Projet SDA : Les fondamentaux de la programmation impérative

Objet : **Rapport du projet SDA**

Université de Paris

Noms des participants : Esteves Gabriel, Lenouvel Louis

Groupe : 104

Table des matières

Présentation du projet

Brève présentation :

L'objectif de ce projet était de programmer un jeu intitulé « Démineur » ou « Minesweeper » (en Anglais) en plus d'utiliser tous types d'algorithmes qui font utiliser un grand nombre de fonctions dans le langage C++. Nous devions réaliser ce projet tout en utilisant la compilation séparée qui a permis d'avoir un code plus lisible avec des fichiers d'en têtes et des fichiers sources, séparant le prototype des fonctions du code de celles-ci. Le but de ce jeu était de trouver des mines cachées dans une grille représentant un champ de mines virtuel, avec pour seul indication le nombre de mines dans les zones adjacentes. Nous avions 5 commandes à réaliser pour ce jeu, que l'utilisateur devait entrer pour pouvoir jouer. Pour chaque commande, il y avait un fichier texte disponible, en entrée « in » qu'on devait rentrer sur l'invite de commande et un fichier de sortie « out » qui devait être le résultat attendu en sortie avec notre programme.

Rôles et entrées sorties des commandes :

Pour la commande numéro 1, Le fichier texte entré correspond au numéro de la commande suivi du nombre de lignes, de colonnes et de mines entré par l'utilisateur. L'objectif était de retourner un problème, soit le nombre de lignes, de colonnes ainsi que le nombres de mines avec leurs positions correspondantes, placé aléatoirement au préalable.

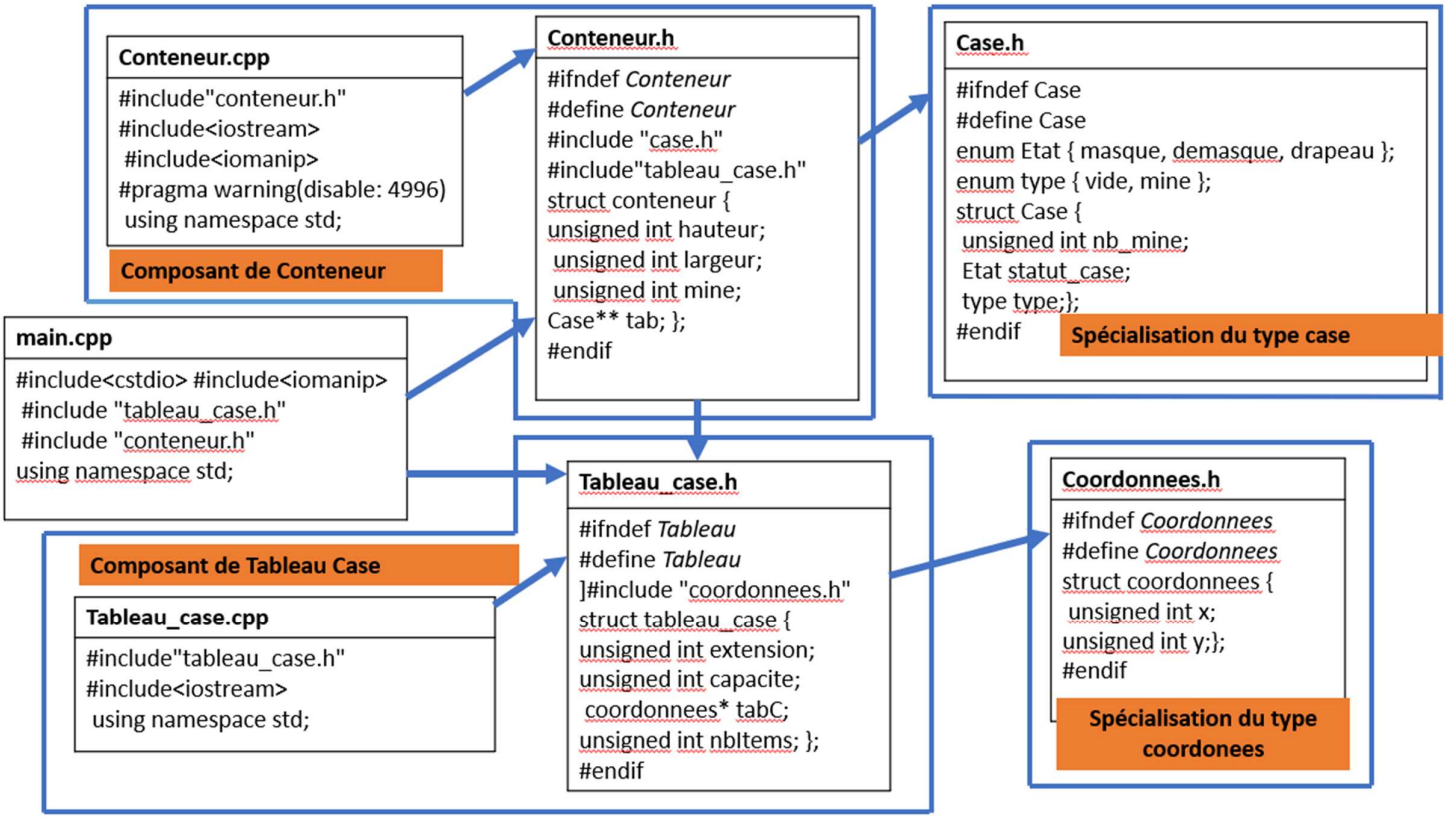
Pour la commande numéro 2, Le fichier texte entré correspond au numéro de la commande suivi du nombre de lignes, de colonnes, de mines avec leurs positions correspondantes ainsi que l'historique de coup (les coups qui ont étés effectués). L'objectif était de retourner la grille correspondant au problème entrée.

Pour la commande numéro 3, Le fichier texte entré correspond au numéro de la commande suivi d'un problème et un historique de coup. L'objectif était de retourner un message pour savoir si la partie est gagnée ou non. Une partie est gagnée lorsque toutes les cases ne contenant pas de mines sont démasqués et qu'aucune case contenant des mines sont démasqués.

Pour la commande numéro 4, Le fichier texte entré correspond au numéro de la commande suivi d'un problème et un historique de coup. L'objectif était de retourner un message pour savoir si la partie est perdue ou non. Une partie est perdue dès lors qu'une case contenant une mine est démasquée ou qu'une case ne contenant pas de mine est marqué.

Pour la commande numéro 5, Le fichier texte entré correspond au numéro de la commande suivi du nombre de lignes, de colonne ainsi que la grille correspondante. L'objectif était de retourner un coup possible à effectuer sur la grille entrée. Un coup possible est une case masquée qui n'a donc jamais été joué (marqué ou démasqué).

Graphe de dépendances des fichiers sources



Les Jeux d'essai mis au point

In1 : 1 4 6 8

Out1 : 4 6 8 8 10 11 17 3 4 7 13

In2 : 2 4 6 6 23 10 3 15 8 20 3 D19 M2 D12

Out2 :

```
4 6
 ___ ___ ___ ___ ___ ___
| . | . | x | m | . | . |
 ___ ___ ___ ___ ___ ___
| . | . | m | . | m | . |
 ___ ___ ___ ___ ___ ___
|   | 2 | 3 | m | . | . |
 ___ ___ ___ ___ ___ ___
|   | 1 | m | . | . | m |
 ___ ___ ___ ___ ___ ___
```

In3 : 3 4 6 6 23 10 3 15 8 20 3 D19 M2 D12

Out3 : game not won

In4 : 4 4 6 6 23 10 3 15 8 20 3 D19 M2 D12

Out4 : game lost

In5 :

```
5 5 7
 ___ ___ ___ ___ ___ ___ ___
| . | . | . | x | 2 |   |   |
 ___ ___ ___ ___ ___ ___ ___
| . | . | . | . | 2 | 1 | 1 |
 ___ ___ ___ ___ ___ ___ ___
| . | . | . | . | 2 | 2 | . |
 ___ ___ ___ ___ ___ ___ ___
| . | . | . | 2 | 1 | . | 3 |
 ___ ___ ___ ___ ___ ___ ___
| . | . | . | 2 | 1 | . | . |
 ___ ___ ___ ___ ___ ___ ___
```

Out5 : D10

Organisation des tests

Organisations des tests :

Les tests ont été effectué sur l'invite de commande généré par Visual Studio 2019. Nous exécutions le fichier d'entrée de la commande sur lequel nous travaillions et nous vérifiions si le résultat renvoyé était cohérant avec le résultat attendu. S'il nous semblait que le résultat était bel et bien cohérent, alors nous passions à la commande suivante.

Lorsqu'il y avait une erreur cependant, nous utilisions bien souvent des cout que nous mettions à chaque endroit où il pouvait potentiellement y avoir une erreur pour voir plus précisément ce que faisait le programme. Nous avons surtout souvent mis des cout (''IF'') pour des if par exemple ou des cout (''FOR'') pour des for pour ainsi voir combien de fois le programme exécutait les boucles ou les conditions. Nous utilisions également Paint, pour pouvoir faire un dessin représentant une grille du démineur en simulant ce que faisait notre fonction. Et si l'erreur n'était pas trouvée par ce moyen, nous imaginions ligne par ligne l'exécution du programme pour comprendre à quel moment il ne faisait pas ce qui était attendu.

Bilan de validation des Sprints :

La plupart du temps, nous trouvions assez rapidement la structure du programme et le moyen de réussir à obtenir ce que l'on souhaitait, mais nous oubliions un détail ou commettions une erreur discrète qui nous prenait parfois beaucoup de temps à trouver et à régler. Nous avons passé la majeure partie de notre temps à chercher où nous nous étions trompés en utilisant les méthodes détaillées plus tôt. Un exemple à cela est lors de la réalisation du code de la commande numéro 2, nous avions utilisés des « unsigned int » dans une boucle for dans la fonction qui permettait de vidé les cases, or à un moment donné la position d'une case devenait négative et donc la boucle ne s'exécutait pas. Pour cette même fonction, nous avions plus tôt penser à faire une fonction récursive (fonction qui s'appelle elle-même) , mais cela n'a pas abouti.

La plupart du temps, lorsque nous comparions les textes affichés et attendus, le résultat était le bon, car les erreurs étaient assez visibles lors des essais effectués avant.

Bilan du projet

Difficultés rencontrées :

Durant la réalisation de ce projet plusieurs difficultés ont été rencontrées, la plus grande était l'incompréhension du sujet initialement. Car nous avions commencé le projet assez tôt pour pouvoir anticiper les révisions des DST lors des vacances de Noël. C'est pour cela qu'au début nous avions programmer le démineur dans un while , nous pensions qu'il fallait faire un jeu interactif, ce qui était une erreur. Nous nous étions précipitées car l'analyse du sujet avait été bâclé.

Ensuite nous avons mis du temps à comprendre le sujet et ce que faisait chaque commande. Les fichiers texte « in » et « out » nous ont ensuite, grandement aidées.

D'autre part, il y a deux commandes particulières qui nous ont bloqués. La plus difficile, la commande 5, nous avons passé pas mal de temps pour arriver à lire la grille qui était fournis en entrée. Parce qu'il y avait un décalage lorsque qu'on scannait la grille 5 éléments par 5, l'indice de l'élément de chaque case changeait à chaque nouvelle ligne.

Enfin, la commande 2 a été moins difficile, mais nous avons mis du temps pour élaborer la fonction qui permet de vider toutes les cases vides lorsque l'on démasque une case, nous avons eu une erreur pendant la création de notre tableau de case vide qui devait être vidé. Pour éviter les doublons, nous avons vérifié avant chaque ajout de case qu'elle n'était pas déjà dans notre tableau et initialement, nous avons essayé de comparer le tableau avec lui-même et deux indices différents. Cela n'a pas marché donc nous avons opté pour une comparaisons de valeurs dans notre tableau. Nous avons aussi eu une autre erreur subtile, notamment à cause de l'erreur du « unsigned int » cité au-dessus.

Ce qui est réussi :

Ce projet est un très bon moyen de progresser en langage C++, d'assimiler la méthode de la compilation séparée et de réviser indirectement pour le DST de SDA. il nous fait utiliser tous types de fonctions et d'algorithmes et nous en fait découvrir des bien utiles. Il nous fais également travailler la documentation des prototypes ainsi que celle des fonctions. Nous avons gagné en aisance de programmation C++ grâce à ce projet, car nous avons passé beaucoup de temps à coder. Par ailleurs, on trouve que ce projet a été plus plaisant que celui de la première période, de par le choix du sujet.

Ce qui peut être amélioré :

Les fichiers « in » et « out » auraient pu être donnés plus tôt afin de mieux comprendre le résultat de chaque commande. Il n'y a pas eu d'autres réels problèmes rencontrées.

Conclusion :

Ce projet était intéressant et riche la familiarisation avec le langage C++, qui se rapproche de très près du langage C. Il a été utile pour les révisions des DST de janvier. Le bilan de ce projet est plutôt positif car la charge de travail qu'il apportait était surmontable et il nous a sûrement beaucoup aidé à progresser en programmation.

Sources :

Le cours d'initiation au développement, openclassroom.

Annexe

```c
#ifndef _Case_
#define _Case_

/**
 * @file case.h
 * Projet Démineur
 *  @author Lenouvel Louis et Gabriel Esteves
 * @version 23/11/2021
 * @brief Composant de case sur un conteneur
 */

/** @brief type Etat pour chaque état possible de la case
 */
enum  Etat { masque, demasque, drapeau };

/** @brief Type type pour chaque type de case différent
 */
enum type { vide, mine };


/** @brief type Case
* contenant le nombre de mine
* avec le statut de la case et son type
 */
struct Case {
	unsigned int nb_mine;	//nombre de mine présent à côté de la case
	Etat statut_case;	//etat de la case
	type type;	//type de la case
};
#endif

#ifndef _Conteneur_
#define _Conteneur_

/**
 * @file conteneur.h
 * Projet Démineur
 *  @author Lenouvel Louis et Gabriel Esteves
 * @version 23/11/2021
 * @brief Composant d'un conteneur de case
 */

#include "case.h"
#include"tableau_case.h"

/** @brief Conteneur de case en deux dimensions
 *  de capacité définit par largeur et hauteur
 * contenant un nombre de mine définit par l'utilisateur
 */
struct conteneur {
	unsigned int hauteur;	//hauteur du conteneur
	unsigned int largeur;	//largeur du conteneur
	unsigned int mine;	//nombre de mine du conteneur
	Case** tab;	//conteneur à deux dimensions
};
```

```
/**
 * @brief Initialise un conteneur de case
 * @param[out] c : le conteneur de case
 * @param [in] hauteur : hauteur du conteneur
 * @param [in] largeur: largeur du conteneur
 */
void initialiser(conteneur& c, unsigned int hauteur, unsigned int largeur);

/**
 * @brief Ajoute les mines au conteneur de case
 * @param[out] c : le conteneur de case
 * @param [in] nb : le nombre de mine à placer
 */
void ajoutMine(conteneur& c, unsigned int nb);

/**
 * @brief melange les mines au conteneur de case de manière aléatoire
 * @param[out] c : le conteneur de case
 * @param [in] nb : le nombre de mine à placer
 * @param [out] coo : coordonnés avec abscisse et ordonnée
 */
void melangerMine(conteneur& c, unsigned int nb, coordonnees& coo);

/**
 * @brief ajoute les nombres de mines dans les cases
 * @param[out] c : le conteneur de case
 * @param [in] x : hauteur de la case
 * @param [in] x : largeur de la case
 */
void ajouter_nbMine(conteneur& c, int x, int y);

/**
 * @brief cherche toutes les mines mines dans le conteneur c
 * @param[out] c : le conteneur de case
 */
void chercher_mine(conteneur& c);

/**
 * @brief convertit une position avec l'abscisse et l'ordonnée
 * @param[in] c : le conteneur de case
 * @param[in] x : hauteur de la case
 * @param[in] y : largeur de la case
 * @param[return] position : position convertis
 */
int recup(const conteneur& c, unsigned int x, unsigned int y);

/**
 * @brief affiche le conteneur de case
 * @param[in] c : le conteneur de case
 */
void afficher(const conteneur& c);

/**
 * @brief détermine si la partie est perdu
 * @param[in] c : le conteneur de case
 */
bool perdu(const conteneur& c);

/**
```

```
* @brief détermine si la partie est gagné


* @param[in] c : le conteneur de case
 */
bool gagne(const conteneur& c);

/**
 * @brief créer la map avec la commande 5 à partir d'un texte représentant un conteneur
 * @param[out] c : le conteneur de case
 */
void creer_map(conteneur& c);

/*
 * @brief démasque toutes les mines du conteneur c si le joueur a perdu
 * @param[out] c : le conteneur de case
 */
void afficher_amine(conteneur& c);

/*
 * @brief executes les différents commandes rentrées par l'utilisateur
 * @param[out] c : le conteneur de case
 * @param[out] coo : coordonnées utilisés pour les différents appelles de fonction
 * @param[out] t : tableau de cases démasquées
 */
void jouer(conteneur& c, coordonnees& coo, tableau_case& t);

/**
 * @brief Ajoute les mines au conteneur de case
 * @param[out] c : le conteneur de case
 * @param [in] positon : position à convertir en abscisse et ordonnée en fonction de la
hauteur et la largeur du conteneur c
 * @param [out] coo : coordonnées avec l'abscisse et l'ordonnée
 */
void cPosition(conteneur& c, unsigned int position, coordonnees& coo);

/**
 * @brief place les mines au conteneur de case en fonction des positions entrées
 * @param[out] c : le conteneur de case
 * @param [in] nb : le nombre de mine à placer
 * @param [out] coo : coordonnés avec abscisse et ordonnée
 */
void placerMine(conteneur& c, unsigned int position, coordonnees& coo);

/**
 * @brief ajoute toutes les coordonnées des cases vides dans le tableau de case
 * @param[in] c : le conteneur de case
 * @param[out] t : tableau qui contient les cases vide
 * @param[out] cased : les coordonnées qui vont être utilisé pour l'implémentation dans
le tableau de case
 */
void case_vide(conteneur& c, tableau_case& t, coordonnees& cased);

/*
 * @brief démasque toutes les cases vides présentent dans le tableau
 * @param[out] c : le conteneur de case
 * @param[in] t : le tableau de case
 */
void vider(conteneur& c, tableau_case& t);
```

```
/*
 * @brief executes les différents coup rentrées par l'utilisateur
 * @param[out] c : le conteneur de case


 * @param[in] action1 : action de l'utilisateur (masqué ou démasqué)
 * @param[in] position : case choisit pour faire l'action
 * @param[in] coo : coordonnées qui sont utilisés pour les vérifications de la fonction
 * @param[in] t : tableau de cases à démasquées
 */
void coup(conteneur& c, char action1, unsigned int position, coordonnees& coo, ta-
bleau_case& t);

/**
 * @brief stock les cases masqués dans un tableau
 * @param[in] c : le conteneur de case
 * @param[out] t : tableau de case masqué
 */
void case_masque(const conteneur& c, tableau_case& t);

/**
 * @brief propose un coup aléatoire possible sur une case masqué
 * @param[out] c : le conteneur de case
 * @param[in] t : tableau qui contient les cases masqués
 */
void coupAleatoire(const conteneur& c, tableau_case& t);

                                        #endif

#ifndef _Coordonnees_
#define _Coordonnees_

/**
 * @file coordonnees.h
 * Projet Démineur
 *  @author Lenouvel Louis et Gabriel Esteves
 * @version 23/11/2021
 * @brief Composant d'une coordonnée
 */


/**
 * @brief Type coordonnées
 * invariant : la position doit être valide
*/
struct coordonnees {
       unsigned int x;            //abscisse de la coordonnée
       unsigned int y;            //ordonnée de la coordonnée
};

#endif

#ifndef _Tableau_
#define _Tableau_

/**
 * @file tableau_case.h
 * Projet Démineur
 *  @author Lenouvel Louis et Gabriel Esteves
 * @version 23/11/2021
 * @brief Composant d'un tableau de case
```

```
*/

#include "coordonnees.h"

/** @brief tableau de case alloués en mémoire dynamique


 *  de capacité extensible suivant un pas d'extension
 */
struct tableau_case {
	unsigned int extension;    //extension du tableau
	unsigned int capacite;     //capacité du tableau
	coordonnees* tabC;  //tableau de case
	unsigned int nbItems;      //nombre d'items du tableau
};

/**
 * @brief Allocation dynamique de la mémoire du tableau de case
 * @param[out] t : le tableau de case
 * @param [in] i : la position où l'on veut écrire
 */
void allocation(tableau_case& t, unsigned int i);

/**
 * @brief vérifie que l'élément que l'on veut ajouter n'est pas déjà dans le tableau de
case
 * @param[in] t : tableau qui contient les cases masqués
 * @param[in] cased : les coordonnées qui vont être utilisé pour la vérification
 * @param[return] bool : true ou false
 */
bool vérifier(tableau_case& t, coordonnees& cop);


#endif
/**
 * @file tableau_case.cpp
 * Projet Démineur
 * @author Lenouvel Louis et Gabriel Esteves
 * @version 23/11/2021
 * @brief Composant d'un tableau de case de capacité extensible
 */

#include"tableau_case.h"
#include<iostream>
using namespace std;

/**
 * @brief Allocation dynamique de la mémoire du tableau de case
 * @param[out] t : le tableau de case
 * @param [in] i : la position où l'on veut écrire
 */
void allocation(tableau_case& t, unsigned int i) {
	if (t.capacite <= i) {
		int newTaille = (i + 1) * t.extension;          //on initialise une nou-
velle taille
		coordonnees* newT = new coordonnees[newTaille];       //on initialise un
nouveau avec la nouvelle taille
		for (int j = 0; j < t.capacite; ++j) {
			newT[j] = t.tabC[j];       //on recopie chaque élément de l'ancien
tableau dans le nouveau
```

```
			}
			delete[] t.tabC;		//on supprime l'ancien tableau
			t.tabC = newT;			//on initialise le tableau à partir du nouveau
			t.capacite = newTaille;		//on initalise la taille du tableau
avec la nouvelle taille
		}
}



/**
 * @brief vérifie que l'élément que l'on veut ajouter n'est pas déjà dans le tableau de
case
 * @param[in] t : tableau qui contient les cases masqués
 * @param[in] cased : les coordonnées qui vont être utilisé pour la vérification
 * @param[return] bool : true ou false
 */
bool vérifier(tableau_case& t, coordonnees& cop) {
		for (int l = 0; l <= t.nbItems; l++) {
			if (cop.x == t.tabC[l].x && cop.y == t.tabC[l].y) {		//on cherche
si l'élément est déjà dans le tableau
				return true;
			}
		}
		return false;
}

/**
 * @file conteneur.cpp
 * Projet Démineur
 * @author Lenouvel Louis et Gabriel Esteves
 * @version 23/11/2021
 * @brief Composant de conteneur de case
 */


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

/**
 * @brief Initialise un conteneur de case
 * @param[out] c : le conteneur de case
 * @param [in] hauteur : hauteur du conteneur
 * @param [in] largeur: largeur du conteneur
 */
void initialiser(conteneur& c, unsigned int hauteur, unsigned int largeur) {
		c.hauteur = hauteur;
		c.largeur = largeur;
		c.tab = new Case * [hauteur];		//initialisation du conteneur
		for (int i = 0; i < hauteur; i++) {
			c.tab[i] = new Case[largeur]; //dans chaque élément on initialise un ta-
bleau pour obtenir un tableau à deux dimensions
			for (int j = 0; j < largeur; j++) {
				c.tab[i][j].statut_case = masque; //on initialise par défaut les
cases comme masqués
```

```
                        c.tab[i][j].type = vide;    //on initialise par défaut les cases
                comme vide
                        c.tab[i][j].nb_mine = 0;    //on initialise par défaut les cases
comme n'ayant pas de mine à coté d'elle
                }
        }
}

/**
 * @brief Ajoute les mines au conteneur de case


* @param[out] c : le conteneur de case
 * @param [in] nb : le nombre de mine à placer
 */
void ajoutMine(conteneur& c, unsigned int nb) {
        int i = 0, h = 0, j = 0, taille=c.hauteur*c.largeur;
        assert(nb <= taille);
        while (i < nb) { //on ajoute le nombre de mine demandé
                if (h > c.largeur) { //si nous avons dépassons la largeur deu tableau
alors nous passons à la ligne suivante et nous revenons au premier élément
                        j++;
                        h = 0;
                }
                if (c.tab[j][h].statut_case == masque) {
                        c.tab[j][h].type = mine;    //on ajoute une mine
                        i++;
                }
                h++;
        }
}

/**
 * @brief melange les mines au conteneur de case de manière aléatoire
 * @param[out] c : le conteneur de case
 * @param [in] nb : le nombre de mine à placer
 * @param [out] coo : coordonnés avec abscisse et ordonnée
 */
void melangerMine(conteneur& c, unsigned int nb, coordonnees& coo) {
        unsigned int début = 0, h = 0, i = 0, j = 0, n = c.largeur * c.hauteur;
        cout << c.hauteur << " " << c.largeur << " " << nb << " ";
        while(i<nb) { //on mélange toutes les mines dans le conteneur
                if (h > c.largeur) { //si nous avons dépassons la largeur deu tableau
alors nous passons à la ligne suivante et nous revenons au premier élément
                        j++;
                        h = 0;
                }
                début = rand() % n;         //on cherche une nouvelle position
                cPosition(c, début, coo);         //nous récupérons les coodonnées
                if(c.tab[coo.x][coo.y].type!=mine){             //on vérifie d'abord que
ce n'est pas une mine
                        c.tab[coo.x][coo.y].type = c.tab[j][h].type;    //sinon on attribue
la mine à une nouvelle case dans le tableau
                        c.tab[j][h].type = vide;    //et l'ancienne case devient de type
vide pour éviter les doublons
                        h++;
                        i++;
                        cout << début << " ";
                }
        }
```

```
}



/**
 * @brief ajoute les nombres de mines dans les cases
 * @param[out] c : le conteneur de case
 * @param [in] x : hauteur de la case
 * @param [in] x : largeur de la case
 */
void ajouter_nbMine(conteneur& c, int x, int y) {


	for (int i = x - 1; i <= x + 1; i++) {
		for (int j = y - 1; j <= y + 1; j++) {
			if (i >= 0 && i <= c.hauteur - 1 && j >= 0 && j <= c.largeur - 1)
{ // si la case est compris dans le conteneur c
				c.tab[i][j].nb_mine++; // augmenter le nombre de mine
			}
		}

	}
}

/**
 * @brief cherche toutes les mines mines dans le conteneur c
 * @param[out] c : le conteneur de case
 */
void chercher_mine(conteneur& c) {
	for (unsigned int i = 0; i < c.hauteur; i++) {
		for (unsigned int j = 0; j < c.largeur; j++) {
			if (c.tab[i][j].type == mine) {
				ajouter_nbMine(c, i, j); // ajouter les nombres de mines
autour de chaque mine trouvé
			}
		}
	}
}

/**
 * @brief convertis une position avec l'abscisse et l'ordonnée
 * @param[out] c : le conteneur de case
 * @param[in] x : hauteur de la case
 * @param[in] y : largeur de la case
 * @param[return] position : position convertis
 */
int recup(const conteneur& c, unsigned int x, unsigned int y) {
	unsigned int position;
	x = x * c.largeur;		//nous recalculons son abscisse
	position = x + y;		//nous recalculons sa position
	return position;
}

/**
 * @brief affiche le conteneur de case
 * @param[in] c : le conteneur de case
 */
void afficher(const conteneur& c) {
	int compteur = 0, i = 0;
```

```
	while (i < c.hauteur) {		//on affiche toutes les lignes de notre conte-
	neur
		if (compteur % 2 == 0) {		//une fois sur deux nous affichons une
ligne de "___" grâce à un compteur
			for (int j = 0; j < c.largeur; j++) {		//nous affichons la
ligne de "___"
				cout << " "<< "___";
			}
			++compteur;
			cout << endl;
		}
		else {
			for (int j = 0; j < c.largeur; j++) {		//nous affichons
chaque élément de la ligne de notre conteneur



				if (c.tab[i][j].statut_case == masque) {
					if (j == 0) {
						cout << setiosflags(ios::right) << setw(1) <<
"| " << ".";	//on affiche un "." pour chaque case masquée
					}
					else
						cout << setiosflags(ios::right) << setw(1) <<
" | " << ".";
				}
				else if (c.tab[i][j].statut_case == drapeau) {
					if (j == 0) {
						cout << setiosflags(ios::right) << setw(1) <<
"| " << "x";	//on affiche un "X" pour chaque case  marquée d'un drapeau
					}
					else
						cout << setiosflags(ios::right) << setw(1) <<
" | " << "x";
				}
				else if (c.tab[i][j].statut_case == demasque) {
					if (c.tab[i][j].type == vide) {
						if (c.tab[i][j].nb_mine == 0) {
							if (j == 0) {
								cout << setiosflags(ios::right)
<< setw(1) << "| " << " "; //on affiche un " " pour chaque case démasquée et vide
							}
							else
								cout << setiosflags(ios::right)
<< setw(1) << " | " << " ";
						}
						else {
							if (j == 0) {
								cout << setiosflags(ios::right)
<< setw(1) << "| " << c.tab[i][j].nb_mine;		//on affiche un chiffre pour
chaque case démasquée s'il y a une mine à coté
							}
							else
								cout << setiosflags(ios::right)
<< setw(1) << " | " << c.tab[i][j].nb_mine;
						}
					}
					if (c.tab[i][j].type == mine) {
						if (j == 0) {
```

```
                                                cout << setiosflags(ios::right) <<
                                                setw(1) << "| " << "m";           //on
                                                affiche un "m" pour chaque case démas-
                                                quée qui est une mine
                                        }
                                        else
                                                cout << setiosflags(ios::right) <<
setw(1) << " | " << "m";
                                }
                        }

                }
                ++compteur;
                i++;
                cout << " |" << endl;
        }
    }



        for (int j = 0; j < c.largeur; j++) {           //on affiche une dernière
    ligne de "___"
        cout << " " << "___";
    }
    cout << endl;
}

/**
 * @brief détermine si la partie est perdu
 * @param[in] c : le conteneur de case
 */
bool perdu(const conteneur& c) {
    for (int i = 0; i < c.hauteur; i++) {
        for (int j = 0; j < c.largeur; j++) {
            if (c.tab[i][j].statut_case == demasque && c.tab[i][j].type ==
mine) {             //si une case est de type mine et démasquée alors l'utilisateur a
perdu
                    return true;
            }
            else if (c.tab[i][j].statut_case == drapeau && c.tab[i][j].type ==
vide) {             //si une case est de type vide et marquée par un drapeau alors
l'utilisateur a perdu
                    return true;
            }
        }
    }
    return false;
}

/**
 * @brief détermine si la partie est gagné
 * @param[in] c : le conteneur de case
 */
bool gagne(const conteneur& c) {
    if (perdu(c) == false) {   //si la partie est perdu alors l'utilisateur n'a pas
gagné
        return false;
    }
    else {
```

```
		for (int i = 0; i < c.hauteur; i++) {
			for (int j = 0; j < c.largeur; j++) {
				if (c.tab[i][j].statut_case == masque && c.tab[i][j].type
== vide) {	//si une case est de type vide et masquée alors l'utilisateur n'a tou-
jours pas gagnée
					return false;
				}
			}
		}
	}
}

/**
 * @brief créer la map avec la commande 5 à partir d'un texte représentant un conteneur
 * @param[out] c : le conteneur de case
 */
void creer_map(conteneur& c) {
	unsigned int hauteur = 0, largeur = 0, compteur = 0, nb = 0, dep = 2;
	const int TAILLE = 5 * c.largeur;
	cin >> std::ws; // ignore le \n qui suit les 3 entiers


	cin.ignore(TAILLE, '\n'); // ignore la ligne de séparation " ___ ___ ..."
	char valeur[5];
	while (hauteur < c.hauteur) {
		cin.get(valeur, 5);  //on lit 5 éléments par 5 éléments
		if (valeur[2] == '|') {			//si on lit "|" au deuxième élément de
ce que l'on a scanné alors on modifie le lieu de lecture car sinon on subit le décalage
après chaque ligne
			dep = 0;
		}
		if (dep < 4) {			//on ne lit que si le lieu de lecture est in-
férieur à 4 car à cause du décalage le lieu de lecture ne fonctionne qu'avec 0,1,2 et 3
			switch (valeur[dep]) {
			case ' ':
				c.tab[hauteur][largeur].statut_case = demasque;
				c.tab[hauteur][largeur].type = vide;
				largeur++;
				break;
			case '.':
				c.tab[hauteur][largeur].statut_case = masque;
				largeur++;
				break;
			case 'x':
				c.tab[hauteur][largeur].statut_case = drapeau;
				largeur++;
				break;
			case '_':
				break;
			default:			//par défaut si aucun des éléments lu ne cor-
respond à une condition c'est forcément un chiffre donc nous le convertissons
				sscanf(&valeur[dep], "%d", &nb);		//nous convertis-
sons donc l'éléments en int
				c.tab[hauteur][largeur].nb_mine = nb;
				c.tab[hauteur][largeur].statut_case = demasque;
				largeur++;
				break;
			}
		}
```

```
			if (largeur >= c.largeur) {
				hauteur++;
				largeur = 0;
				cin.ignore(TAILLE, '\n'); // ignore le \n qui fini la ligne
				cin.ignore(TAILLE, '\n'); // ignore la ligne de séparation " ___
___ ..."
			}
		}
}

/*
 * @brief démasque toutes les mines du conteneur c si le joueur a perdu
 * @param[out] c : le conteneur de case
 */
void afficher_amine(conteneur& c) {
		for (int i = 0; i < c.hauteur; i++) {
			for (int j = 0; j < c.largeur; j++) {
				if (c.tab[i][j].type == mine) {		//dès que nous trouvons
une mine nous la démasquons
					c.tab[i][j].statut_case = demasque;
				}
			}
		}


}


/*
 * @brief executes les différents commandes rentrées par l'utilisateur
 * @param[out] c : le conteneur de case
 * @param[out] coo : coordonnées utilisés pour les différents appelles de fonction
 * @param[out] t : tableau de cases démasquées
 */
void jouer(conteneur& c, coordonnees& coo, tableau_case& t) {
		unsigned int commande, hauteur, largeur, mine, position_mine, nb_coup, posi-
tion_action;
		char coup_joueur[30];
		char action;
		cin >> commande;			//commande rentrée par l'utilisateur
		switch (commande) {		//on regarde quelle commande a été rentrée
		case 1:					//On produit un problème à partir du nombre de ligne,
de collones et de mines
			cin >> hauteur >> largeur >> mine;
			c.mine = mine;
			initialiser(c, hauteur, largeur);
			ajoutMine(c, c.mine);
			melangerMine(c, c.mine, coo);
			chercher_mine(c);
			break;
		case 2:					//On produit une grille  à partir d'un problème et
d'un historique de coups
			cin >> hauteur >> largeur >> mine;
			initialiser(c, hauteur, largeur);
			for (unsigned int i = 0; i < mine; i++) {
				cin >> position_mine;
				placerMine(c, position_mine, coo);
			}
			chercher_mine(c);
```

```
                cin >> nb_coup;
                for (unsigned int i = 0; i < nb_coup; i++) {
                        cin >> coup_joueur;
                        sscanf(coup_joueur, "%c %d", &action, &position_action);
                        coup(c, action, position_action, coo, t);
                }
                cout << hauteur << " " << largeur << endl;
                afficher(c);
                break;
        case 3:             //On détermine si la partie est gagnée à partir d'un pro-
blème et d'un historique de coups
                cin >> hauteur >> largeur >> mine;
                initialiser(c, hauteur, largeur);
                for (unsigned int i = 0; i < mine; i++) {
                        cin >> position_mine;
                        placerMine(c, position_mine, coo);
                }
                chercher_mine(c);
                cin >> nb_coup;
                for (unsigned int i = 0; i < nb_coup; i++) {
                        cin >> coup_joueur;
                        sscanf(coup_joueur, "%c %d", &action, &position_action);
                        coup(c, action, position_action, coo, t);
                }
                if (gagne(c) == 0) {
                        cout << "game not won";


                }
                else
                        cout << "Game won";
                break;
        case 4:             //On détermine si la partie est perdu à partir d'un pro-
blème et d'un historique de coups
                cin >> hauteur >> largeur >> mine;
                initialiser(c, hauteur, largeur);
                for (unsigned int i = 0; i < mine; i++) {
                        cin >> position_mine;
                        placerMine(c, position_mine, coo);
                }
                cin >> nb_coup;
                for (unsigned int i = 0; i < nb_coup; i++) {
                        cin >> coup_joueur;
                        sscanf(coup_joueur, "%c %d", &action, &position_action);
                        coup(c, action, position_action, coo, t);
                }
                chercher_mine(c);
                if (perdu(c) == 1) {
                        cout << "game lost";
                }
                else
                        cout << "game not lost";
                break;
        case 5:             //On produit un nouveau coup à partir d'une grille donnée
                cin >> hauteur >> largeur;
                initialiser(c, hauteur, largeur);
                creer_map(c);
                case_masque(c, t);
                coupAleatoire(c, t);
```

```
			break;
		default:	//la commande n'existe pas
			cout << "commande non reconnue" << endl;
			break;
		}
}


/**
 * @brief Ajoute les mines au conteneur de case
 * @param[out] c : le conteneur de case
 * @param [in] positon : position à convertir en abscisse et ordonnée en fonction de la
hauteur et la largeur du conteneur c
 * @param [out] coo : coordonnées avec l'abscisse et l'ordonnée
 */
void cPosition(conteneur& c, unsigned int position, coordonnees& coo) {
		coo.x = (position - (position % c.largeur)) / c.largeur; // abscisse de la posi-
tion
		coo.y = position % c.largeur; // ordonnée de la position
}


/**
 * @brief place les mines au conteneur de case en fonction des positions entrées
 * @param[out] c : le conteneur de case
 * @param [in] nb : le nombre de mine à placer
 * @param [out] coo : coordonnés avec abscisse et ordonnée
 */
void placerMine(conteneur& c, unsigned int position, coordonnees& coo) {


		cPosition(c, position, coo); // coordonnées convertis pour les placer dans le
tableau
		c.tab[coo.x][coo.y].type = mine; // changement du type de la case (mine) en
fonction des coordonnées
}

/**
 * @brief ajoute toutes les coordonnées des cases vides dans le tableau de case
 * @param[in] c : le conteneur de case
 * @param[out] t : tableau qui contient les cases vide
 * @param[out] cased : les coordonnées qui vont être utilisé pour l'implémentation dans
le tableau de case
 */
void case_vide(conteneur& c, tableau_case& t, coordonnees& cased) {
		int x, y;
		t.nbItems = 1;
		t.capacite = 1;
		t.extension = 2;
		coordonnees cop;
		t.tabC = new coordonnees[t.capacite];		//on initialise le tableau
		unsigned int i = 0;
		t.tabC[i].x = cased.x;			//on implémente un premier élément qui est la
case vide que l'on a démasqué
		t.tabC[i].y = cased.y;			//on implémente un premier élément qui est la
case vide que l'on a démasqué
		while (i <= t.nbItems) {		//on vérifie pour chaque case vide du tableau
			y = t.tabC[i].y;
			x = t.tabC[i].x;
			for (int j = x - 1; j <= x + 1; j++) {
```

```
                for (int h = y - 1; h <= y + 1; h++) {          //on parcourt
            toutes les cases autour d'elle y compris elle même
                        if (j >= 0 && j <= c.hauteur - 1 && h >= 0 && h <= c.lar-
geur - 1 && c.tab[j][h].type == vide && c.tab[j][h].nb_mine == 0) {        //on vérifie
si les coordonnées sont valide et si la case est bien une case vide sans chiffre
                                cop.x = j;          //alors on convertie le x trouvé
x de type coordonnées
                                cop.y = h;          //alors on convertie le y trouvé
y de type coordonnées
                                if (vérifier(t, cop) == false) {         //on vérifie
avant d'ajouter que l'élément n'est pas déjà dedans
                                        allocation(t, t.nbItems);         //on vérifie
que l'on dépasse pas la capacité du tableau
                                        t.tabC[t.nbItems].x = cop.x;
      //alors on ajoute les coordonnées trouvées dans notre tableau de case
                                        t.tabC[t.nbItems].y = cop.y;
      //alors on ajoute les coordonnées trouvées dans notre tableau de case
                                        t.nbItems++;        //on passe à l'élément
suivant
                                }
                        }
                }
            }
            i++;
      }
}

/*
 * @brief démasque toutes les cases vides présentent dans le tableau
 * @param[out] c : le conteneur de case
 * @param[in] t : le tableau de case


*/
void vider(conteneur& c, tableau_case& t) {
      unsigned int x, y;
      for (int i = 0; i < t.capacite; i++) {          //on parcourt toutes les cases
du tableau
            y = t.tabC[i].y;
            x = t.tabC[i].x;
            for (unsigned int j = x - 1; j <= x + 1; j++) {
                  for (unsigned int h = y - 1; h <= y + 1; h++) {       //on par-
court toutes les cases autour de l'élément du tableau y compris lui même
                        if (j >= 0 && j <= c.hauteur - 1 && h >= 0 && h <= c.lar-
geur - 1 && c.tab[j][h].type == vide) {        //on vérifie si les coordonnées sont
valide et si la case est bien une case vide
                                c.tab[j][h].statut_case = demasque;            //on
démasque la case
                        }
                  }
            }
      }
}
/*
 * @brief executes les différents coup rentrées par l'utilisateur
 * @param[out] c : le conteneur de case
 * @param[in] action1 : action de l'utilisateur (masqué ou démasqué)
 * @param[in] position : case choisit pour faire l'action
 * @param[in] coo : coordonnées qui sont utilisés pour les vérifications de la fonction
```

```
* @param[in] t : tableau de cases à démasquées
 */
void coup(conteneur& c, char action1, unsigned int position, coordonnees& coo, ta-
bleau_case& t) {
	unsigned int ligne, colonne;
	if (action1 == 'M') {
		cPosition(c, position, coo); // converti de la position en coordonnées x
et y
		if (c.tab[coo.x][coo.y].type == mine) {
			c.tab[coo.x][coo.y].statut_case = drapeau;
		}
		else if (c.tab[coo.x][coo.y].type == vide) { //si la case marqué est vide
alors on a perdu
			afficher_amine(c);		//donc nous affichons toutes les mines
			c.tab[coo.x][coo.y].statut_case = drapeau;
		}
	}
	else if (action1 == 'D') {
		cPosition(c, position, coo); // converti de la position en coordonnées x
et y
		c.tab[coo.x][coo.y].statut_case = demasque;
		if (c.tab[coo.x][coo.y].type == vide) {
			case_vide(c, t, coo);			//nous vidons toutes les cases
autour
			vider(c, t);
		}
		else if (c.tab[coo.x][coo.y].type == mine) { // si la case démasqué est
une mine alors on a perdu
			afficher_amine(c); // affichage de toutes les mines
		}
	}
	else
		cout << "Commande non reconnue" << endl;
}


/**
 * @brief stock les cases masqués dans un tableau
 * @param[in] c : le conteneur de case
 * @param[out] t : tableau de case masqué
 */
void case_masque(const conteneur& c, tableau_case& t) {
	unsigned int indice = 0;
	t.nbItems = 1;
	t.capacite = 1;
	t.extension = 2;
	t.tabC = new coordonnees[t.capacite];		//on initialise le tableau
	coordonnees cop;
	for (unsigned int i = 0; i < c.hauteur; i++) {
		for (unsigned int j = 0; j < c.largeur; j++) {
			if (i >= 0 && i <= c.hauteur - 1 && j >= 0 && j <= c.largeur - 1
&& c.tab[i][j].statut_case == masque) {
				cop.x = i;
				cop.y = j;
				allocation(t, t.nbItems);		//on vérifie que l'on dé-
passe pas la capacité du tableau
				t.tabC[indice].x = cop.x;		//alors on ajoute la case
dans le tableau
				t.tabC[indice].y = cop.y;
				indice++;
```

```
                        t.nbItems++;        //on augmente le nombre d'items
                    }
                }
        }
}

/**
 * @brief propose un coup aléatoire possible sur une case masqué
 * @param[out] c : le conteneur de case
 * @param[in] t : tableau qui contient les cases masqués
 */
void coupAleatoire(const conteneur& c, tableau_case& t) {
        unsigned int case_a, action;
        case_a = rand() % t.nbItems - 1;        //on choisit une case aléatoirement
dans le tableau de case masquées
        action = rand() % 2;       //on choisit une des deux actions possible aléatoire-
ment (1 = démasqué et 0 = masqué=)
        if (t.nbItems - 1 == 0) {
                cout << "Aucun coup possible" << endl;
        }
        else if (action == 1) {
                cout << 'D' << recup(c, t.tabC[case_a].x, t.tabC[case_a].y);
        }
        else if (action == 0) {
                cout << 'M' << recup(c, t.tabC[case_a].x, t.tabC[case_a].y);
        }
}
```

```
/**
 * @file main.cpp
 * Projet Démineur
 * @author Lenouvel Louis et Gabriel Esteves
 * @version 23/11/2021
 */


#include<cstdio>
#include<iomanip>
#include "tableau_case.h"
#include "conteneur.h"
using namespace std;



/* Test du jeu*/
int main() {
	srand(time(NULL));
	coordonnees coo;
	conteneur c;
	tableau_case t;
	jouer(c, coo, t);
}
```