DOSSIER DE DÉVELOPPEMENT LOGICIEL

DÉMINEUR - MINESWEEPER

1ère année - BUT Informatique
2022

JEAN-PIERRE Wania / Léa THAI

SOMMAIRE

PRÉSENTATION DU PROJET

Ce projet consiste à réaliser un jeu du démineur/minesweeper en C++, composé de 5 commandes élémentaires :

1. Production d'un problème
2. Production d'une grille
3. Indiquer si la partie est gagnée ou non
4. Indiquer si la partie est perdue ou non
5. Production d'un nouveau coup

Le but de ce jeu avec l'aide de ces commandes est de dévoiler toutes les cases qui ne sont pas minées afin de gagner la partie.

Une particularité du projet est qu'après l'appel d'une commande le programme se termine, ainsi aucune donnée n'est enregistrée, donc chaque commande s'appuie uniquement sur les informations données dans son appel que ce soit un problème, un historique de coup ou encore une grille.

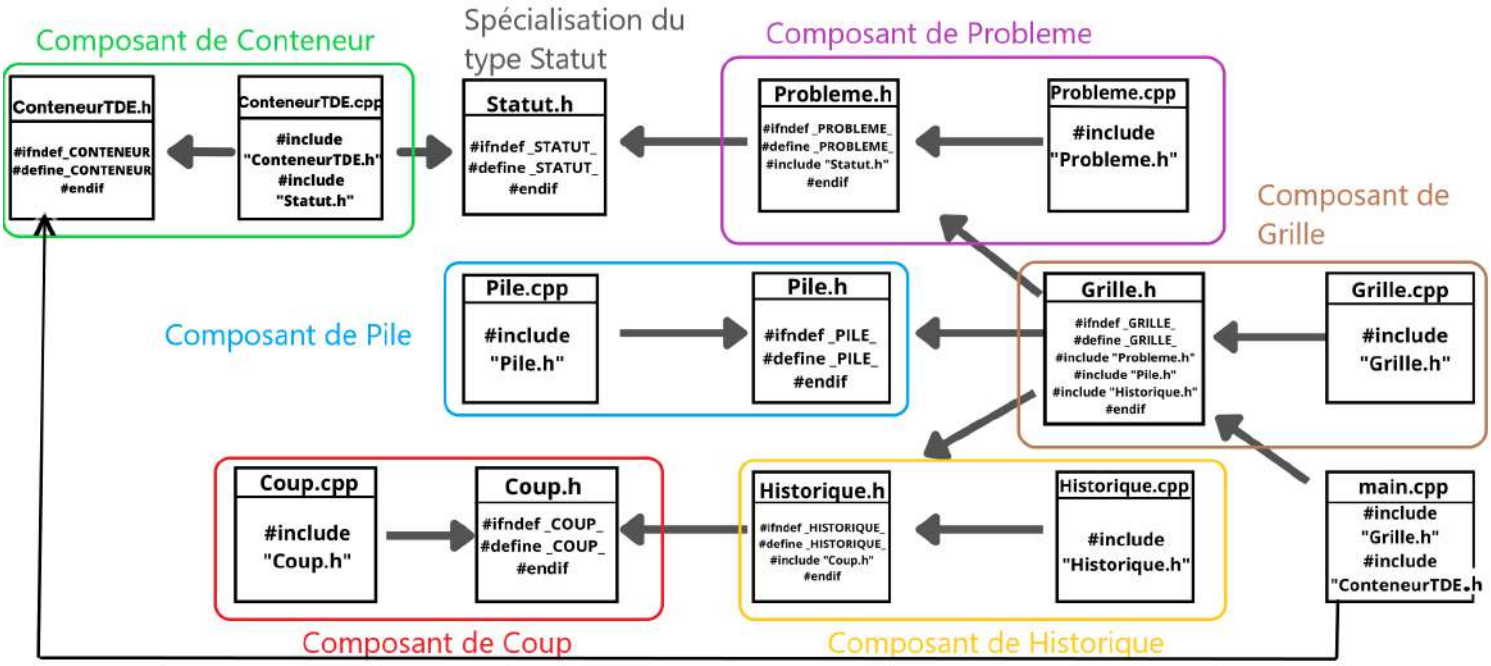
Une autre chose essentielle, est que l'on admet que les entrées n'ont aucune erreur et que marquer une case non-minée est considéré comme une erreur.

Le but final de ce projet est la création d'un jeu du démineur/minesweeper fonctionnel, mais nous pouvons y ajouter plusieurs sous objectifs :

1. La réalisation des différentes opérations élémentaires
2. Utiliser des structures et types particuliers pour faciliter la compréhension du code
3. Organiser le code avec des header et des cpp
4. Ne pas utiliser les conteneurs de la bibliothèque standard du C++
5. Respecter le format de saisie et d'affichage
6. Documenter le code

GRAPHE DE DÉPENDANCES

Voici le graphe de dépendances de nos différents fichiers, vous pouvez le retrouver en annexe



Graphe de Dépendances du projet Minesweeper

Le composant Pile et celui ConteneurTDE proviennent des différents que nous avons pu faire au cours du semestre.

Les autres composants, ou type ont été créés spécialement pour ce projet, une majorité d'entre eux sont exploités par le composant Grille.

LES JEUX D'ESSAIS

Afin de mener à bien ce projet, nous avons mis au point différents jeux d'essai in-out qui représente plusieurs cas importants à tester pour vérifier que notre programme fonctionne correctement :

- **Cas 0** - un jeu avec aucune mine
- **Cas Full Mine** - un jeu avec que des cases minées
- **Cas Carré 1** - un jeu avec une grille carrée
- **Cas Carré 2** - un jeu avec une grille carrée
- **Cas Rectangle Verticale 1** - un jeu avec une grille où le nombre de colonnes est inférieur au nombre de ligne
- **Cas Rectangle Verticale 2** - un jeu avec une grille où le nombre de colonnes est inférieur au nombre de ligne

Ces jeux d'essais ont été mis au point en respectant les formats d'entrées et d'affichage du code.

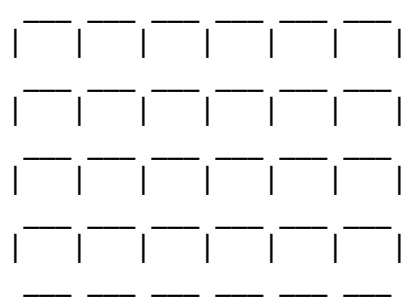
// Ces jeux de tests sont basées sur un cas 0, c'est à dire une partie où aucune mine n'est présente

in1.txt - 1 4 6 0

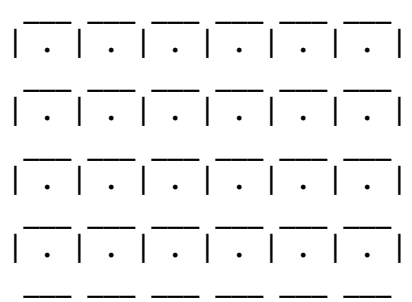
out1.txt - 4 6 0

in2a.txt - 2 4 6 0 1 D15
in2b.txt - 2 4 6 0 0
in2c.txt - 2 4 6 0 1 M15

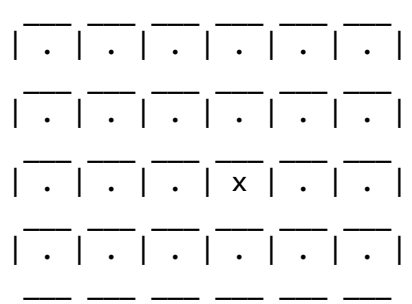
out2a.txt
4 6



out2b.txt
4 6



out2c.txt
4 6



in3a.txt - 3 4 6 0 1 D15
in3b.txt - 3 4 6 0 0
in3c.txt - 3 4 6 0 1 M15

out3a.txt - game won
out3bc.txt - game not won

in4a.txt - 4 4 6 0 1 D15
in4b.txt - 4 4 6 0 0
in4c.txt - 4 4 6 0 1 M15

out4ab.txt - game not lost
out4c.txt - game lost

// Ces jeux de tests sont basés sur des cas où la grille est carrée (nombre de colonnes = nombre de lignes)

in1.txt - 1 3 3 1

out1.txt - 3 3 1 2

in2a.txt - 2 3 3 1 1 1 D6
in2b.txt - 2 3 3 1 1 2 D6 M1
in2c.txt - 2 3 3 1 1 2 D6 D1

out2a.txt -
3 3

```
 ___ ___ ___
| . | . | . |
 ___ ___ ___
| 1 | 1 | 1 |
 ___ ___ ___
|   |   |   |
 ___ ___ ___
```

out2b.txt -
3 3

```
 ___ ___ ___
| . | x | . |
 ___ ___ ___
| 1 | 1 | 1 |
 ___ ___ ___
|   |   |   |
 ___ ___ ___
```

out2c.txt -
3 3

```
 ___ ___ ___
| . | m | . |
 ___ ___ ___
| 1 | 1 | 1 |
 ___ ___ ___
|   |   |   |
 ___ ___ ___
```

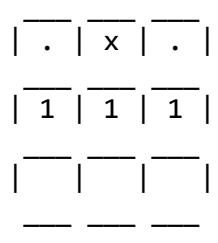
in3a.txt - 3 3 3 1 1 1 D6
in3b.txt - 3 3 3 1 1 2 D6 M2
in3c.txt - 3 3 3 1 1 2 D6 D2

out3abc.txt - game not won

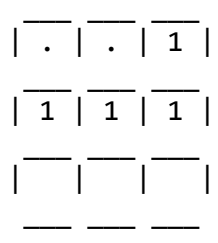
in4a.txt - 4 3 3 1 1 1 D6
in4b.txt - 4 3 3 1 1 2 D6 M2
in4c.txt - 4 3 3 1 1 2 D6 D2

out4ac.txt - game not lost
out4b.txt - game lost

in5a.txt -
5 3 3



in5b.txt -
5 3 3



out5a.txt - D0
out5b.txt - M1

// Ces jeux de tests sont basés sur des cas où la grille est carrée (nombre de colonnes = nombre de lignes)

in1.txt - 1 3 3 3

out1.txt - 3 3 3 1 2 5

in2.txt - 2 3 3 3 1 2 5 1 D6

out2.txt -
3 3

```
 ___ ___ ___
| . | . | . |
 ___ ___ ___
| 1 | . | . |
 ___ ___ ___
|   | 1 | . |
 ___ ___ ___
```

in3a.txt - 3 3 3 3 1 2 5 1 D6
in3b.txt - 3 3 3 3 1 2 5 2 D6 D0

out3ab.txt - game not won

in4a.txt - 4 3 3 3 1 2 5 1 D6
in4b.txt - 4 3 3 3 1 2 5 2 D6 D0

out4a.txt
out4b.txt - game not lost

in5a.txt -
5 3 3

```
 ___ ___ ___
| . | . | . |
 ___ ___ ___
| 1 | 3 | . |
 ___ ___ ___
|   | 1 | 1 |
 ___ ___ ___
```

in5b.txt   -
5 3 3

```
 ___ ___ ___
| . | x | x |
 ___ ___ ___
| 1 | 3 | x |
 ___ ___ ___
|   | 1 | 1 |
 ___ ___ ___
```

out5a.txt - M5
out5b.txt - D0

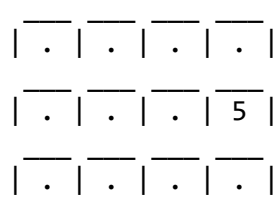
// Ces jeux de tests sont basés sur des cas où toutes les cases sont minées

in1.txt - 1 3 4 12

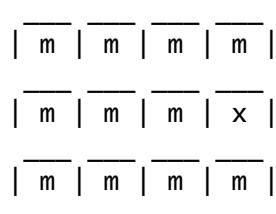
out1.txt - 3 4 12 0 1 2 3 4 5 6 7 8 9 10 11

in2a.txt - 2 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 D7
in2b.txt - 2 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 M7
in2c.txt - 2 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 M10
in2d.txt - 2 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 D5

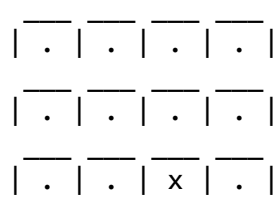
out2a.txt -
3 4



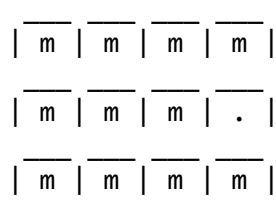
out2b.txt -
3 4



out2c.txt -
3 4



out2d.txt -
3 4



in3a.txt - 3 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 D7
in3b.txt - 3 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 M7

in3c.txt - 3 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 M10
in3d.txt - 3 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 D5

out3a.txt - game won
out3bcd.txt - game not won

in4a.txt - 4 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 D7
in4b.txt - 4 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 M7
in4c.txt - 4 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 M10
in4d.txt - 4 3 4 11 0 4 6 8 1 9 2 10 3 5 11 1 D5

out4ac.txt - game not lost
out4bd.txt - game lost

in5a.txt -
5 3 4

```
 ___ ___ ___ ___
| . | . | . | . |
 ___ ___ ___ ___
| . | . | . | 5 |
 ___ ___ ___ ___
| . | . | . | . |
 ___ ___ ___ ___
```

in5b.txt -
5 3 4

```
 ___ ___ ___ ___
| . | . | . | . |
 ___ ___ ___ ___
| . | . | . | . |
 ___ ___ ___ ___
| . | . | x | . |
 ___ ___ ___ ___
```

out5a.txt - M3
out5b.txt - D8

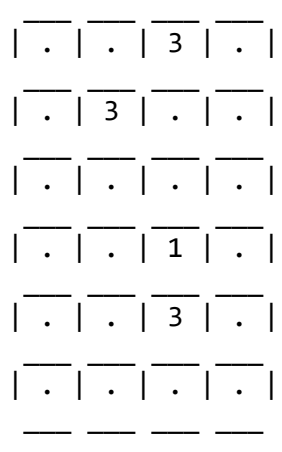
// Ces jeux de tests sont basés sur des cas où la grille est rectangulaire à la verticale (nombre de colonnes < nombre de lignes)

in1.txt - 1 6 4 7

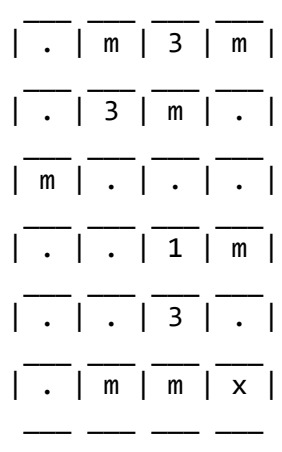
out1.txt - 6 4 7 1 3 6 8 15 21 22

in2a.txt - 2 6 4 7 1 3 6 8 15 21 22 4 D2 D18 D14 D5
in2b.txt - 2 6 4 7 1 3 6 8 15 21 22 5 D2 D18 D14 D5 M23
in2c.txt - 2 6 4 7 1 3 6 8 15 21 22 5 D2 D18 D14 D5 D3

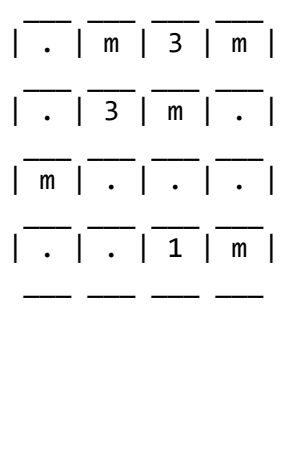
out2a.txt -
6 4



out2b.txt -
6 4



out2c.txt -
6 4



```
| . | . | 3 | . |
 ___ ___ ___ ___
| . | m | m | . |
 ___ ___ ___ ___
```

in3a.txt - 3 6 4 7 1 3 6 8 15 21 22 4 D2 D18 D14 D5
in3b.txt - 3 6 4 7 1 3 6 8 15 21 22 5 D2 D18 D14 D5 M23
in3c.txt - 3 6 4 7 1 3 6 8 15 21 22 5 D2 D18 D14 D5 D3

out3abc.txt - game not won

in4a.txt - 4 6 4 7 1 3 6 8 15 21 22 4 D2 D18 D14 D5
in4b.txt - 4 6 4 7 1 3 6 8 15 21 22 5 D2 D18 D14 D5 M23
in4c.txt - 4 6 4 7 1 3 6 8 15 21 22 5 D2 D18 D14 D5 D3

out4a.txt - game not lost
out4bc.txt - game lost

in5.txt -
5 6 4

```
 ___ ___ ___ ___
| . | . | 3 | . |
 ___ ___ ___ ___
| . | 3 | . | . |
 ___ ___ ___ ___
| . | . | . | . |
 ___ ___ ___ ___
| . | . | 1 | . |
 ___ ___ ___ ___
| . | . | 3 | . |
 ___ ___ ___ ___
| . | . | . | . |
 ___ ___ ___ ___
```

out5.txt - D13

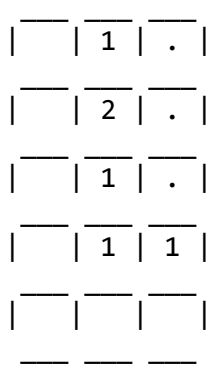
// Ces jeux de tests sont basés sur des cas où la grille est rectangulaire à la verticale (nombre de colonnes < nombre de lignes)

in1.txt - 1 5 3 2

out1.txt - 5 3 2 8 2

in2.txt - 2 5 3 2 8 2 1 D0

out2.txt -
5 3



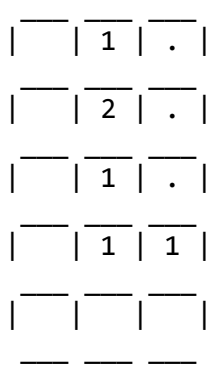
in3.txt - 3 5 3 2 8 2 1 D0

out3.txt - game not won

in4.txt - 4 5 3 2 8 2 1 D0

out4.txt - game not lost

in5.txt -
5 5 3



out5.txt - M8

BILAN

Dans sa globalité, ce projet a été très intéressant et amusant à faire toutefois, nous avons rencontré quelques difficultés, une des principales étant répartition de la charge de travail dans le binôme puisque l'une d'entre nous à plus de facilité que l'autre, faisant que cette personne peut rapidement se sentir surchargée par la quantité de travail.

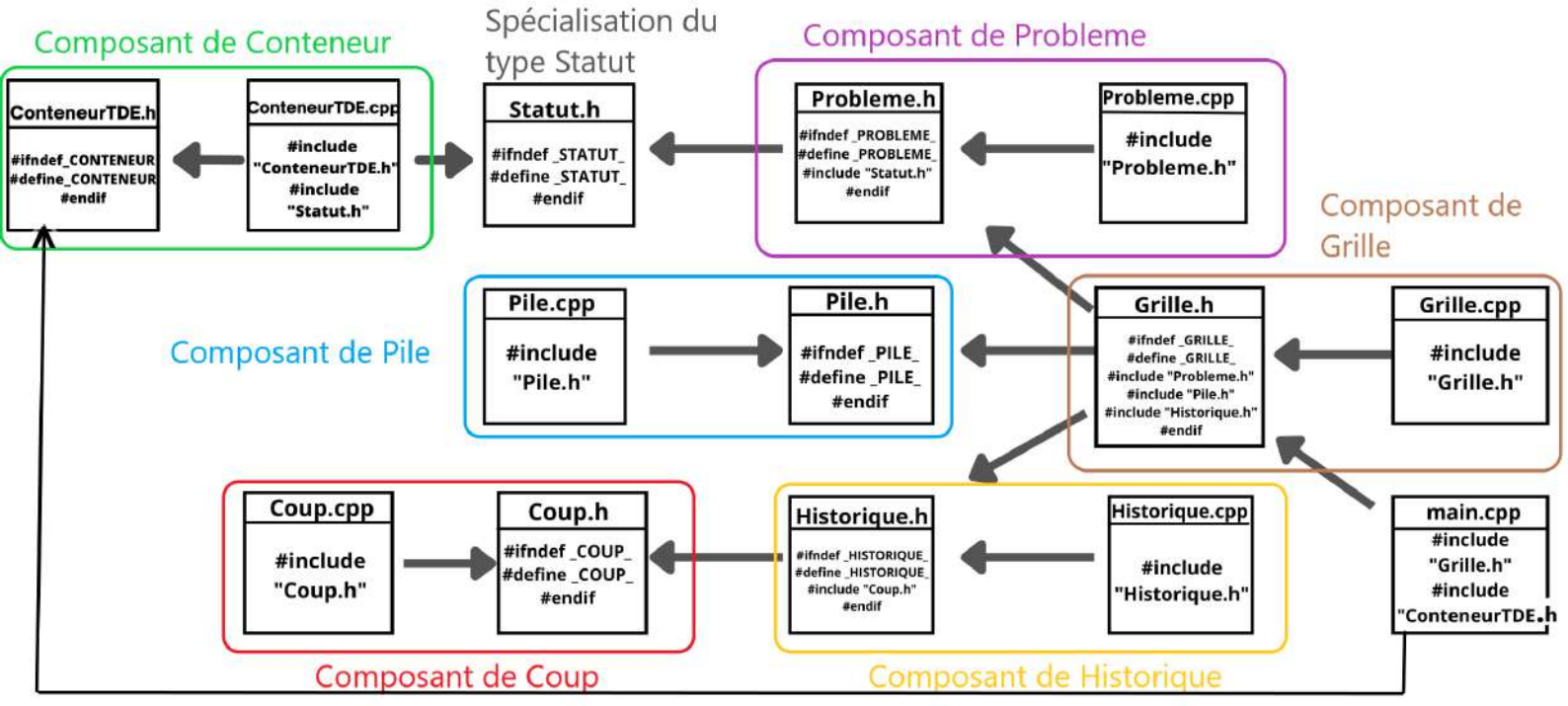
Ensuite, la gestion fut aussi un problème significatif, ce qui est normal puisque cela est lié à la répartition de la charge de travail. Un autre point essentiel est la relecture du code, la création de la documentation et l'organisation du code qui sont des étapes très importantes, mais aussi très chronophages et ennuyantes, que nous avons pris beaucoup de temps à faire.

Pour faire face à ces difficultés beaucoup de points pourraient être améliorés, comme la coordination du travail d'équipe et la répartition de la charge de travail. D'autres choses sont à améliorer telles que la communication, la connaissance de nos cours (nous sommes restées bloquées longtemps sur un problème à cause de l'ordre des priorités) et la gestion du temps.

Malgré les diverses difficultés et choses à améliorer, il y a des choses que nous trouvons très réussies dans notre projet comme la documentation et les commentaires, comparé à notre dernier projet, ces derniers sont plus présents, mais pas inutilement et aide réellement à la compréhension du code. La compilation séparée, nous trouvons que notre code est très bien organisé, aidé par la compilation séparée justement qui montre bien la manière dont nous avons pensé notre code.

Ce projet est bien mieux que notre dernier, dans sa globalité nous trouvons qu'il est bien plus réussi, que notre organisation malgré les quelques difficultés est bien meilleure. En conclusion, nous sommes beaucoup plus fières de ce projet.

ANNEXE


Composant de Conteneur
ConteneurTDE.h
#ifndef_CONTENEUR
#define_CONTENEUR
#endif
ConteneurTDE.cpp
#include
"ConteneurTDE.h"
#include
"Statut.h"
Spécialisation du
type Statut
Statut.h
#ifndef _STATUT_
#define _STATUT_
#endif
Composant de Probleme
Probleme.h
#ifndef _PROBLEME_
#define _PROBLEME_
#include "Statut.h"
#endif
Probleme.cpp
#include
"Probleme.h"
Composant de
Grille
Composant de Pile
Pile.cpp
#include
"Pile.h"
Pile.h
#ifndef _PILE_
#define _PILE_
#endif
Grille.h
#ifndef _GRILLE_
#define _GRILLE_
#include "Probleme.h"
#include "Pile.h"
#include "Historique.h"
#endif
Grille.cpp
#include
"Grille.h"
Coup.cpp
#include
"Coup.h"
Coup.h
#ifndef _COUP_
#define _COUP_
#endif
Historique.h
#ifndef _HISTORIQUE_
#define _HISTORIQUE_
#include "Coup.h"
#endif
Historique.cpp
#include
"Historique.h"
main.cpp
#include
"Grille.h"
#include
"ConteneurTDE.h
Composant de Coup
Composant de Historique

```
#ifndef _STATUT_
#define _STATUT_

/**
 * @file Statut.h
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 1 - 12/12/2021
 * @brief Spécialisation du Type Statut
 */

 /* @brief Structure Statut d'une case, qui peut
prendre 4 valeurs : mine, marque, masque, desmasque
 */
enum Statut { mine = -1, marque = -2, masque = -3,
marqueSp = -4, demasque = 0 };

#endif
```

```
#ifndef _PROBLEME_
#define _PROBLEME_

/**
 * @file Probleme.h
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 4 - 08/01/2022
 * @brief Composant de Problème
 */

#include "Statut.h"

/*
* @brief Type Probleme
* Contient toutes les informations d'un problème :
nombre de lignes, de colonnes et de mines, et un
tableau avec l'emplacement des mines
*/
struct Probleme {
      unsigned int lignes;
      unsigned int colonnes;
      unsigned int nbMines;
      unsigned int* emplacementMines; //Tableau
dynamique contenant l'emplacement des bombes
      // Possède la taille du nombre de Mines
};

/*
* @brief Crée un problème et l'affiche
*/
void creerProbleme();

/*
* @brief Affiche un problème
```

```
* @param[in] pb : le problème à afficher
*/
void afficherProbleme(Probleme& pb);

/*
* @brief Crée un tableau avec l'emplacement des
mines choisi aléatoirement
* @param[in] nbMines : le nombre de mines
* @return un tableau contenant l'emplacement de
mines
*/
unsigned int* nvTabMine(unsigned int nbMines,
unsigned int nbCases);

/*
* @brief Initialise un tableau de Statut
* @param[in-out] tableau : tableau contenant des
statuts
* @param[in] taille : taille du tableau
* @param[in] nbMines : nombre de mines
*/
void initialiser(Statut* tableau, unsigned int
taille, unsigned nbMines);


#endif

/**
 * @file Probleme.cpp
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 4 - 08/01/2022
 * @brief Composant de Problème
 */

#include <iostream>
#include <cassert>
#include "Probleme.h"
#pragma warning(disable : 4996)
using namespace std;

/*
* @brief Crée un problème et l'affiche
*/
void creerProbleme(){
      Probleme pb;
      cin >> pb.lignes >> pb.colonnes >> pb.nbMines;
      assert(pb.nbMines <= (pb.lignes *
pb.colonnes));
      pb.emplacementMines = nvTabMine(pb.nbMines,
pb.lignes * pb.colonnes);

      afficherProbleme(pb);
}

/*
* @brief Affiche un problème
* @param[in] pb : le problème à afficher
```

```
*/
void afficherProbleme(Probleme& pb){
	cout << pb.lignes << " " << pb.colonnes << " "
<< pb.nbMines;

	for (unsigned int i = 0; i < pb.nbMines; ++i)
{
		cout << " " << pb.emplacementMines[i];
	}
}

/*
* @brief Crée un tableau contenant l'emplacement des
mines choisi aléatoirement
* @param[in] nbMines : le nombre de mines
* @param[in] taille : la taille du tableau
* @return un tableau contenant l'emplacement de
mines
*/
unsigned int* nvTabMine(unsigned int nbMines,
unsigned int nbCases) {
	Statut* tableau = new Statut[nbCases];
	unsigned int* tabMines = new unsigned
int[nbMines];

	initialiser(tableau, nbCases, nbMines);
//initialisation du tableau de Statut

	for (unsigned int i = 0; i < nbMines; ++i) {
		tabMines[i] = i;
	}

	unsigned int case1, case2; // variable
temporaire pour stocker les valeurs des cases
	unsigned int nbEchange = nbCases * nbMines *
2; // détermine le nombre d'échanges de cases à
faire
	unsigned int tmp;
	Statut echange;

	for (unsigned int i = 0; i <= nbEchange; ++i)
{
		/* Tirage aléatoire des cases */
		case1 = rand() % nbCases;
		case2 = rand() % nbCases;

		if (case1 == case2 || tableau[case1] ==
mine && tableau[case2] == mine) {
			continue;
		}
		else {
			if (tableau[case1] == mine) {
				tmp = 0;
				while (tabMines[tmp] !=
case1) { // on cherche l'indice où est la position
de la bombe
					++tmp;
				}
```

```
                        tabMines[tmp] = case2; // on
change la position à l'indice de la bombe
                  }

                  if (tableau[case2] == mine) {
                        tmp = 0;
                        while (tabMines[tmp] !=
case2) {
                              ++tmp;
                        }
                        tabMines[tmp] = case1;
                  }

                  /* Echange des cases */
                  echange = tableau[case1]; // on
stocke la valeur de la première valeur pour ne pas
la perdre
                  tableau[case1] = tableau[case2];
                  tableau[case2] = echange;
            }
      }

      return tabMines;
}

/*
* @brief Initialise un tableau de Statut
* @param[in-out] tableau : tableau contenant des
statuts
* @param[in] taille : taille du tableau
* @param[in] nbMines : nombre de mines
*/
void initialiser(Statut* tableau, unsigned int
taille, unsigned nbMines) {
      for (unsigned int i = 0; i < taille; ++i) { //
initialisation de toutes les cases du tableau à
masque
            tableau[i] = masque;

            if (i < nbMines) { // place les mines
sur les premières cases pour faciliter la phase
d'aléatoirisation
                  tableau[i] = mine;
            }
      }
}

#ifndef _COUP_
#define _COUP_

/**
 * @file Coup.h
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 2 - 27/12/2021
 * @brief Type Coup
 */
```

```
 /* @brief Type Coup comporte le type du coup et la
position
 */
struct Coup {
    char type; // 1 seul élément donc pas de tableau
    unsigned int position;

};

/*@brief Effectue le traitement d'un coup
* @param[in] coup : le coup a traité
* @param[in-out] nvC : le nouveau coup
*/
void traiterCoup(Coup& nvC);

#endif

/**
 * @file Coup.cpp
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 1 - 27/12/2021
 * @brief Type Coup
 */

#include <iostream>
#include "Coup.h"
using namespace std;
#pragma warning(disable : 4996)
```

```
/*@brief Effectue le traitement d'un coup
* @param[in] coup : le coup a traité
* @param[in-out] nvC : le nouveau coup
*/
void traiterCoup(Coup& nvC) {
    cin >> nvC.type >> nvC.position;
}


#ifndef _HISTORIQUE_
#define _HISTORIQUE_

/**
 * @file Historique.h
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 3 - 09/01/2021
 * @brief Composant de Historique
 */

#include "Coup.h"

 /* L'historique qui a le nombre de coups et la
liste des coups
 */
struct Historique {
    unsigned int nbCoups;
    Coup* listeCoups; // Tableau dynamique contenant
```

```
les coups
};

/*@brief Initialise une structure Historique
* @param[in-out] histo : historique du jeu
*/
void initiaHisto(Historique& histo);


#endif

/*
 * @file Historique.cpp
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 3 - 27/12/2021
 * @brief Composant de Historique
 */

#include <iostream>
#include "Historique.h"
using namespace std;
#pragma warning(disable : 4996)


/*@brief Initialise une structure Historique
* @param[out] histo : historique du jeu
* @pram
*/
void initiaHisto(Historique& histo) {
	cin >> histo.nbCoups; // On récupère le nombre
de coups
	histo.listeCoups = new Coup[histo.nbCoups];

	for (unsigned int i = 0; i < histo.nbCoups;
++i) { // On remplit la liste des coups
		traiterCoup(histo.listeCoups[i]);
	}
}

#ifndef _GRILLE_
#define _GRILLE_

/**
 * @file Grille.h
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 6 - 08/01/2022
 * @brief Composant de Grille
 */

#include "Probleme.h"
#include "Historique.h"
#include "Pile.h"

/* @brief Type Grille,
* Contient toutes les informations d'une grille : sa
taille, le tableau contenant les Statuts de cases,
```

```
le probleme, l'historique de jeu
*/
struct Grille {
    unsigned int taille;
    int* tab; // Tableau dynamique de Statut
    Probleme jeuPb;
    Historique jeuHistorique;
};


/*
* @brief Affiche une Grille
* @param[in] grille : la grille à afficher
* @param[in] PoG : indique si la partie est perdue
ou non (change l'affichage)
*/
void afficherGrille(Grille& grille, unsigned int
PoG);

/*
* @brief Permet de d'initialiser correctement le
tableau de Grille
* @param[in-out] grille : la grille à remplir
* @return un nombre indiquant si la partie est perdu
ou non
*/
unsigned int abracadabra(Grille& grille); // Nom à
modifier
```

```
/* @brief Créer une Grille
*/
void creerGrille();

/*
* @brief Crée un problème pour un type Grille
* @param[in-out] pb : le problème à créer
*/
void creerNvProbleme(Probleme& pb);

/*
* @brief initialiser le Tableau d'un type Grille
* @param[in-out] grille : la grille dont le tableau
doit être initialisé
*/
void initialiserTabGrille(Grille& grille);

/*
* @brief Détermine le nombre de mines à proximité
* @param[in-out] tab: tableau de statut
* @param[in] position : la case
* @param[in] taille : la taille du tableau
* @param[in] nbColonnes : nombre de colonnes
* return le nombre de mines à proximité
*/
unsigned int nbMineProche(int* tab, unsigned int
position, unsigned int taille, unsigned int
nbColonnes);
```

```
/*
* @brief Empile les bonnes cases à proximité
* @param[in-out] tab: tableau de statut
* @param[in] position : la case
* @param[in] taille : la taille du tableau
* @param[in] nbColonnes : nombre de colonnes
* @param[in-out] abracadabra : la pile
*/
void queDoisJeEmpiler(int* tab, unsigned int
position, unsigned int taille, unsigned int
nbColonnes, Pile& abracadabra);

/* @brief Affiche si la partie est gagnée ou non
*/
void partieGagne();

/* @brief Affiche si la partie est perdue ou non
*/
void partiePerdu();



#endif
```

```
/**
 * @file Grille.cpp
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 6 - 08/01/2022
 * @brief Composant de Grille
 */

#include "Grille.h"
#include <iostream>
#include <cstdlib>
#include <cassert>
#include <iomanip>
#include <stdlib.h>
using namespace std;

/*
* @brief Affiche une Grille
* @param[in] grille : la grille à afficher
* @param[in] PoG : indique si la partie est perdue
ou gagnée (change l'affichage)
*/
void afficherGrille(Grille& grille, unsigned int
PoG) {
      unsigned int numeroCase = 0;
      cout << grille.jeuPb.lignes << " " <<
grille.jeuPb.colonnes << endl;
      for (unsigned int i = 0; i <
grille.jeuPb.lignes; ++i) { // la boucle
s'initialise sur les lignes car on print lignes par
lignes
            for (unsigned int m = 0; m <
grille.jeuPb.colonnes; m++) {
                  cout << " " << "___";
```

```
            }
            cout << endl;

            for (unsigned int m = 0; m <
grille.jeuPb.colonnes; m++) {
            cout << "| ";
                  switch (grille.tab[numeroCase]) {
                        case masque: cout << ". ";
                              break;

                        case marque:
                              if (PoG != 0) cout <<
"m ";
                              else cout << "x ";
                              break;

                        case marqueSp: // pour
afficher l'erreur
                              cout << "x ";
                              break;

                        case mine:
                              if (PoG != 0) cout <<
"m ";
                              else cout << ". ";
                              break;

                        default: //le cas par défaut
est celui des cas masque
                              if
(grille.tab[numeroCase] != 0) { //une case non-miné
et non-masqué à comme valeur 0 (donc demasque)
                                    cout <<
grille.tab[numeroCase] << " "; // ou le nombre de
bombe proche dans ce cas on le print
                              }
                              else cout << "  ";

                  }
                  ++numeroCase;
            }
        cout << "|" << endl;
      }

      for (unsigned int m = 0; m <
grille.jeuPb.colonnes; m++) {
            cout << " " << "___";
      }
}

/*
* @brief Permet de d'initialiser correctement le
tableau de Grille
* @param[in-out] grille : la grille à remplir
* @return un nombre indiquant si la partie est perdu
ou non
*/
unsigned int abracadabra(Grille& grille) { //Nom à
```

```
changer
    Coup coup;
    Pile flood;
    unsigned int caseAct = 0;
    int tmp = 0;
    initialiser(flood, grille.taille, 1); //
initialisation de la pile

    for (unsigned int i = 0; i <
grille.jeuHistorique.nbCoups; ++i) { // On regarde
l'historique de coups
        coup = grille.jeuHistorique.listeCoups[i];
        if (coup.type == 'D') { // C'est de la merde
pensé à mettre ' ' et pas " " car erreur vu que le C
c'est de la merde
                if (grille.tab[coup.position] ==
mine) {
                      return 1;
                }

                /* REMPLISSAGE PAR DIFFUSION */
            empiler(flood, coup.position);

            while (!(estVide(flood))) {
                caseAct = sommet(flood); // On
récupère la valeur que l'on va traiter
                depiler(flood); // On dépile pour
enlever la case que l'on traite
                        //On donne le nombre de mine
à proximité comme valeur de la case pour les traiter
plus facilement à l'affichage
                        grille.tab[caseAct] =
nbMineProche(grille.tab, caseAct, grille.taille,
grille.jeuPb.colonnes);

                if (grille.tab[caseAct]==0) { // On
vérifie qu'il n'y a pas de mines à proximité

queDoisJeEmpiler(grille.tab, caseAct, grille.taille,
grille.jeuPb.colonnes, flood);
                }
            }
        }
        else { // Si le type du coup est 'M' donc
marqué
                if (grille.tab[coup.position] !=
mine) { // Si la case qu'on veut marqué n'est pas
une mine, on perd
                        grille.tab[coup.position] =
marqueSp; //la mauvaise case marqué doit apparaître
dans l'affichage on utilise donc un statut spécial
                        return 1;
                }
            grille.tab[coup.position] = marque;
        }
    }
      detruire(flood);
      return 0;
```

```
}

/* @brief Créer une Grille
*/
void creerGrille() {
    Grille grille;
    creerNvProbleme(grille.jeuPb);  // On récupère
le problème
    grille.taille = grille.jeuPb.colonnes *
grille.jeuPb.lignes;

    grille.tab = new int[grille.taille];
      initialiserTabGrille(grille); // On initialise
le tableau de statut
    initiaHisto(grille.jeuHistorique);

    unsigned int WoL = abracadabra(grille);
    afficherGrille(grille, WoL);
}

/*
* @brief Crée un problème pour un type Grille
* @param[in-out] pb : le problème à créer
*/
void creerNvProbleme(Probleme& pb) {
    cin >> pb.lignes >> pb.colonnes >> pb.nbMines;
    assert(pb.nbMines <= (pb.lignes * pb.colonnes));
    pb.emplacementMines = new unsigned
int[pb.nbMines];
    for (unsigned int i = 0; i < pb.nbMines; ++i) {
        cin >> pb.emplacementMines[i];
    }
}

/*
* @brief Détermine le nombre de mines à proximité
* @param[in-out] tab: tableau de statut
* @param[in] position : la case
* @param[in] taille : la taille du tableau
* @param[in] nbColonnes : nombre de colonnes
* return le nombre de mines à proximité
*/
unsigned int nbMineProche(int* tab, unsigned int
position, unsigned int taille, unsigned int
nbColonnes) {
      unsigned int nbMine = 0;

      /* Cas des coins */
      // Case 1er ligne tout à gauche
      if (position == 0) {
            if (tab[position + 1] == mine ||
tab[position + 1] == marque) ++nbMine;
            if (tab[position + nbColonnes] == mine
|| tab[position + nbColonnes] == marque) ++nbMine;
            if (tab[position + nbColonnes + 1] ==
mine || tab[position + nbColonnes + 1] == marque)
++nbMine;
            return nbMine;
```

```
	}
	//case 1er ligne tout à droite
	if (position + 1 == nbColonnes) {
		if (tab[position + nbColonnes] == mine
|| tab[position + nbColonnes] == marque) ++nbMine;
		if (tab[position + nbColonnes - 1] ==
mine || tab[position + nbColonnes - 1] == marque)
++nbMine;
		if (tab[position - 1 ] == mine ||
tab[position - 1] == marque) ++nbMine;
		return nbMine;
	}
	//case tout en bas à gauche
	if (position + nbColonnes == taille) {
		if (tab[position + 1] == mine ||
tab[position + 1] == marque) ++nbMine;
		if (tab[position - nbColonnes] == mine
|| tab[position - nbColonnes] == marque) ++nbMine;
		if (tab[position - nbColonnes + 1] ==
mine || tab[position - nbColonnes + 1] == marque)
++nbMine;
		return nbMine;
	}

	if (position + 1 == taille) {
		if (tab[position - 1] == mine ||
tab[position - 1] == marque) ++nbMine;
		if (tab[position - nbColonnes] == mine
|| tab[position - nbColonnes] == marque) ++nbMine;
		if (tab[position - nbColonnes - 1] ==
mine || tab[position - nbColonnes - 1] == marque)
++nbMine;
		return nbMine;
	}


	/****** Cas particulier des lignes et colonnes
en bordure ******/

	// toutes les cases de la dernière ligne sauf
les coins
	if (position + nbColonnes > taille) {
		if (tab[position + 1] == mine ||
tab[position + 1] == marque) ++nbMine;
		if (tab[position - 1] == mine ||
tab[position - 1] == marque) ++nbMine;
		if (tab[position - nbColonnes] == mine
|| tab[position - nbColonnes] == marque) ++nbMine;
		if (tab[position - nbColonnes + 1] ==
mine || tab[position - nbColonnes + 1] == marque)
++nbMine;
		if (tab[position - nbColonnes - 1] ==
mine || tab[position - nbColonnes - 1] == marque)
++nbMine;
		return nbMine;
	}

	// toutes les cases de la 1ere ligne sauf les
```

```
coins
      if (position - nbColonnes < 0) {
            if (tab[position + nbColonnes] == mine
|| tab[position + nbColonnes] == marque) ++nbMine;
            if (tab[position + nbColonnes + 1] ==
mine || tab[position + nbColonnes + 1] == marque)
++nbMine;
            if (tab[position + nbColonnes - 1] ==
mine || tab[position + nbColonnes - 1] == marque)
++nbMine;
            if (tab[position + 1] == mine ||
tab[position + 1] == marque) ++nbMine;
            if (tab[position - 1] == mine ||
tab[position - 1] == marque) ++nbMine;
            return nbMine;
      }

      // COLONNES Particulière

      // toutes les cases de la colonne de droite
sauf les coins
      if ((position +1) % nbColonnes == 0) {
            if (tab[position + nbColonnes] == mine
|| tab[position + nbColonnes] == marque) ++nbMine;
            if (tab[position + nbColonnes] == mine
|| tab[position + nbColonnes] == marque) ++nbMine;
            if (tab[position - 1] == mine ||
tab[position - 1] == marque) ++nbMine;
            if (tab[position - nbColonnes] == mine
|| tab[position - nbColonnes] == marque) ++nbMine;
            if (tab[position - nbColonnes - 1] ==
mine || tab[position - nbColonnes - 1] == marque)
++nbMine;
            return nbMine;
      }

      // toutes les cases de la colonne de gauche
sauf les coins
      if (position % nbColonnes == 0) {
            if (tab[position + nbColonnes] == mine
|| tab[position + nbColonnes] == marque) ++nbMine;
            if (tab[position + nbColonnes] == mine
|| tab[position + nbColonnes] == marque) ++nbMine;
            if (tab[position + 1] == mine ||
tab[position + 1] == marque) ++nbMine;
            if (tab[position - nbColonnes] == mine
|| tab[position - nbColonnes] == marque) ++nbMine;
            if (tab[position - nbColonnes + 1] ==
mine || tab[position - nbColonnes + 1] == marque)
++nbMine;
            return nbMine;
      }

      //Cas par défaut

      //toute le cases sauf ceux précisées plus
haut, cases du centre
      if (tab[position + nbColonnes] == mine ||
```

```
tab[position + nbColonnes] == marque) ++nbMine;
      if (tab[position + nbColonnes + 1] == mine ||
tab[position + nbColonnes + 1] == marque) ++nbMine;
      if (tab[position + nbColonnes - 1] == mine ||
tab[position + nbColonnes - 1] == marque) ++nbMine;
      if (tab[position + 1] == mine || tab[position
+ 1] == marque) ++nbMine;
      if (tab[position - 1] == mine || tab[position
- 1] == marque) ++nbMine;
      if (tab[position - nbColonnes] == mine ||
tab[position - nbColonnes] == marque) ++nbMine;
      if (tab[position - nbColonnes - 1] == mine ||
tab[position - nbColonnes - 1] == marque) ++nbMine;
      if (tab[position - nbColonnes + 1] == mine ||
tab[position - nbColonnes + 1] == marque) ++nbMine;
      return nbMine;
}

/*
* @brief Empile les bonnes cases à proximité
* @param[in-out] tab: tableau de statut
* @param[in] position : la case
* @param[in] taille : la taille du tableau
* @param[in] nbColonnes : nombre de colonnes
* @param[in-out] abracadabra : la pile
*/
void queDoisJeEmpiler(int* tab, unsigned int
position, unsigned int taille, unsigned int
nbColonnes, Pile& abracadabra) {
      /* Cas des coins */
      // Case 1er ligne tout à gauche
      if (position == 0) {
            if (tab[position + 1] == masque)
empiler(abracadabra, position + 1);
            if (tab[position + nbColonnes] ==
masque) empiler(abracadabra, position + nbColonnes);
            return;
      }
      //case 1er ligne tout à droite
      if (position + 1 == nbColonnes) {
            if (tab[position - 1] == masque)
empiler(abracadabra, position - 1);
            if (tab[position + nbColonnes] ==
masque) empiler(abracadabra, position + nbColonnes);
            return;
      }
      //case tout en bas à gauche
      if (position + nbColonnes == taille) {
            if (tab[position + 1] == masque)
empiler(abracadabra, position + 1);
            if (tab[position - nbColonnes] ==
masque) empiler(abracadabra, position - nbColonnes);
            return;
      }
      //case tout en bas à droite
      if (position + 1 == taille) {
            if (tab[position - 1] == masque)
```

```
empiler(abracadabra, position - 1);
            if (tab[position - nbColonnes] ==
masque) empiler(abracadabra, position - nbColonnes);
            return;
      }


      /****** Cas particulier des lignes et colonnes
en bordure ******/

      // LIGNES
      // toutes les cases de la dernière ligne sauf
les coins
      if (position + nbColonnes > taille) {
            if (tab[position + 1] == masque)
empiler(abracadabra, position + 1);
            if (tab[position - 1] == masque)
empiler(abracadabra, position - 1);
            if (tab[position - nbColonnes] ==
masque) empiler(abracadabra, position - nbColonnes);
            return;
      }

      // toutes les cases de la 1ere ligne sauf les
coins
      if (position - nbColonnes < 0) {
            if (tab[position + 1] == masque)
empiler(abracadabra, position + 1);
            if (tab[position - 1] == masque)
empiler(abracadabra, position - 1);
            if (tab[position + nbColonnes] ==
masque) empiler(abracadabra, position + nbColonnes);
            return;
      }

      // COLONNES Particulière

      // toutes les cases de la colonne de droite
sauf les coins
      if ((position + 1) % nbColonnes == 0) {
            if (tab[position - 1] == masque)
empiler(abracadabra, position - 1);
            if (tab[position - nbColonnes] ==
masque) empiler(abracadabra, position - nbColonnes);
            if (tab[position + nbColonnes] ==
masque) empiler(abracadabra, position + nbColonnes);
            return;
      }

      // toutes les cases de la colonne de gauche
sauf les coins
      if (position % nbColonnes == 0) {
            if (tab[position + 1] == masque)
empiler(abracadabra, position + 1);
            if (tab[position - nbColonnes] ==
masque) empiler(abracadabra, position - nbColonnes);
            if (tab[position + nbColonnes] ==
masque) empiler(abracadabra, position + nbColonnes);
```

```
            return;
      }

      //Cas par défaut

      //toute le cases sauf ceux précisées plus
haut, cases du centre
      if (tab[position + nbColonnes] == masque)
empiler(abracadabra, position + nbColonnes);
      if (tab[position - nbColonnes] == masque)
empiler(abracadabra, position - nbColonnes);
      if (tab[position - 1] == masque)
empiler(abracadabra, position - 1);
      if (tab[position + 1] == masque)
empiler(abracadabra, position + 1);
      return;
}

/*
* @brief initialiser le Tableau d'un type Grille
* @param[in-out] grille : la grille dont le tableau
doit être initialisé
*/
void initialiserTabGrille(Grille& grille) {
      for (unsigned int i = 0; i < grille.taille;
++i) { // on initialise tout le tableau au statut
masque
            grille.tab[i] = masque;
      }
      for (unsigned int i = 0; i <
grille.jeuPb.nbMines; ++i) { // puis on met le
statut mine aux bonnes cases

grille.tab[grille.jeuPb.emplacementMines[i]] = mine;
      }
}

/* @brief Affiche si la partie est gagnée ou non
*/
void partieGagne() {
      Grille grille;
      creerNvProbleme(grille.jeuPb);
      grille.taille = grille.jeuPb.colonnes *
grille.jeuPb.lignes;
      grille.tab = new int[grille.taille];

      initialiserTabGrille(grille);

      initiaHisto(grille.jeuHistorique);

      unsigned int WoL = abracadabra(grille);

      if (WoL != 0) { //valeur retourné par
abracadabra indique si la partie est perdu ou pas
            cout << "game not won";
      }
      else {
```

```
            /*Pour savoir si une partie est gagné on
doit vérifier que tous les cases qui ne sont pas des
mines soient découvertes*/
            unsigned int nbCaseDecou = 0;
            for (unsigned int i = 0; i <
grille.taille; ++i) {
                  if (grille.tab[i] == masque) {
//si un case masque est toujours présente la partie
n'est pas gagné
                        cout << "game not won";
                        break;
                  }
                  if(grille.tab[i] >= demasque)
++nbCaseDecou; // demasque = 0
            }
            if (nbCaseDecou == (grille.taille -
grille.jeuPb.nbMines)) cout << "game won";
      }
}

/* @brief Affiche si la partie est perdue ou non
*/
void partiePerdu() {
      Probleme pb;
      Historique histo;
      creerNvProbleme(pb);
      initiaHisto(histo);

      unsigned int i;

      if (histo.listeCoups[histo.nbCoups-1].type ==
'D') { //On vérifie qu'on a pas demasque une mine
            for (i = 0; i < pb.nbMines; ++i) {
                  if
(histo.listeCoups[histo.nbCoups-1].position ==
pb.emplacementMines[i]) { // Si la position du Coup
est celle d'une mine
                        cout << "game lost";
                        break;
                  }
            }
            if( i == pb.nbMines) cout << "game not
lost"; // i a parcouru le tableau sans erreur (if
jamais valide)
      }

      else { // Cas où le type du Coup est 'M'
            for (i = 0; i < pb.nbMines; ++i) { //On
vérifie qu'on a pas masqué une non-mine
                  if
(histo.listeCoups[histo.nbCoups-1].position ==
pb.emplacementMines[i]) { // Si la position du Coup
est celle d'une mine
                        cout << "game not lost"; //
La Case marquée est correcte
                        break;
                  }
            }
            if( i == pb.nbMines) cout << "game
```

```
lost"; // marqué une case sans mine
      }
}


#ifndef _PILE_
#define _PILE_

/**
 * @file Pile.h
 * Projet sem04-cours-Cpp2
 * @author l'équipe pédagogique
 * @version 1 - 29/11/2014
 * @brief Composant de pile à capacité paramétrée
 */

struct Pile {
      unsigned int capacite;  // capacité de la pile
(c>0)
      unsigned int pasExtension;
      unsigned int* tab;                        //
tableau des éléments de pile en mémoire dynamique
      int sommet;                   // indice de
sommet de pile dans tab
};

 /**
 * @brief Initialiser une pile vide
 * la pile est allouée en mémoire dynamique
 * @see detruire, pour une désallocation en fin
d'utilisation
 * @param[out] p : la pile à initialiser
 * @param[in] c : capacité de la pile (nb maximum
d'items stockés)
 * @param[in] extension : le pas d'extension de la
pile
 * @pre c>0
 */
void initialiser(Pile& p, unsigned int c, unsigned
extension);


/**
 * @brief Désallouer une pile
 * @see initialiser, la pile a déjà été initialisée
 * @param[in,out] p : la pile à désallouer
 */
void detruire(Pile& p);

/**
 * @brief Test de pile pleine
 * @param[in] p : la pile testée
 * @return true si p est pleine, false sinon
 */
bool estPleine(const Pile& p);

/**
 * @brief Test de pile vide
```

```
 * @param[in] p : la pile testée
 * @return true si p est vide, false sinon
 */
bool estVide(const Pile& p);

/**
 * @brief Lire l'item au sommet de pile
 * @param[in] p : la pile
 * @return la valeur de l'item au sommet de pile
 * @pre la pile n'est pas vide
 */
unsigned int sommet(const Pile& p);

/**
 * @brief Empiler un item sur une pile si pile
pleine ajoute de l'espace
 * @param[in,out] p : la pile
 * @param[in] it : l'item à empiler
 */
void empiler(Pile& p, const unsigned int& it);

/**
 * @brief Dépiler l'item au sommet de pile
 * @param[in,out] p : la pile
 * @pre la pile n'est pas vide
 */
void depiler(Pile& p);

#endif
```

```
/**
 * @file Pile.cpp
 * Projet sem04-cours-Cpp2
 * @author l'équipe pédagogique
 * @version 1 - 29/11/2014
 * @brief Composant de pile à capacité paramétrée
 */

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

#include "Pile.h"

 /**
* @brief Initialiser une pile vide
* la pile est allouée en mémoire dynamique
* @see detruire, pour une désallocation en fin
d'utilisation
* @param[out] p : la pile à initialiser
* @param[in] c : capacité de la pile (nb maximum
d'unsigned ints stockés)
* @param[in] extension : le pas d'extension de la
pile
* @pre c>0
*/
```

```
void initialiser(Pile& p, unsigned int c, unsigned
extension) {
	assert(c>0);
	p.capacite = c;
	p.pasExtension = extension;
	p.tab = new unsigned int[c];
	p.sommet = -1;
}

/**
 * @brief Désallouer une pile
 * @see initialiser, la pile a déjà été initialisée
 * @param[in,out] p : la pile à désallouer
 */
void detruire(Pile& p) {
	delete [] p.tab;
	p.tab = NULL;
}

/**
 * @brief Test de pile pleine
 * @param[in] p : la pile testée
 * @return true si p est pleine, false sinon
 */
bool estPleine(const Pile& p) {
	return (p.sommet == (p.capacite-1));
}

/**
 * @brief Test de pile vide
 * @param[in] p : la pile testée
 * @return true si p est vide, false sinon
 */
bool estVide(const Pile& p) {
	return (p.sommet == -1);
}

/**
 * @brief Lire l'unsigned int au sommet de pile
 * @param[in] p : la pile
 * @return la valeur de l'unsigned int au sommet de
pile
 * @pre la pile n'est pas vide
 */
unsigned int sommet(const Pile& p) {
	assert(!estVide(p));
	return p.tab[p.sommet];
}

/**
 * @brief Empiler un unsigned int sur une pile si
pile pleine ajoute de l'espace
 * @param[in,out] p : la pile
 * @param[in] it : l'unsigned int à empiler
 */
void empiler(Pile& p, const unsigned int& it) {
	if (estPleine(p) == true) {
		unsigned int* nPile;
```

```
            unsigned int newCapa = (p.capacite + 1)
* p.pasExtension;
            nPile = new unsigned int[newCapa];

            for (int i = 0; i < p.capacite; ++i) {
                  nPile[i] = p.tab[i];
            }

            p.capacite = newCapa;
            delete[] p.tab;
            p.tab = nPile;

            cout << "Extension -
Allocaton/Réallocation de " << newCapa *
sizeof(unsigned int)
                  << " octets (" << newCapa << "
unsigned ints)." << endl;

      }
      p.sommet++;
      p.tab[p.sommet] = it;
}

/**
 * @brief Dépiler l'unsigned int au sommet de pile
 * @param[in,out] p : la pile
 * @pre la pile n'est pas vide
 */
void depiler(Pile& p) {
      assert(!estVide(p));
      p.sommet--;
}


/**
 * @file ConteneurTDE.h
 * Projet Minesweeper
 * @author l'équipe pédagogique et Wania Jean-Pierre
 * @version 4 08/01/22
 * @brief Composant de ConteneurTDE
 */

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



/** @brief Conteneur d'unsigned ints alloués en
mémoire dynamique
 *  de capacité extensible suivant un pas
d'extension
 */
struct ConteneurTDE {
      unsigned int capacite;     // capacité du
conteneur (>0)
      unsigned int pasExtension; // pas d'extension
du conteneur (>0)
```

```
    unsigned int nbElemDansTab = 0;
    unsigned int* tab;              // conteneur
alloué en mémoire dynamique
};

/*
 * @brief Initialise un conteneur d'unsigned int
 * Allocation en mémoire dynamique du conteneur
d'items
 * de capacité (capa) extensible par pas d'extension
(p)
 * @see detruire, pour sa désallocation en fin
d'utilisation
 * @param[out] c : le conteneur d'items
 * @param [in] capa : capacité du conteneur
 * @param [in] p : pas d'extension de capacité
 * @pre capa>0 et p>0
 */
void initialiser(ConteneurTDE& c, unsigned int capa,
unsigned int p);

/**
 * @brief Désalloue un conteneur d'items en mémoire
dynamique
 * @see initialiser, le conteneur d'unsigned int a
déjà été alloué
 * @param[out] c : le conteneur d'unsigned int
 */
void detruire(ConteneurTDE& c);
```

```
/**
 * @brief Lecture d'un unsigned int d'un conteneur
d'unsigned int
 * @param[in] c : le conteneur d'unsigned int
 * @param[in] i : la position de l'unsigned int dans
le conteneur
 * @return l'unsigned int à la position i
 * @pre i < c.capacite
 */
unsigned int lire(const ConteneurTDE& c, int i);

/**
 * @brief Ecrire un unsigned int dans un conteneur
d'unsigned int
 * @param[in,out] c : le conteneur d'unsigned int
 * @param[in] i : la position où ajouter/modifier
l'item
 * @param[in] it : l'unsigned int à écrire
 */
void ecrire(ConteneurTDE& c, unsigned int i,
unsigned int it);


/* @brief Génére un nouveau coup
*/
void nvCoup();

/*
```

```
* @brief Remplie grille
*  @param[in] nombreCharLigne : au nombre max de
caractere dans une ligne
*  @param[in] *tab : le tableau
*  @param[in] nbrL : nombre de ligne dans la grille
*  @param[in-out] posNbMineCase : nombre de ligne
dans la grille

*/
void remplissageGrille(unsigned int nombreCharLigne,
int* tab, unsigned int nbrL, ConteneurTDE&
posNbMineCase);

/**
* @brief Echange les valeurs des indices i et j dans
un tableau
* @param[in-out] c : le conteneur
* @param[in] i : le premier indice
* @param[in] j : le deuxieme indice
*/

void echanger(ConteneurTDE& c, unsigned int i,
unsigned int j);


/**
* @brief Sélectionne le pivot et tri une partie du
tableau
* @param[in-out] c : le conteneur
* @param[in] tableauGrille : la grille remplie
* @param[in] debut : le premier indice
* @param[in] fin : le deuxieme indice
* return un indice
*/
int repartition(ConteneurTDE& c, int* tableauGrille,
int debut, int fin);


/**
* @brief Fait des appels récursifs pout trier la
partie gauche et droite du pivot
* @param[in-out] c : le conteneur
* @param[in] tableauGrille : la grille remplie
* @param[in] debut : le premier indice
* @param[in] fin : le deuxieme indice
*/
void TriRapideRec(ConteneurTDE& c, int*
tableauGrille, int debut, int fin);


/* Tri rapide dans l'ordre décroissant d'un tableau
* @param[in-out] c : le conteneur
* @param[in] tableauGrille : la grille remplie
*/
void TriRapide(ConteneurTDE& c, int* tableauGrille);


/*
```

```
* @brief Indique s'il y a des cases découvertes sans
mines proches, à proximité de la case masquée
* @param[in] tabGrille : un tableau de statut
* @param[in] nbrC : nombre de colonnes
* @param[in] nbrL : nombre de lignes
* @param[in] caseMasque : la case à vérifier
* @return true ou false
*/
bool checkZero(int* tabGrille, unsigned int nbrC,
unsigned int nbrL, unsigned int caseMasque);

/*
* @brief Remplie le conteneur caseMasqueProxi qui
contient les positions des cases masqués autour
d'une autre donnée et indique si on peut marquer une
case
* @param[in] tableauGrille : tableau de statut
* @param[in] caseMineProx : la case traité
* @param[in-out] caseMasqueProxi : le conteneur
remplie
* @param[in] nbrC : le nombre de colonnes
* @param[in] nbrL : le nombre de lignes
* @return true ou false
*/
bool checkCarre(int* tableauGrille, unsigned int
CaseMineProx, ConteneurTDE& caseMasqueProxi,
unsigned int nbrC, unsigned int nbrL);
```

```
/**
* @brief Remplie le conteneur caseMasqueProxi qui
contient les positions des cases masqués autour
d'une autre donnée et indique si on peut marquer une
case
* @param[in] tableauGrille: tableau de statut
* @param[in] CaseMineProx : case actuelle
* @param[in-out] caseMasqueProxi : le conteneur
rempli
* @param[in] nbrC : nombre de colonnes
* @param[in] nbrL : nombre de lignes
* return true ou false
*/
bool checkCarrePourMarque(int* tableauGrille,
unsigned int CaseMineProx, ConteneurTDE&
caseMasqueProxi, unsigned int nbrC, unsigned int
nbrL);

/**
 * @file ConteneurTDE.cpp
 * Projet Minesweeper
 * @author l'équipe pédagogique et Wania Jean-Pierre
 * @version 4 08/01/22
 * @brief Composant de ConteneurTDE
 */

#include <iostream>
#include <cassert>
```

```cpp
#include "Statut.h"
#include "ConteneurTDE.h"
#include <stdlib.h>
#include <cstdlib>
#include <time.h>
#pragma warning(disable : 4996)
using namespace std;


/**
 * @brief Initialise un conteneur d'unsigned ints
 * Allocation en mémoire dynamique du conteneur
d'unsigned ints
 * de capacité (capa) extensible par pas d'extension
(p)
 * @see detruire, pour sa désallocation en fin
d'utilisation
 * @param[out] c : le conteneur d'unsigned ints
 * @param [in] capa : capacité du conteneur
 * @param [in] p : pas d'extension de capacité
 * @pre capa>0 et p>0
 */
void initialiser(ConteneurTDE& c, unsigned int capa,
unsigned int p) {
	assert(capa > 0 && p > 0);
	c.capacite = capa;
	c.pasExtension = p;

	c.tab = new unsigned int[capa];
}

/**
 * @brief Désalloue un conteneur d'unsigned ints en
mémoire dynamique
 * @see initialiser, le conteneur d'unsigned ints a
déjà été alloué
 * @param[out] c : le conteneur d'unsigned ints
 */
void detruire(ConteneurTDE& c) {
	if (c.tab != NULL) {
		delete[] c.tab;
	}
}

/**
 * @brief Lecture d'un unsigned int d'un conteneur
d'unsigned ints
 * @param[in] c : le conteneur d'unsigned ints
 * @param[in] i : la position de l'unsigned int dans
le conteneur
 * @return l'unsigned int à la position i
 * @pre i < c.capacite
 */
 unsigned int lire(const ConteneurTDE& c, int i) {

	assert(i < c.capacite);
	return c.tab[i];
}
```

```
/**
 * @brief Ecrire un unsigned int dans un conteneur
d'unsigned ints
 * @param[in,out] c : le conteneur d'unsigned ints
 * @param[in] i : la position où ajouter/modifier
l'unsigned int
 * @param[in] it : l'unsigned int à écrire
 */
void ecrire(ConteneurTDE& c, unsigned int i,
unsigned int it) {
	unsigned int newTaille;
	unsigned int* newT;
	if (i>=c.capacite) {
		/* Stratégie de réallocation
proportionnelle au pas d'extension :
		 * initialisez la nouvelle taille du
conteneur (newTaille)
		 * à (i+1) * c.pasExtension */
		newTaille = i + 1 * c.pasExtension;
		/* Allouez en mémoire dynamique un
nouveau tableau (newT)
		 * à cette nouvelle taille*/
		/* Recopiez les unsigned ints déjà
stockés dans le conteneur */
		newT = new unsigned int[newTaille];

		for (int i = 0; i < c.capacite; ++i) {
			newT[i] = c.tab[i];
		}
		/* Désallouez l'ancien tableau support
du conteneur */
		delete[] c.tab;
	/* Actualiser la mise à jour du conteneur en
mémoire dynamique
	 * Faites pointer le tableau support du
conteneur
	 * sur le nouveau tableau en mémoire dynamique
*/
	c.tab = newT;
	/* Actualisez la taille du conteneur */
		c.capacite = newTaille;
	}
	/* Ecriture de l'unsigned int (it) à la
position i dans le conteneur */
	c.tab[i] = it;
	c.nbElemDansTab += 1;
}

/****************************************** TRI
************************************************
/

/**
* @brief Echange les valeurs des indices i et j dans
un tableau
* @param[in-out] c : le conteneur
* @param[in] i : le premier indice
```

```
* @param[in] j : le deuxieme indice
*/

void echanger(ConteneurTDE& c, unsigned int i,
unsigned int j) {
	assert(i >= 0 && j < c.nbElemDansTab);
	unsigned int tmp = c.tab[i];
	c.tab[i] = c.tab[j];
	c.tab[j] = tmp;
}


/**
* @brief Sélectionne le pivot et tri une partie du
tableau
* @param[in-out] c : le conteneur
* @param[in] debut : le premier indice
* @param[in] fin : le deuxieme indice
*/


int repartition(ConteneurTDE& c,int* tableauGrille,
int debut, int fin) {
	int pivot = lire(c, fin); // pivot prend la
denière val du conteneur, qui en faite la possition
case qui a une mine a proximitée
	int i = debut - 1;
	for (unsigned int j = debut; j < fin - 1; j++)
{
		if ( tableauGrille[lire(c, j)] <
tableauGrille[pivot]) { // ordre croissant, utili
			i += 1;
			echanger(c, i, j);

		}
	}
	echanger(c, ++i, fin);
	return i; // pas besoin de i+1 du au ++i
}


/**
* @brief Fait des appels récursifs pout trier la
partie gauche et droite du pivot
* @param[in-out] c : le conteneur
* @param[in] debut : le premier indice
* @param[in] fin : le deuxieme indice
*/

void TriRapideRec(ConteneurTDE& c, int*
tableauGrille, int debut, int fin) {
	int p;
	if (fin - debut >= 1) { // s'il reste des
éléments
		p = repartition(c, tableauGrille, debut,
fin); // séletionne pivot
		TriRapideRec(c, tableauGrille, debut, p
- 1); // trie partie gauche du pivot
```

```
            TriRapideRec(c, tableauGrille, p + 1,
fin); // trie partie droite du pivot
      }
}


/**
* @brief Tri rapide dans l'ordre décroissant d'un
tableau
* @param[in-out] c : le conteneur
*/

void TriRapide(ConteneurTDE& c, int* tableauGrille)
{
      TriRapideRec(c, tableauGrille, 0,
c.nbElemDansTab - 1);
}

/****************************************** FIN
TRI
************************************************
/


/* @brief Génére un nouveau coup
*/
void nvCoup() {
      unsigned int nbrL, nbrC;
      cin >> nbrL >> nbrC;
      unsigned int nombreCharLigne = 4 * nbrC + 2;
// schéma répétitif de 4 caractères ex: "| x " , le
+2 = \0 + "|"
      int* tableauGrille = new int[nbrL * nbrC];
      ConteneurTDE posCaseMineProx;
      initialiser(posCaseMineProx, nbrC, nbrC);
      remplissageGrille(nombreCharLigne,
tableauGrille, nbrL, posCaseMineProx);
      TriRapide(posCaseMineProx, tableauGrille);
      bool nvCoupPossible = false;

      //Parcours inverse car tri croissant, et on
veut en décroissant
      for (int i = posCaseMineProx.nbElemDansTab -
1; i >= 0; --i) {
            ConteneurTDE caseMasqueProxi;
            initialiser(caseMasqueProxi, 2, 2);

            nvCoupPossible =
checkCarrePourMarque(tableauGrille,
posCaseMineProx.tab[i], caseMasqueProxi, nbrC,
nbrL);
            if (nvCoupPossible == true) {
                  unsigned int caseMarque = rand() %
caseMasqueProxi.nbElemDansTab;
                  cout << "M" <<
lire(caseMasqueProxi, caseMarque);
                  break;
            }
```

```
            detruire(caseMasqueProxi);
      }


      if (nvCoupPossible == false) {
            for (int i =
posCaseMineProx.nbElemDansTab - 1; i >= 0; --i) { //
Sens inverse
                  ConteneurTDE caseMasqueProxi;
                  initialiser(caseMasqueProxi, 2,
2);

                  nvCoupPossible =
checkCarre(tableauGrille, posCaseMineProx.tab[i],
caseMasqueProxi, nbrC, nbrL);
                  if (nvCoupPossible == true) {
                        unsigned int caseDemasque =
rand() % caseMasqueProxi.nbElemDansTab;
                        cout << "D" <<
lire(caseMasqueProxi, caseDemasque);
                        break;
                  }
                  detruire(caseMasqueProxi);
            }
      }

      //dernier cas pour démasquer
      if (nvCoupPossible == false &&
posCaseMineProx.nbElemDansTab !=0) {
            unsigned int i = 0;
            bool cbon = false;
            do {
                  ConteneurTDE caseMasqueProxi;
                  unsigned int caseMineProxi =
lire(posCaseMineProx, i);
                  initialiser(caseMasqueProxi, 2,
2);
                  bool verif =
checkCarre(tableauGrille, caseMineProxi,
caseMasqueProxi, nbrC, nbrL);
                  ++i;
                  if(caseMasqueProxi.nbElemDansTab
!= 0){
                        cbon = true;
                        unsigned int caseDemasque =
rand() % caseMasqueProxi.nbElemDansTab;
                        cout << "D" <<
lire(caseMasqueProxi, caseDemasque);
                        nvCoupPossible = true;
                  }
                  detruire(caseMasqueProxi);
            } while (cbon == false);

            detruire(posCaseMineProx);
      }

      //Cas où aucune case n'a encore été dévoilé
ou, où tout ceux du dessus sont impossibles
```

```
	if (nvCoupPossible == false) {
		unsigned int nvCoup = rand() % (nbrC *
nbrL);;
		while (tableauGrille[nvCoup] != masque)
{
			++nvCoup;
			if (nvCoup == (nbrC * nbrL))
nvCoup = 0;
		}
		cout << "D" << nvCoup;

	}
}

/*
* @brief Indique s'il y a des cases découvertes sans
mines proches, à proximité de la case masquée
* @param[in] tabGrille : un tableau de statut
* @param[in] nbrC : nombre de colonnes
* @param[in] nbrL : nombre de lignes
* @param[in] caseMasque : la case à vérifier
* @return true ou false
*/
bool checkZero(int* tabGrille, unsigned int nbrC,
unsigned int nbrL, unsigned int caseMasque) {
	if (caseMasque > nbrC) { //Nord
		if (tabGrille[caseMasque - nbrC] ==
demasque) return false; // pas de mines à
proximitées
	}
	if ((caseMasque + nbrC) < (nbrC * nbrL)) {
//Sud
		if (tabGrille[caseMasque + nbrC] ==
demasque) return false;
	}
	if (caseMasque % nbrC != 0) { //Ouest
		if (tabGrille[caseMasque - 1] ==
demasque) return false;
	}
	if ((caseMasque + 1) % nbrC != 0) { //Est
		if (tabGrille[caseMasque + 1] ==
demasque) return false;
	}

	if (caseMasque > nbrC && caseMasque % nbrC !=
0) { //Nord-Ouest
		if (tabGrille[caseMasque - nbrC - 1] ==
demasque) return false;
	}
	if (caseMasque > nbrC && (caseMasque + 1) %
nbrC != 0) { //Nord-Est

		if (tabGrille[caseMasque - nbrC + 1] ==
demasque) return false;
	}

	if (caseMasque + nbrC < (nbrC * nbrL) &&
```

```
caseMasque % nbrC != 0) { //Sud-Ouest
            if (tabGrille[caseMasque + nbrC - 1] ==
demasque) return false;
      }

      if ((caseMasque + nbrC) < (nbrC * nbrL) &&
(caseMasque + 1) % nbrC != 0) { //Sud-Est
            if (tabGrille[caseMasque + nbrC + 1] ==
demasque) return false;
      }

      return true;
}

/*
* @brief Remplie grille
*  @param[in] nombreCharLigne : au nombre max de
caractere dans une ligne
*  @param[in-out] tab : le tableau
*  @param[in] nbrL : nombre de ligne dans la grille
*/
void remplissageGrille(unsigned int nombreCharLigne,
int* tab, unsigned int nbrL, ConteneurTDE&
posCaseMineProx) {
      char* ligne = new char[nombreCharLigne];
      unsigned int nCase = 0;
      for (unsigned int i = 0; i < nbrL; ++i) {
            cin >> ws;
            cin.getline(ligne, nombreCharLigne); //
On récupère la ligne d'underscore

            cin >> ws;
            cin.getline(ligne, nombreCharLigne); //
On écrase l'ancienne ligne sauvegarde pour traiter
celle qui nous intéresse

            //On commence la boucle à la valeur 2
car c'est la position de la première valeur
d'intérêt
            for (unsigned int m = 2; m <
strlen(ligne); m += 4) { //voir schéma répétitif
(ligne 402), la valeur intéressante se trouve
toujours à 4 indice

                                                //de la
précedente dû au schéma répétitif
                  if (sscanf(&ligne[m], "%d",
&tab[nCase]) == true) { // On vérifie qu'une
conversion est possible, si c'est possible
                        ecrire(posCaseMineProx,
posCaseMineProx.nbElemDansTab, nCase); // stock dans
posCaseMineProx (contene) la position des cases
ayant des mines à proximité
                        ++nCase; // stock positions
                  }
                  else {
                        if (ligne[m] == '.') {
                              tab[nCase] = masque;
```

```
                        ++nCase;
                    }
                    else {
                        if (ligne[m] == ' ') {
                            tab[nCase] =
demasque;
                            ++nCase;
                        }
                        else {
                            if (ligne[m] ==
'x') {
                                tab[nCase]
= marque;
                                ++nCase;
                            }
                        }
                    }
                }
            }
        }

        cin >> ws;
        cin.getline(ligne, nombreCharLigne); // on
récupère la dernière ligne d'underscore
}

/**
* @brief Indique si on peut marquer une case et
remplie le conteneur
* @param[in] tableauGrille: tableau de statut
* @param[in] CaseMineProx : case actuelle
* @param[in-out] caseMasqueProxi : le conteneur
rempli
* @param[in] nbrC : nombre de colonnes
* @param[in] nbrL : nombre de lignes
*/

bool checkCarrePourMarque(int* tableauGrille,
unsigned int CaseMineProx, ConteneurTDE&
caseMasqueProxi,  unsigned int nbrC, unsigned int
nbrL) {
        unsigned valCaseActuel =
tableauGrille[CaseMineProx];
        if (CaseMineProx >= nbrC) { //Nord
            if (tableauGrille[CaseMineProx - nbrC]
== masque && checkZero(tableauGrille, nbrC, nbrL,
CaseMineProx - nbrC) == true) {
                ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx - nbrC);
            }
            if (tableauGrille[CaseMineProx - nbrC]
== marque) --valCaseActuel; //On veut savoir si il y
a déjà des case marqué à proximité
        }
            if ((CaseMineProx + nbrC) < (nbrC *
nbrL)) { //Sud
                if (tableauGrille[CaseMineProx +
nbrC] == masque && checkZero(tableauGrille, nbrC,
```

```
nbrL, CaseMineProx + nbrC) == true) {
                        ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx + nbrC);
                  }
                  if (tableauGrille[CaseMineProx +
nbrC] == marque) --valCaseActuel;
            }
            if (CaseMineProx % nbrC != 0) { //Ouest
                  if (tableauGrille[CaseMineProx -
1] == masque && checkZero(tableauGrille, nbrC, nbrL,
CaseMineProx - 1) == true) {
                        ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx - 1);
                  }
                  if (tableauGrille[CaseMineProx -
1] == marque) --valCaseActuel;
            }
            if ((CaseMineProx + 1) % nbrC != 0) {
//Est
                  if (tableauGrille[CaseMineProx +
1] == masque && checkZero(tableauGrille, nbrC, nbrL,
CaseMineProx + 1) == true) {
                        ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx + 1);
                  }
                  if (tableauGrille[CaseMineProx +
1] == marque) --valCaseActuel;
            }
```

```
            if (CaseMineProx >= nbrC && CaseMineProx
% nbrC != 0) { //Nord-Ouest
                  if (tableauGrille[CaseMineProx -
nbrC - 1] == masque && checkZero(tableauGrille,
nbrC, nbrL, CaseMineProx - nbrC - 1) == true) {
                        ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx - nbrC -
1);
                  }
                  if (tableauGrille[CaseMineProx -
nbrC - 1] == marque) --valCaseActuel;
            }
            if (CaseMineProx >= nbrC &&
(CaseMineProx + 1) % nbrC != 0) { //Nord-Est
                  if (tableauGrille[CaseMineProx -
nbrC + 1] == masque && checkZero(tableauGrille,
nbrC, nbrL, CaseMineProx - nbrC + 1) == true) {
                        ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx - nbrC +
1);
                  }
                  if (tableauGrille[CaseMineProx -
nbrC + 1] == marque) --valCaseActuel;
            }

            if (CaseMineProx + nbrC < (nbrC * nbrL)
&& CaseMineProx % nbrC != 0) { //Sud-Ouest
                  if (tableauGrille[CaseMineProx +
nbrC - 1] == masque && checkZero(tableauGrille,
```

```
nbrC, nbrL, CaseMineProx + nbrC - 1) == true) {
                        ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx + nbrC -
1);
                }
                if (tableauGrille[CaseMineProx +
nbrC - 1] == marque) --valCaseActuel;
            }

            if ((CaseMineProx + nbrC) < (nbrC *
nbrL) && (CaseMineProx + 1) % nbrC != 0) { //Sud-Est
                if (tableauGrille[CaseMineProx +
nbrC + 1] == masque && checkZero(tableauGrille,
nbrC, nbrL, CaseMineProx + nbrC + 1) == true) {
                        ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx + nbrC +
1);
                }
                if (tableauGrille[CaseMineProx +
nbrC + 1] == marque) --valCaseActuel;
            }

            if (valCaseActuel >=
caseMasqueProxi.nbElemDansTab && valCaseActuel != 0)
return true;
            return false;
      }
```

```
/*
* @brief Remplie le conteneur caseMasqueProxi qui
contient les positions des cases masqués autour
d'une autre donnée et indique si on peut marquer une
case
* @param[in] tableauGrille : tableau de statut
* @param[in] caseMineProx : la case traité
* @param[in-out] caseMasqueProxi : le conteneur
remplie
* @param[in] nbrC : le nombre de colonnes
* @param[in] nbrL : le nombre de lignes
* @return
*/
bool checkCarre(int* tableauGrille, unsigned int
CaseMineProx, ConteneurTDE& caseMasqueProxi,
unsigned int nbrC, unsigned int nbrL) {
      unsigned int valCaseActuel =
tableauGrille[CaseMineProx];
      if (CaseMineProx >= nbrC) { //Nord
            if (tableauGrille[CaseMineProx - nbrC]
== masque) {
                  ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx - nbrC);
            }
            if (tableauGrille[CaseMineProx - nbrC]
== marque) --valCaseActuel;
      }
      if ((CaseMineProx + nbrC) < (nbrC * nbrL)) {
//Sud
```

```
            if (tableauGrille[CaseMineProx + nbrC]
== masque) {
                  ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx + nbrC);
            }
            if (tableauGrille[CaseMineProx + nbrC]
== marque) --valCaseActuel;
      }
      if (CaseMineProx % nbrC != 0) { //Ouest
            if (tableauGrille[CaseMineProx - 1] ==
masque) {
                  ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx - 1);
            }
            if (tableauGrille[CaseMineProx - 1] ==
marque) --valCaseActuel;
      }
      if ((CaseMineProx + 1) % nbrC != 0) { //Est
            if (tableauGrille[CaseMineProx + 1] ==
masque) {
                  ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx + 1);
            }
            if (tableauGrille[CaseMineProx + 1] ==
marque) --valCaseActuel;
      }

      if (CaseMineProx >= nbrC && CaseMineProx %
nbrC != 0) { //Nord-Ouest
            if (tableauGrille[CaseMineProx - nbrC -
1] == masque) {
                  ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx - nbrC -
1);
            }
            if (tableauGrille[CaseMineProx - nbrC -
1] == marque) --valCaseActuel;
      }
      if (CaseMineProx >= nbrC && (CaseMineProx + 1)
% nbrC != 0) { //Nord-Est
            if (tableauGrille[CaseMineProx - nbrC +
1] == masque) {
                  ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx - nbrC +
1);
            }
            if (tableauGrille[CaseMineProx - nbrC +
1] == marque) --valCaseActuel;
      }

      if (CaseMineProx + nbrC < (nbrC * nbrL) &&
CaseMineProx % nbrC != 0) { //Sud-Ouest
            if (tableauGrille[CaseMineProx + nbrC -
1] == masque) {
                  ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx + nbrC -
1);
            }
```

```
            if (tableauGrille[CaseMineProx + nbrC -
1] == marque) --valCaseActuel;
      }

      if ((CaseMineProx + nbrC) < (nbrC * nbrL) &&
(CaseMineProx + 1) % nbrC != 0) { //Sud-Est
            if (tableauGrille[CaseMineProx + nbrC +
1] == masque) {
                  ecrire(caseMasqueProxi,
caseMasqueProxi.nbElemDansTab, CaseMineProx + nbrC +
1);
            }
            if (tableauGrille[CaseMineProx + nbrC +
1] == marque) --valCaseActuel;
      }

      if (valCaseActuel == 0 &&
caseMasqueProxi.nbElemDansTab != 0) return true;

      return false;
}

/**
 * @file Demineur.cpp
 * Projet Minesweeper
 * @author Wania Jean-Pierre et Léa Thai
 * @version 2 - 24/12/2021
 */
```

```
#include <iostream>
#include "Grille.h"
#include "ConteneurTDE.h"
#pragma warning(disable : 4996)
using namespace std;

int main() {
      srand((unsigned int)time(NULL));
      unsigned int code;
      cin >> code;
      if (code == 1) {
            creerProbleme();
            exit(0);
      }

      if (code == 2) {
            creerGrille();
            exit(0);
      }

      if (code == 3) {
            partieGagne();
            exit(0);
      }

      if (code == 4) {
            partiePerdu();
            exit(0);
      }
```

```
	if (code == 5) {
		nvCoup();
		exit(0);
	}

	return 0;
}
```