer à l'aide de la commande 2). Suite à cela, le programme devra fournir en sortie standard un coup, qu'il soit marqué ou démasqué. Ce coup sera un coup dit « juste », c'est-à-dire un coup ou la case ne sera pas déjà marquée ou démasquée par le joueur. (Davantage de précisions seront apportées dans « jeux d'essai » de la rubrique « Annexes » à la page 6).

II. Bilan de projet

❖ Les difficultés rencontrées

Durant le cycle de développement, le plus dur a été la compréhension du sujet. Mais nous avons pu contacter l'enseignant afin qu'il puisse nous éclairer, nous donner des pistes de compréhension.

❖ Le développement

Lors du développement des commandes 2 et 5, nous avons rencontré quelques problèmes. En effet, au sein de la commande 2, nous avons eu besoin d'une fonction de récurrence pour démasquer les cases adjacentes à celle de départ (si elle est vide). L'opération devait se répéter jusqu'à ce que la zone vide soit délimitée par des chiffres. La partie la plus fastidieuse a été de trouver les bonnes conditions afin d'éviter de se retrouver dans une boucle infinie.

Dans la commande 5, le plus compliqué a été de traiter les enchaînements des caractères ' | ', effectivement si plusieurs « barre » s'enchaine (par exemple ' | | | '), cela veut dire qu'il y aura une case vide entre la barre n°1, n°2 et n°3. Ainsi, celle-ci sera à l'état démasqué. Il fallait alors que nous développions un programme, qui puisse prendre en compte le nombre de « barres » entrevu dans le fichier d'entrée donné. Et en fonction de ce nombre, nous devions démasquer la ou les cases qui se trouvent sur notre grille numérique. Cette grille devra être créée au préalable avec le nombre de lignes et de colonnes lu dans le fichier d'entrée avant celle de grille. Finalement, à la fin du programme, nous devions avoir la même grille fournit en entrée, mais cette fois, stockée sous forme numérique.

<u>Les erreurs d'exécutions :</u>

Nous avons rencontré 2 erreurs majeures, une première qui est revenue plusieurs fois lors de la programmation, et une deuxième qui nous a donné du fil à retordre.

- L'Access violation Wittring location

Tout au long du développement, et ce dans la majorité des fonctions, nous avions dû utiliser une double boucle pour le parcours du tableau à deux dimensions. À certain moment d'inattention, nous avions incrémenté la même variable (++i) dans les deux boucles for. De plus, à chaque fois nous mettions du temps à comprendre pourquoi est-ce qu'il y a une telle erreur qui se produit.

- Le Stack overflow

Cette erreur est provoquée par un dépassement de la pile d'exécution. La fonction de récurrence de notre programme (utilisée pour le démasquage des cases) ne contenait pas suffisamment de condition pour en ressortir. La résolution du problème a pris pas moins de 4h.

❖ Les réussites

Dans l'ensemble, nous sommes à peu près passés par toutes les étapes du développement de ce programme, que ce soit sur l'organisation, le développement ou la validation de l'application.

L'organisation :

Avant de commencer à programmer, nous nous sommes consultés à plusieurs reprises afin de discuter du problème fourni. Nous avons d'abord émis des hypothèses sur la conception en générale soit, commencé à construire les structures de donnée potentiel pour le démineur mais aussi le « squelette » du programme principal. Puis nous avons prêté attention à leurs conceptions, cette fois-ci un peu plus dans les détails en donnant des bouts de code « potentiel ».

Le développement :

Puis vient alors la réalisation, chacun d'entre nous était chargé d'une commande, si l'une des deux personnes se trouvait dans une impasse on se consultait afin d'y remédier. À chaque fin de commande, nous faisons une réunion. Elle a pour but d'expliquer à l'autre ce que nous avons fait pour telle commande ou les modifications que nous avons apportées au programme.

La validation :

Durant tout le cycle de développement, à chaque fois que nous finissions une fonction, on le testait tout de suite, pour s'assurer de son bon fonctionnement. Les tests effectués étaient majoritairement des ajouts de cout dans les fonctions. Cela nous permettait de visualiser ce qui était stocké dans une ou plusieurs variable(s). Si le(s) résultat(s) est/sont cohérent(s) nous validions la fonction puis passions au codage de la suivante. Quand nous finissions une commande nous la testions avec tous les cas qui nous viennent tout en restant raisonnable (entré des fichiers « in » cohérentes (voir page 6 pour les jeux d'essai)).

Finalement, nous avons réussi à programmer les 4 premières commandes qui nous permettent de jouer au démineur de A à Z, ainsi que la dernière qui, grâce aux commandes antérieures, nous permet de simuler une évolution de partie aléatoire.

❖ Les améliorations

Les variables intermédiaires :

Il se trouve que lors du développement de la commande 2 nous avons montré un extrait de code à un enseignant. Celui-ci nous a fait remarquer que nous avions systématiquement employé des variables intermédiaires. De notre point de vue, on pensait que cela aidait à la compréhension du code, mais cela produit apparemment l'effet inverse. Forts de cette remarque, nous l'avons tout de suite corrigé. À la suite du développement, nous avons réitéré cette même erreur, ce point sera donc à améliorer.

La compréhension du sujet :

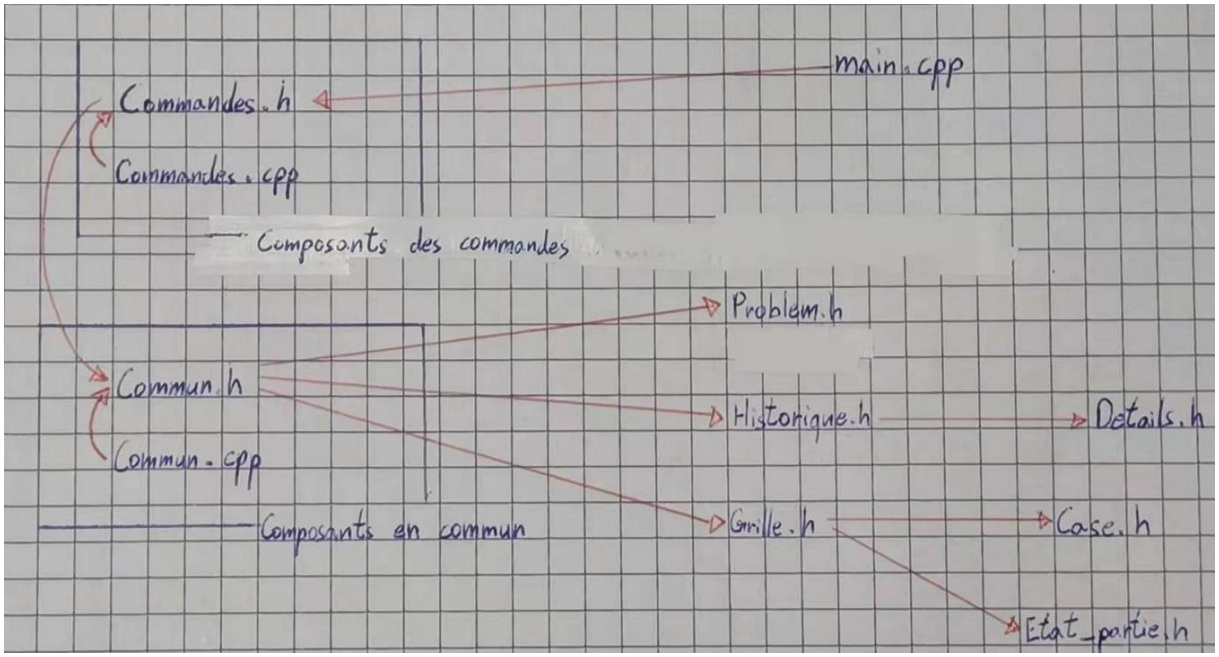
Tout au long du cycle, nous avons fréquemment posé des questions à l'enseignant chargé du projet. Après qu'il nous ait répondues et que nous revoyions le sujet, nous avons pu constater que les réponses apportées étaient déjà présentes dans le sujet.

La structuration des fichiers :

Pour ce projet, nous avons dû créer une arborescence (voir III) pour structurer l'application. Le problème que nous avions alors rencontré était la quantité de lignes de code qui diverge beaucoup. En effet, dans la majorité des fichiers les lignes de code varient entre 10 et 100. Pourtant, dans le fichier Grille.cpp il y a environ 600 lignes. De notre point de vue, cela reste assez cohérent car le projet est centré sur une grille. Donc, la majorité des fonctions qui y seront développées l'utiliseront. Cependant, nous avons pensé qu'il existe sûrement une solution pour améliorer la structure du code, mais pas laquelle.

III. Les annexes

- **Graphe de dépendance des fichiers sources de notre application**



❖ Les jeux d'essai (fichier « in », « out » utilisé)

Les jeux de testes dont nous vous avons parlé un peu plus tôt se divise en 5 parties, chaque partie correspond à une commande.

Aide à la lecture des jeux d'essai :

Les jeux d'entrées se trouvent à gauche et celui des sorties à droite pour toutes les commandes. Les nombres en rouge correspondent respectivement au nombre de bombes suivi des positions des bombes générées. En bleu le nombre de coups jouer avec la position des coups joué, la lettre D pour démasquer et M pour marquer.

Commande **1** :

```
1 10 10 30
```

```
10 10 30 2 3 4 10 13 15 16 20 23 24 27 29 30 32
34 38 39 40 41 42 44 58 63 65 66 70 73 76 92 99
```

Commande **2** :

```
2 10 10 30 2 3 4 10 13 15
16 20 23 24 27 29 30 32 34
38 39 40 41 42 44 58 63 65
66 70 73 76 92 99 8 D0 D79
M3 D94 D9 D46 D90 D52
```

```
10 10
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
| 1 | . | . | x | . | . | . | 1 |   |   |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
| . | . | . | . | . | . | . | 2 | 2 | 1 |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
| . | . | . | . | . | . | . | . | . | . |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
| . | . | . | . | . | 3 | 1 | 2 | . | . |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
| . | . | . | . | . | 2 |   | 2 | . | . |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
| . | . | 3 | . | . | 3 | 2 | 2 | . | . |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
| . | . | . | . | . | . | . | 3 | 1 | 1 |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
| . | . | . | . | . | . | . | 2 |   |   |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
| 1 | 2 | . | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
|   | 1 | . | 1 |   |   |   |   | 1 | . |
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
```

Commande **3** : (Nous avons pris des valeurs plus petites pour donner un maximum de cas possible)

- Toutes les cases sont démasquées

3 4 6 3 13 15 19 7 M13 D0 D12 D14 D18 D20 D21

game won

- Toutes les bombes sont marquées

3 4 6 3 13 15 19 4 D8 M13 M15 M19

game won

- Démasquage d'une bombe (sur le dernier coup joué)

3 4 6 3 13 15 19 4 D8 M13 M15 D19

game not won

- Marquage d'une case vide (sur le dernier coup joué)

3 4 6 3 13 15 19 4 D8 M13 M15 M1

game not won

Commande **4** : (Nous sommes partis du même principe que la commande 3)

- Toutes les cases sont démasquées

4 4 6 3 13 15 19 7 M13 D0 D12 D14 D18 D20 D21

game not lost

- Toutes les bombes sont marquées

4 4 6 3 13 15 19 4 D8 M13 M15 M19

game not lost

- Démasquage d'une bombe (sur le dernier coup joué)

4 4 6 3 13 15 19 4 D8 M13 M15 D19

game lost

- Marquage d'une case vide (sur le dernier coup joué)

4 4 6 3 13 15 19 4 D8 M13 M15 M1

game lost

Commande **5** : (nous avons repris le tableau de la commande 2 pour avoir un panel de choix plus large)

5 10 10

1	.	.	x	.	.	.	1		
.	.	.	.	.	.	.	2	2	1
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	3	1	2	.	.
.	.	.	.	.	2		2	.	.
.	.	3	.	.	3	2	2	.	.
.	.	.	.	.	.	.	3	1	1
.	.	.	.	.	.	.	2		
1	2	.	2	1	1	1	1	1	1
	1	.	1					1	.

Les out possible sont toutes les cases avec un '.'
Voici quelques exemples :

D33

M48

M58

D71

❖ **Code complet de nos sources**

```
/**
 * @file Detail.h
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 1 05/01/2022
 * Composant des détails des coups
 */

#ifndef _DETAILS_
#define _DETAILS_

 /**
  * @brief Type structuré qui montre les détails
  *        d'un coup
  *
  */
struct Details_coups {
    char lettre;
    unsigned int position;
};

#endif // !_DETAILS_
```

```
/**
 * @file Case.h
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 1 05/01/2022
 * Composant de(s) case(s
 */

#ifndef _CASE_
#define _CASE_

 /**
  * @brief Type enumeré pour connaitre l'état d'une case
  *
  */
enum Etat_case { DEMASQUER, MARQUER, CACHER };

/**
 * @brief Type structuré contient les informations nessessaire
 *        pour une case donnée
 *
 */
struct Case {
    unsigned int contenue;
    Etat_case etat;
};

#endif // !_CASE_
```

```
/**
 * @file Etat_partie.h
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 1 05/01/2022
 * Composant le l'état de partie
 */

#ifndef _ETAT_PARTIE_
#define _ETAT_PARTIE_

 /**
 * @brief Type énuméré pour connaitre l'état de la partie
 *
 */
enum Etat_Partie { GAME_WON, GAME_NOT_WON, GAME_LOST, GAME_NOT_LOST };

#endif // !_ETAT_PARTIE_



/**
 * @file Grille.h
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 1 05/01/2022
 * Composant de la grille
 */

#ifndef _GRILLE_
#define _GRILLE_

#include "Case.h"
#include "Etat_partie.h"

 /**
  * @brief Type structuré de la grille de jeu
  *
  */
struct Grille {
    Case** grille_jeu;
    unsigned int lignes;
    unsigned int colonnes;
    Etat_Partie etatj; // état du jeu
};

#endif // !_GRILLE_



/**
 * @file Historique.h
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 1 05/01/2022
 * Composant de l'historique
 */

#ifndef _HISTORIQUE_
#define _HISTORIQUE_

#include "Details.h"
```

```
 /**
  * @brief Type structuré pour l'historique
  *        des coups joués
  *
  */
struct Historique_coup {
    unsigned int nb_coup;
    Details_coups* liste_coup;
};

#endif // !_HISTORIQUE_



/**
 * @file Problem.h
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 3 05/01/2022
 * Composant du problème
 */

#ifndef _PROBLEM_
#define _PROBLEM_

 /**
  * @brief Type structuré qui stock toutes les données relatives à
  *        la géneration d'un problème
  *
  */
struct Problem {
    unsigned int lignes;
    unsigned int colonnes;
    unsigned int bombes;
    unsigned int* pos_bombe; // position des bombes
};

#endif // !_PROBLEM_

/**
 * @file Commun.h
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 1 05/01/2022
 * Ensemble des fonctions appelées indirectements par le main.cpp
 */

#ifndef _COMMUN_
#define _COMMUN_

#include "Problem.h"
#include "Historique.h"
#include "Grille.h"

/**
 * @brief Alloue un tableau en fonction du nombre
 *        de coup donné en entré
 *
 * @see desallocation_coup: Desalloue le tableau
 *
 * @param [out] hc: l'historique de coups
 */
void allocation_coup(Historique_coup & hc);
```

```
/**
 * @brief Alloue dynamiquement un tableau pour le nombre
 *        de bombes donné
 *
 * @see desallocation_bombe: désalloue le tableau de bombe
 *
 * @param [out] p: le problème
 */
void allocation_bombe(Problem& p);

/**
 * @brief Alloue dynamiquement un tableau à deux
 *        avec les dimensions donnés
 *
 * @see allocation_grille: désalloue le tableau à
 *                         2 dimensions
 *
 * @param [out] g: la grille
 */
void allocation_grille(Grille& g);

/**
 * @brief Initialise toutes les cases du tableau
 *        l'état CACHER et son contenue à 0
 *
 * @param [out] g: la grille
 * @param [in] p: le problème
 */
void initialisation_grille(Grille& g, const Problem& p);

/**
 * @brief Initalise les positions des bombes donnée en entré
 *        dans la grille de jeu (sont toujours à l'état CACHER)
 *
 * @param [out] g: le grille
 * @param [in] p: le problème
 */
void initialisation_bombe(Grille& g, const Problem& p);

/**
 * @brief Place le nombre de bombe indiquée en entré
 *        au début du tableau
 *
 * @see rand_bombes: mélange le tableau pour les placer
 *                   alléatoirement
 *
 * @param [out] g: la grille de jeu
 * @param [in] p: le problème
 * @pre p.bombes <= p.lignes * p.colonnes
 */
void place_bombes_debut(Grille& g, const Problem& p);
```

```
/**
 * @brief Mélange les cases du tableau pour placer
 *        alléatoirement les bombes qui sont initialement au
 *        debut de ce-dernier
 *
 * @see get_pos_bombes: cherche les positions des bombes
 *
 * @param [out] g: la grille de jeu
 * @param [in] p: le problème
 */
void rand_bombes(Grille& g, const Problem& p);

/**
 * @brief Cherche toute les bombes qui se trouvent
 *        dans la grille pour ensuite stocker
 *        leurs positions dans le tableau de bombes
 *        crée dynamiquement
 *
 * @param [in] g: la grille de jeu
 * @param [in, out] p: le problème
 */
void get_pos_bombes(const Grille& g, Problem& p);

/**
 * @brief Indique le nombre de bombe dans les cases adjacentes
 *
 * @param [out] g: la grille de jeu
 * @param [in] p: le problème
 */
void give_value_adjacent(Grille& g, const Problem& p);

/**
 * @brief Affecte les bombes à leurs position respective
 *        et met toutes les cases contenant des bombes
 *        à l'etat DEMASQUER si le dernier coup
 *        joué met fin à la partie
 *
 * @param [out] g: la grille
 * @param [in] p: le problème
 */
void attribution_bombe_perdu(Grille& g, const Problem& p);

/**
 * @brief Verifie si la position du coup joué
 *        corespond à celle d'une bombes
 *
 * @param [in] p: le problème
 * @param [in] pos_coup: la position du coup
 * @return int 1 si c'est vrais, 0 si c'est faux
 */
int verification(const Problem& p, const unsigned int pos_coup);
```

```
/**
 * @brief Marque une case donnée que si elle ne pas
 *        corespond à celle d'une bombe
 *
 * @see attribution_bombe_perdu: met toute les bombes à l'état DEMASQUER
 *                               dans le cas contraire
 *
 * @param [out] g: la grille
 * @param [in] p: le problème
 * @param [in] posc: position du coup joué
 * @param [in] lc: ligne du coup joué
 * @param [in] cc: colonne du coup joué
 */
void marquer(Grille& g, const Problem& p, const unsigned int posc,
       const unsigned int lc, const unsigned int cc);

/**
* @brief Met qu'une case à l'état DEMASQUER si la case à démasquer contient
*        un chiffre mais si la case de départ est vide et que ces cases ajdacentes
*        sont elles aussi vide la fonction démasquera ces case,
*        l'opération se répète jusqu'à que la zone soit entouré par des chiffres
*
* @param [out] g: la grille
* @param [in] p: le problème
* @param [in] pos: la postion du coup
*/
void recurrence(Grille& g, const Problem& p, const unsigned int pos);

/**
 * @brief Démasque une ou plusieurs cases
 *               si celle-ci n'est pas une bombe
 *
 * @see attribution_bombe_perdu: met toute les bombes à l'état DEMASQUER
 *                               dans le cas contraire
 * @see recurrence: se charge du démasquage si le coup joué n'est pas à la
 *                  position d'un bombe
 *
 * @param [out] g: la grille
 * @param [in] p: le problème
 * @param [in] posc: position du coup joué
 * @param [in] lc: ligne du coup joué
 * @param [in] cc: colonne du coup joué
 */
void demasquer(Grille& g, const Problem& p, const unsigned int posc,
       const unsigned int lc, const unsigned int cc);

/**
 * @brief Traite les coups joué pour savoir s'il faut
 *        marquer ou démasqer
 *
 * @see demasquer: démasque les cases indiquées
 * @see marquer: marque les cases indiquées
 *
 * @param [in] g: la grille de jeu
 * @param [in] p: le problème
 * @param [in] hc: l'historique de coup
 */
void reconnaissance_coup(Grille& g, const Problem& p, const Historique_coup hc);
```

```
/**
 * @brief Convertit un chiffre de type char en unsigned int
 *        grace au calcul storage - '0' avec 0 = 48
 *
 * @param [in] storage: le chiffre à convertir
 * @return unsigned int: la valeur convertit
 * @pre storage >= '1' && storage <= '8'
 */
unsigned int val_adja(char storage);

/**
 * @brief Analyse la grille en entré et le stock
 *        les données en fonction des valeurs lu
 *
 * @see val_adja: convertit un chiffre ASCCI en entier naturel
 *
 * @param [in,out] g: la grille
 */
void traitement_grille(Grille& g);

/**
 * @brief Compte le nombre de case qui sont
 *        à l'état CACHER
 *
 * @param [in] g: la grille
 * @return unsigned int: le totale de case CACHER
 */
unsigned int count_hidden(const Grille& g);

/**
 * @brief Choisie un coups parmis les cases
 *        à l'état CACHER
 *
 * @see count_hidden: compte le nombre totale de case caché
 *
 * @param [in] g: la grille
 * @param [out] nw: la nouvelle position
 * @param [in] nb_cacher: nombre de case à l'état CACHER
 */
void choice_move(const Grille& g, Details_coups& nw, const unsigned int nb_cacher);

#endif



/**
 * @file Commun.cpp
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 1 05/01/2022
 * Ensemble des fonctions appelées indirectements par le main.cpp
 */

#include <iostream>
#include <cassert>
using namespace std;

#include "Commun.h"

void allocation_coup(Historique_coup& hc)
{
    hc.liste_coup = new Details_coups[hc.nb_coup];
}
```

```
void allocation_bombe(Problem& p)
{
    p.pos_bombe = new unsigned int[p.bombes];
}

void allocation_grille(Grille& g)
{
    g.grille_jeu = new Case * [g.lignes];
    for (unsigned int i = 0; i < g.lignes; ++i)
    {
        g.grille_jeu[i] = new Case[g.colonnes];
    }
}


void initialisation_grille(Grille& g, const Problem& p)
{
    for (unsigned int i = 0; i < p.lignes; ++i)
    {
        for (unsigned int j = 0; j < p.colonnes; ++j)
        {
            g.grille_jeu[i][j].contenue = 0;
            g.grille_jeu[i][j].etat = CACHER;
        }
    }
}


void initialisation_bombe(Grille& g, const Problem& p)
{
    unsigned int pos_bombe;
    unsigned int ligne, colonne;

    for (unsigned int i = 0; i < p.bombes; ++i)
    {
        pos_bombe = p.pos_bombe[i];
        ligne = pos_bombe / p.colonnes;
        colonne = pos_bombe % p.colonnes;
        g.grille_jeu[ligne][colonne].contenue = 9;
    }
}


void place_bombes_debut(Grille& g, const Problem& p)
{
    assert(p.bombes <= p.lignes * p.colonnes);

    /* indiqué le nombre de bombe qu'on à deja placé
    sert aussi d'indice */
    int indice_bombe = 0;

    // sort de la fonction si le nombre de bombe est null
    if (p.bombes == 0)
    {
        return;
    }
    else
    {
        for (unsigned int j = 0; j < p.lignes; ++j) // ligne
        {
```

```
            for (unsigned int k = 0; k < p.colonnes; ++k)// colonne
            {
                g.grille_jeu[j][k].contenue = 9; // '9' indique une bombe
                ++indice_bombe;
                if (indice_bombe == p.bombes)
                {
                    rand_bombes(g, p);
                    return; // sort quand toute les bombes sont placé
                }
            }
        }
    }
}


void rand_bombes(Grille& g, const Problem& p)
{
    Case temps;
    unsigned int randl, randc; // respectivement ligne et colonne

    for (unsigned int i = 0; i < p.lignes; ++i)
    {
        for (unsigned int j = 0; j < p.colonnes; ++j)
        {
            randl = rand() % p.lignes;
            randc = rand() % p.colonnes;

            temps = g.grille_jeu[i][j];
            g.grille_jeu[i][j] = g.grille_jeu[randl][randc];
            g.grille_jeu[randl][randc] = temps;
        }
    }
}

void get_pos_bombes(const Grille& g, Problem& p)
{
    unsigned int valeur;
    unsigned int position;
    int indice_bombe = 0;

    for (unsigned int i = 0; i < p.lignes; ++i)
    {
        for (unsigned int j = 0; j < p.colonnes; ++j)
        {
            valeur = g.grille_jeu[i][j].contenue;
            if (valeur == 9) // si valeur = mine
            {
                // calcule de la position
                position = (i * p.colonnes) + j;
                p.pos_bombe[indice_bombe] = position;
                ++indice_bombe;
            }
        }
    }
}
```

```
void give_value_adjacent(Grille& g, const Problem& p)
{
    unsigned int pos_bombe;
    unsigned int ligne, colonne;

    for (unsigned int i = 0; i < p.bombes; ++i)
    {
        pos_bombe = p.pos_bombe[i];
        ligne = pos_bombe / p.colonnes;
        colonne = pos_bombe % p.colonnes;

        for (int j = -1; j < 2; ++j)
        {
            for (int k = -1; k < 2; ++k)
            {
                if ((ligne + j) < p.lignes && (ligne + j) >= 0 &&
                    ((colonne + k) < p.colonnes && (colonne + k) >= 0))
                {
                    unsigned int contient =
                     g.grille_jeu[ligne + j][colonne + k].contenue;
                    if (contient >= 0 && contient <= 7)
                    {
                        ++g.grille_jeu[ligne + j][colonne + k].contenue;
                    }
                }
            }
        }
    }
}


void attribution_bombe_perdu(Grille& g, const Problem& p)
{
    unsigned int pos_bombe;
    unsigned int ligne, colonne;

    for (unsigned int i = 0; i < p.bombes; ++i)
    {
        pos_bombe = p.pos_bombe[i];
        ligne = pos_bombe / p.colonnes;
        colonne = pos_bombe % p.colonnes;

        g.grille_jeu[ligne][colonne].etat = DEMASQUER;
    }
}

int verification(const Problem& p, const unsigned int pos_coup)
{
    unsigned int pos_bombe;

    for (unsigned int i = 0; i < p.bombes; ++i)
    {
        pos_bombe = p.pos_bombe[i];

        if (pos_bombe == pos_coup)
        {
            return 1;
        }
    }
    return 0;
}
```

```
void marquer(Grille& g, const Problem& p, const unsigned int posc,
    const unsigned int lc, const unsigned int cc)
{
    int etat;
    etat = verification(p, posc);

    if (etat == 1) // coup juste
    {
        g.grille_jeu[lc][cc].etat = MARQUER;
    }
    else // mauvais coup
    {
        g.grille_jeu[lc][cc].etat = MARQUER;
        attribution_bombe_perdu(g, p);
    }
}


void recurrence(Grille& g, const Problem& p, const unsigned int pos)
{
    unsigned int position_case = pos;
    unsigned int ligne = position_case / p.colonnes;
    unsigned int colonne = position_case % p.colonnes;

    if (g.grille_jeu[ligne][colonne].etat == DEMASQUER)
    {
        return;
    }

    g.grille_jeu[ligne][colonne].etat = DEMASQUER;

    if (g.grille_jeu[ligne][colonne].contenue >= 1
        && g.grille_jeu[ligne][colonne].contenue <= 8)
    {
        return;
    }

    for (int j = -1; j < 2; ++j)
    {
        for (int k = -1; k < 2; ++k)
        {
            if (ligne + j >= 0 && ligne + j < p.lignes && colonne + k >= 0
                && colonne + k < p.colonnes)
            {
                if (g.grille_jeu[ligne + j][colonne + k].etat == CACHER)
                {
                    // nouvelle position à traité
                    position_case = ((ligne + j) * p.colonnes) + (colonne + k);
                    recurrence(g, p, position_case);
                }
            }
        }
    }
}
```

```
void demasquer(Grille& g, const Problem& p, const unsigned int posc,
    unsigned int lc, unsigned int cc)
{
    int etat = verification(p, posc);

    if (etat == 1) // mauvais coup
    {
        attribution_bombe_perdu(g, p);
    }
    else // coups juste
    {
        recurrence(g, p, posc);
    }
}

void reconnaissance_coup(Grille& g, const Problem& p, const Historique_coup hc)
{
    unsigned int position;
    unsigned int ligne, colonne;
    char lettre;

    for (unsigned int i = 0; i < hc.nb_coup; ++i)
    {
        lettre = hc.liste_coup[i].lettre;
        position = hc.liste_coup[i].position;
        ligne = position / p.colonnes;
        colonne = position % p.colonnes;

        if (lettre == 'D')
        {
            demasquer(g, p, position, ligne, colonne);
        }
        if (lettre == 'M')
        {
            marquer(g, p, position, ligne, colonne);
        }
    }
}

unsigned int val_adja(char storage)
{
    assert(storage >= '1' && storage <= '8');
    return storage - '0';
}


void traitement_grille(Grille& g)
{
    unsigned int num_ligne = 0; // n°ligne dans les entrés lu
    unsigned int num_colonne = 0; // n°colonne dans les entrés lu
    int i = 0, j = 0; // nombre de '|' lu
    char storage;

    do
    {
        cin >> storage;
        if (storage != '_') // prend tout les caractère sauf '_'
        {
            if (storage == '|')
            {
                ++i; // incrémentation du nombre de '|'
```

```
                if (i == 2)
                {
                    g.grille_jeu[num_ligne][num_colonne].contenue = 0;
                    g.grille_jeu[num_ligne][num_colonne].etat = DEMASQUER;
                    ++num_colonne;

                    i = 1;
                    ++j;
                    if (num_colonne == g.colonnes)
                    {
                        i = 0;
                        j = 0;
                    }
                }
            }
            else if (storage >= '1' && storage <= '8')
            {
                g.grille_jeu[num_ligne][num_colonne].contenue = val_adja(storage);
                g.grille_jeu[num_ligne][num_colonne].etat = DEMASQUER;
                ++num_colonne;
                //initialise à 0 à chaque fois qu'on a une valeur != de '|'
                i = 0;
                j = 0;
            }
            else if (storage == 'm')
            {
                g.grille_jeu[num_ligne][num_colonne].contenue = 9;
                g.grille_jeu[num_ligne][num_colonne].etat = DEMASQUER;
                ++num_colonne;
                i = 0;
                j = 0;
            }
            else if (storage == 'x')
            {
                g.grille_jeu[num_ligne][num_colonne].contenue = 0;
                g.grille_jeu[num_ligne][num_colonne].etat = MARQUER;
                ++num_colonne;
                i = 0;
                j = 0;
            }
            else
            {
                g.grille_jeu[num_ligne][num_colonne].contenue = 0;
                g.grille_jeu[num_ligne][num_colonne].etat = CACHER;
                ++num_colonne;
                i = 0;
                j = 0;
            }
        }

        if (num_colonne == g.colonnes)
        {
            num_colonne = 0;
            ++num_ligne;
            i = -1;
        }

    } while (num_ligne != g.lignes);
}
```

```
unsigned int count_hidden(const Grille& g)
{
    unsigned int nb_cacher = 0; //nombre de case caché
    for (unsigned int i = 0; i < g.lignes; ++i)
    {
        for (unsigned int j = 0; j < g.colonnes; ++j)
        {
            if (g.grille_jeu[i][j].etat == CACHER)
            {
                ++nb_cacher;
            }
        }
    }
    return nb_cacher;
}

void choice_move(const Grille& g, Details_coups& nw, const unsigned int nb_cacher)
{
    // choisit un numéros parmis tout les cases cachés sauf 0
    unsigned int num_case = rand() % (nb_cacher - 1) + 1;

    int k = 0; // indice qui va être incrémenté jusqu'à que k = num_case

    for (unsigned int i = 0; i < g.lignes; ++i)
    {
        for (unsigned int j = 0; j < g.colonnes; ++j)
        {
            if (g.grille_jeu[i][j].etat == CACHER)
            {
                ++k;

                /*prendre un valeur dans l'interval [1;10]
                qui va déterminer si le coup sera M ou D*/
                unsigned int number = rand() % (10 - 1) + 1;
                if (k == num_case)
                {
                    nw.position = (i * g.colonnes) + j;
                    if (number <= 5) // choix de D ou M
                    {
                        nw.lettre = 'D';
                    }
                    else
                    {
                        nw.lettre = 'M';
                    }
                    return; // sort de la fonction car k = num_case
                }
            }
        }
    }
}
```

```
/**
 * @file Commandes.h
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 1 05/01/2022
 * Fonctions de toutes les commandes
 * appellées directements par le main.cpp
 */

#ifndef _COMMANDES_
#define _COMMANDES_

#include "Commun.h"

 /**
  * @brief Permet la saisie d'un problème
  *
  * @param [out] p: problème à créer
  */
void saisie_problem(Problem& p);

/**
 * @brief Créer le problème
 *
 * @param [in] g: la grille qui va être généré avec le problème donné
 * @param [in] p: le problème
 */
void creation_problem(Grille& g, Problem& p);

/**
 * @brief Affiche le problème
 *
 * @param [in] p: le problème
 */
void affichage_problem(const Problem& p);

/**
 * @brief Désalloue le tableau de bombe
 *
 * @param [out] p: le priblème
 */
void desallocation_bombe(Problem& p);

/**
 * @brief Désalloue le tableau à deux
 *        dimensions
 *
 * @param [out] g: la grille
 */
void desallocation_grille(Grille& g);

/**
 * @brief Saisit les données necessaires
 *
 * @param [out] g: la grille
 * @param [out] p: le problème
 * @param [out] hc: l'historique des coups
 */
void saisie_grille(Grille& g, Problem& p, Historique_coup& hc);
```

```
/**
 * @brief Créer la grille pour ensuite exploiter
 *        les données lors de l'affichage
 *
 * @param [in] g: la grille de jeu
 * @param [in] p: le problème
 * @param [in] hc: l'historique de coup
 */
void creation_grille(Grille& g, Problem& p, Historique_coup hc);

/**
 * @brief Affiche une grille avec son contenue en fonction
 *        le l'état des cases
 *
 * @param [in] g: la grille de jeu
 * @param [in] p: le problème
 */
void affichage_grille(const Grille& g, const Problem& p);

/**
 * @brief Desalloue le tableau des coups
 *
 * @param [out] hc: l'historique de coups
 */
void desallocation_coup(Historique_coup& hc);

/**
 * @brief Fait des testes sur l'ensemble de la grille
 *        pour déterminer si la partie est gagné ou pas
 *        en affectant GAME_WON ou GAME_NOT_WON à l'etatj
 *
 * @param [in, out] g: la grille de jeu
 * @param [in] p: le problème
 */
void test_won(Grille& g, const Problem& p);

/**
 * @brief Affiche une ligne "game won" ou "game not won" en
 *        fonction de l'etatj de la grille
 *
 * @param [in] g: la grille de jeu
 */
void affichage_won(const Grille& g);

/**
 * @brief Fait des testes en comparant le dernier coup (car
 *        on ne peut que perde sur le dernier coup joué)
 *        avec les positions des bombes, en fonction du coup
 *        (M ou D) GAME_LOST ou GAME_NOT_LOST sera
 *        attrituer à l'etatj
 *
 * @param [in, out] g: la grille de jeu
 * @param [in] p: le problème
 * @param [in] hc: l'historique des coups
 */
void test_lost(Grille& g, const Problem& p, const Historique_coup& hc);

/**
 * @brief Affiche une ligne "game lost" ou "games not lost"
 *        fonction de l'etatj de la grille
 *
 * @param [in] g: la grille de jeu
```

```
 */
void affichage_lost(const Grille& g);

/**
 * @brief Initialise le nombre de lignes et colonnes dans
 *        la stucture de donnée
 *
 * @param [out] g: la grille
 */
void saisie_dimension_grille(Grille& g);

/**
 * @brief Créer un nouveau coup en fonction de
 *        la grille donnée
 *
 * @param [in] g: la grille
 * @param [in] nw: le nouveau coup
 */
void creat_new_move(Grille& g, Details_coups& nw);

/**
* @breif Affiche un nouveau coup
*
* @param [in] nc : le nouveau coup
*/
void affichage_new_move(const Details_coups& nc);

#endif



/**
 * @file Commandes.cpp
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 1 05/01/2022
 * Fonctions de toutes les commandes
 * appellées directements par le main.cpp
 */

#include <iostream>
using namespace std;

#include "Commandes.h"

void saisie_problem(Problem& p)
{
    cin >> p.lignes >> p.colonnes >> p.bombes;
}

void creation_problem(Grille& g, Problem& p)
{
    g.lignes = p.lignes;
    g.colonnes = p.colonnes;

    allocation_bombe(p);
    allocation_grille(g);
    initialisation_grille(g, p);
    place_bombes_debut(g, p);
    get_pos_bombes(g, p);
}
```

```
void affichage_problem(const Problem& p)
{
    cout << p.lignes << " ";
    cout << p.colonnes << " ";
    cout << p.bombes << " ";
    for (unsigned int i = 0; i < p.bombes; ++i)
    {
        cout << p.pos_bombe[i] << " ";
    }
}

void desallocation_bombe(Problem& p)
{
    delete[]p.pos_bombe;
}

void desallocation_grille(Grille& g)
{
    for (unsigned int i = 0; i < g.lignes; ++i)
    {
        delete[]g.grille_jeu[i];
    }
    delete[]g.grille_jeu;
}

//********************Fin des fonctions de la commande 1**********************

void saisie_grille(Grille& g, Problem& p, Historique_coup& hc)
{
    // récolte du problème
    cin >> p.lignes >> p.colonnes >> p.bombes;
    allocation_bombe(p);
    for (unsigned int i = 0; i < p.bombes; ++i)
    {
        cin >> p.pos_bombe[i];
    }

    g.lignes = p.lignes;
    g.colonnes = p.colonnes;
    allocation_grille(g);

    // récolte de l'historique des coups
    cin >> hc.nb_coup;
    allocation_coup(hc);
    for (unsigned int j = 0; j < hc.nb_coup; ++j)
    {
        cin >> hc.liste_coup[j].lettre;
        cin >> hc.liste_coup[j].position;
    }
}

void creation_grille(Grille& g, Problem& p, Historique_coup hc)
{
    initialisation_grille(g, p);
    initialisation_bombe(g, p);
    give_value_adjacent(g, p);
    reconnaissance_coup(g, p, hc);
}
```

```
void affichage_grille(const Grille& g, const Problem& p)
{
    cout << p.lignes << " " << p.colonnes << endl;

    for (unsigned int i = 0; i < p.colonnes; ++i)
    {
        cout << " ___";
    }
    cout << endl;

    for (unsigned int j = 0; j < p.lignes; ++j)
    {
        for (unsigned int k = 0; k < p.colonnes; ++k)
        {
            Etat_case etat = g.grille_jeu[j][k].etat;
            unsigned int contenue = g.grille_jeu[j][k].contenue;

            if (etat == MARQUER)
            {
                cout << "|" << " x ";
            }
            else if (etat == DEMASQUER)
            {
                if (contenue == 0)
                {
                    cout << "|" << "   ";
                }
                else if (contenue >= 1 && contenue <= 8)
                {
                    cout << "|" << " " << contenue << " ";
                }
                else
                {
                    cout << "|" << " m ";
                }
            }
            else // si etat = CACHER
            {
                cout << "|" << " . ";
            }
        }
        cout << "|" << endl;

        for (unsigned int l = 0; l < p.colonnes; ++l)
        {
            cout << " ___";
        }
        cout << endl;
    }
}

void desallocation_coup(Historique_coup& hc)
{
    delete[]hc.liste_coup;
}

//**********Fin des fonctions de la commande 2 non définies plus tôt**********
```

```
void test_won(Grille& g, const Problem& p)
{
    unsigned int compteur_bombe_marquer = 0;
    unsigned int compteur_case_demasquer = 0;
    int teste = 0;
    unsigned int total_case = p.lignes * p.colonnes;

    for (unsigned int i = 0; i < p.lignes; ++i)
    {
        for (unsigned int j = 0; j < p.colonnes; ++j)
        {
            if (g.grille_jeu[i][j].etat == MARQUER &&
                g.grille_jeu[i][j].contenue == 9)
            {
                ++compteur_bombe_marquer;
            }
            else if (g.grille_jeu[i][j].etat == DEMASQUER &&
                (g.grille_jeu[i][j].contenue <= 8 &&
                    g.grille_jeu[i][j].contenue >= 0))
            {
                ++compteur_case_demasquer;
            }
        }
    }

    if (compteur_bombe_marquer == p.bombes ||
        compteur_case_demasquer == total_case - p.bombes)
        /*total_case - p.bombes désingne le nombre total de case CACHER*/
    {
        g.etatj = GAME_WON;
    }
    else
    {
        g.etatj = GAME_NOT_WON;
    }
}

void affichage_won(const Grille& g)
{
    if (g.etatj == GAME_WON)
    {
        cout << "game won" << endl;
    }
    else
    {
        cout << "game not won" << endl;
    }
}

//******************Fin des fonctions propre à la commande 3*******************
```

```
void test_lost(Grille& g, const Problem& p, const Historique_coup& hc)
{
    int i = 0; // prends les valeurs 1 et -1

    if (hc.liste_coup[hc.nb_coup - 1].lettre == 'M')
    {

        for (unsigned int j = 0; j < p.bombes; ++j)
        {
            if (p.pos_bombe[j] == hc.liste_coup[hc.nb_coup - 1].position)
            {
                i = 1;
            }
        }
        /*si i = 0 alors le dernier coup joué est un marquage sur
        une case vide donc i = -1 pour indique que la partie est perdu*/
        if (i == 0)
        {
            i = -1;
        }
    }
    if (hc.liste_coup[hc.nb_coup - 1].lettre == 'D')
    {
        for (unsigned int k = 0; k < p.bombes; ++k)
        {
            if (p.pos_bombe[k] == hc.liste_coup[hc.nb_coup - 1].position)
            {
                i = -1;
            }
        }
        /*si i = 0 alors le dernier coup joué est un démasquage sur
        une case vide donc i = 1 pour indique que la partie n'est pas perdu*/
        if (i == 0)
        {
            i = 1;
        }
    }

    (i == 1) ? g.etatj = GAME_NOT_LOST : g.etatj = GAME_LOST;
}

void affichage_lost(const Grille& g)
{
    if (g.etatj == GAME_LOST)
    {
        cout << "game lost" << endl;
    }
    else
    {
        cout << "game not lost" << endl;
    }
}

//*****************Fin des fonctions propre à la commande 4*******************
```

```
void saisie_dimension_grille(Grille& g)
{
    cin >> g.lignes >> g.colonnes;
}

void creat_new_move(Grille& g, Details_coups& nw)
{
    allocation_grille(g);
    traitement_grille(g);
    choice_move(g, nw, count_hidden(g));
}

void affichage_new_move(const Details_coups& nc)
{
    cout << nc.lettre << nc.position << endl;
}

//******************Fin des fonctions propre à la commande 5******************



/**
 * @file main.cpp
 * Projet démineur
 * @author Anxian Zhang, Vick Ye
 * @version 8 05/01/2022
 * Application qui nous permet de jouer au
 * démineur de bout en bout
 */

#include <iostream>
#include <cassert>
using namespace std;

#include "Commandes.h"

int main(void)
{
    srand((unsigned)time(NULL));

    Grille jeu;
    Problem probleme;
    Historique_coup historique;
    Details_coups new_move;
    unsigned int num_commande;

    do
    {
        cin >> num_commande;
    } while (!(num_commande > 0 && num_commande <= 5));

    switch (num_commande)
    {
    case 1:
        // Génération du problème
        saisie_problem(probleme);
        creation_problem(jeu, probleme);
        affichage_problem(probleme);
        desallocation_bombe(probleme);
        desallocation_grille(jeu);
        break;
```

```
    case 2:
        // Génération de la grille
        saisie_grille(jeu, probleme, historique);
        creation_grille(jeu, probleme, historique);
        affichage_grille(jeu, probleme);
        desallocation_coup(historique);
        desallocation_bombe(probleme);
        desallocation_grille(jeu);
        break;
    case 3:
        // Partie gagné ?
        saisie_grille(jeu, probleme, historique);
        creation_grille(jeu, probleme, historique);
        test_won(jeu, probleme);
        affichage_won(jeu);
        desallocation_coup(historique);
        desallocation_bombe(probleme);
        desallocation_grille(jeu);
        break;
    case 4:
        // Partie perdu ?
        saisie_grille(jeu, probleme, historique);
        test_lost(jeu, probleme, historique);
        affichage_lost(jeu);
        desallocation_coup(historique);
        desallocation_bombe(probleme);
        desallocation_grille(jeu);
        break;
    case 5:
        // Création d'un nouveau coup
        saisie_dimension_grille(jeu);
        creat_new_move(jeu, new_move);
        affichage_new_move(new_move);
        desallocation_grille(jeu);
        break;
    default:
        break;
    }
    return 0;
}
```