

GASSMANN Clément, MALONZO Ryan
Groupe 104

SDA – PROJET

BOGGLE



5 Janvier 2021

Table des matières

Introduction du projet	2
Graphe de dépendances	3
Tests unitaires.....	4
Bilan du projet.....	5
Codes sources.....	6
main.cpp.....	6
exercices.h.....	7
exercices.cpp	8
utils.h.....	11
utils.cpp.....	12
Mot.h.....	14
ListeMots.h.....	14
ListeMots.cpp.....	16
ListeListesMots.h.....	17
ListeListesMots.cpp	18
PlateauMots.h	20
PlateauMots.cpp.....	21

Introduction du projet

Le projet de Structures des Données et Algorithmes fondamentaux consiste en la réalisation de six programmes indépendants qui permettent, ensemble, de jouer une partie complète de Boggle.

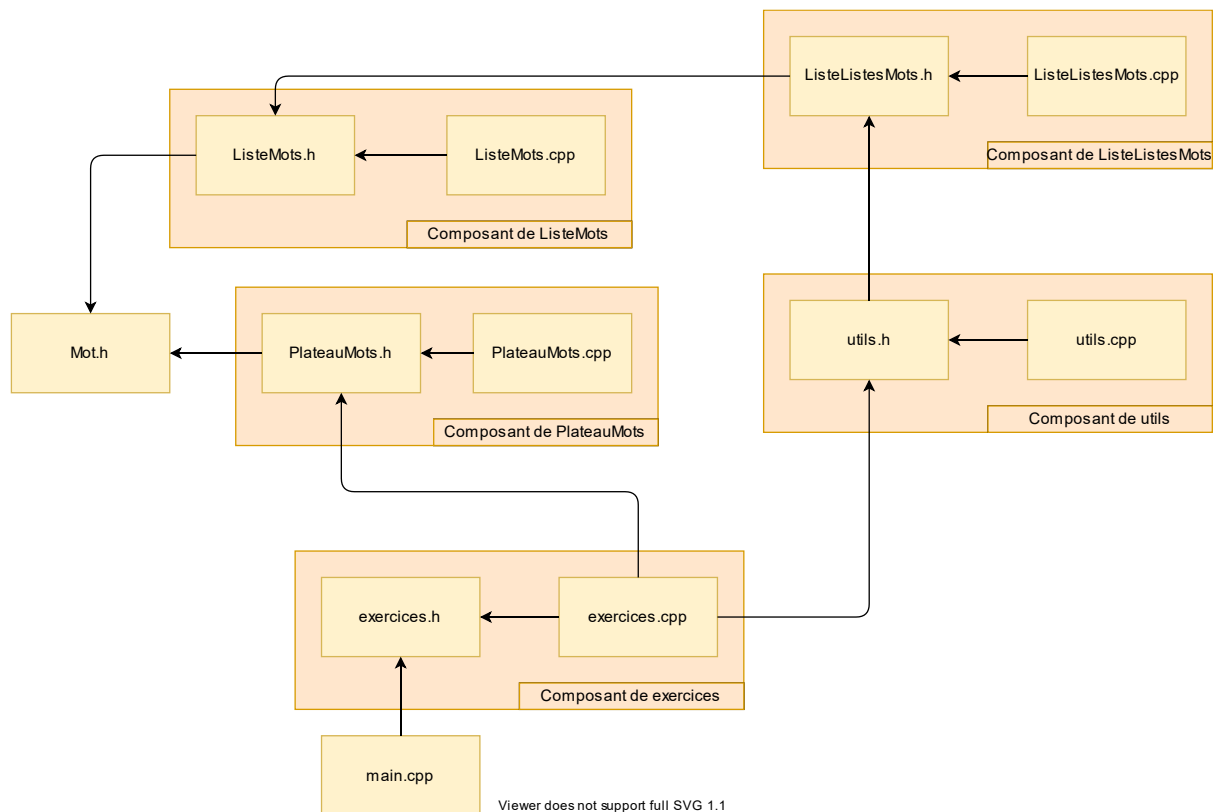
Le Boggle, jeu de lettres conçu par Alan Turoff, consiste en la recherche du maximum de mots de minimum 3 lettres qui peuvent être formés à partir de lettres disposées sur un plateau carré 4x4 et adjacentes horizontalement, verticalement ou en diagonale. Chaque lettre ou case du tableau doit être traversée une unique fois.

A la fin d'une période de temps impartie, les joueurs de la partie comparent les mots qu'ils ont trouvés ; sont retenus ceux qui sont uniques à chaque participant. On relève enfin la longueur de chaque mot afin d'en déduire le nombre de points correspondant, selon un barème établi. Le gagnant de la partie est le joueur possédant le plus grand nombre de points.

Les six programmes demandés sont réalisés à l'aide du langage C ou C++ et en se limitant aux structures de données vues en cours, adaptées si besoin.

L'évaluation des programmes porte principalement sur leur structuration et sur le caractère réutilisable des fonctions et structures développées.

Graphe de dépendances



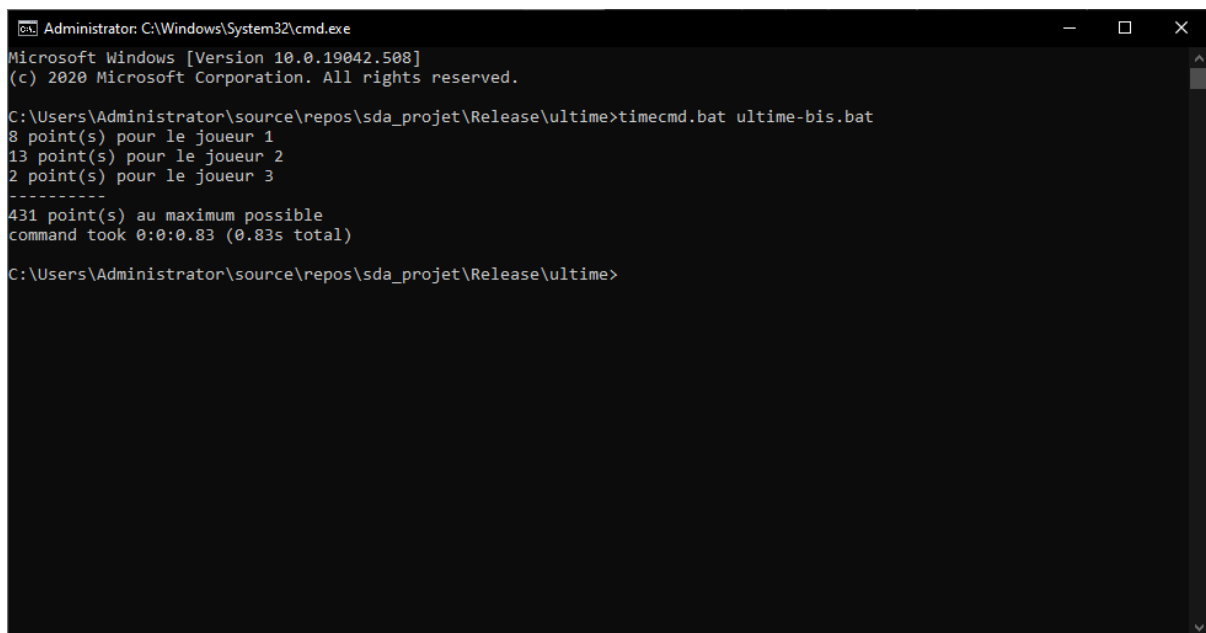
Graphe de dépendances. L'image est vectorielle : vous pouvez zoomer sur celle-ci en cas de besoin.

Tests unitaires

Les fichiers d'entrée fournis avec le sujet du projet ont été testés pour chacun de nos six programmes suivant les instructions relatives au `cmd`. Les fichiers produits en sortie ont ensuite été comparés avec ceux attendus à l'aide du site

<https://www.diffchecker.com/>.

A surtout été privilégié le test ultime disposé ultérieurement sur Moodle représentant une combinaison rigoureuse des six programmes à développer pour simuler une partie réelle de Boggle. Ce dernier renvoie les informations ci-dessous, conformes à celles attendues.

A screenshot of a Windows command prompt window titled "Administrator: C:\Windows\System32\cmd.exe". The window shows the output of a script execution. The text displayed is: "Microsoft Windows [Version 10.0.19042.508] (c) 2020 Microsoft Corporation. All rights reserved. C:\Users\Administrator\source\repos\sda_projet\Release\ultime>timecmd.bat ultime-bis.bat 8 point(s) pour le joueur 1 13 point(s) pour le joueur 2 2 point(s) pour le joueur 3 ----- 431 point(s) au maximum possible command took 0:0:0.83 (0.83s total) C:\Users\Administrator\source\repos\sda_projet\Release\ultime>".

```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.508]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Administrator\source\repos\sda_projet\Release\ultime>timecmd.bat ultime-bis.bat
8 point(s) pour le joueur 1
13 point(s) pour le joueur 2
2 point(s) pour le joueur 3
-----
431 point(s) au maximum possible
command took 0:0:0.83 (0.83s total)

C:\Users\Administrator\source\repos\sda_projet\Release\ultime>
```

Sortie de l'exécution du script ultime-bis.bat

Le temps d'exécution de ce script pour notre programme est de 0,83s.

Bilan du projet

Le projet nous a été intéressant en ce sens qu'en comparaison avec le projet d'IAP, celui-ci nous a permis de décomposer nos fonctions en différents fichiers, dans l'idée de développer des fonctions et structures flexibles et d'obtenir, à l'échelle de nos programmes, une meilleure lisibilité.

Les six programmes attendus ont été développés et validés à l'aide des fichiers fournis, le test ultime très particulièrement, et en utilisant la structure Liste étudiée en TD et TP adaptée pour les chaînes de caractères, qui est extensible et dont les opérations correspondent pleinement au contexte d'une partie de Boggle.

Le projet est décomposé en de nombreux fichiers d'en-têtes et sources et dont les dépendances sont clairement établies (cf. graphe des dépendances) : le fichier contenant les six exercices attendus `exercices.cpp` fait appel à `utils.h` et `PlateauMots.h` de manière directe, lesquels vont ensuite appeler les composants manquants en profondeur.

De plus, cette catégorisation des appels de code récurrents permet leur réutilisation de manière efficace et un gain d'espace considérable au sein du fichier.

Toutefois, certains aspects de notre réalisation pourraient sûrement gagner à être programmés différemment. Nous relevons notamment les points suivants.

- Le tri des mots d'une liste par ordre alphabétique, inclus dans la fonction qui permet de rendre une telle liste « canonique », effectue des comparaisons deux à deux. Celles-ci sont coûteuses en temps d'exécution et employer une autre méthode de tri permettrait d'en conserver.
- Les coordonnées adjacentes à une coordonnée donnée sont initialisées en brut dans la fonction de sous-recherche de l'exercice 6. Une meilleure implémentation de cette dernière et de la structure correspondant à la grille de lettres est certainement possible.

Codes sources

main.cpp

```
/**
 * @file main.cpp
 * @brief Interface d'exécution des six exercices du projet
 */

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

int main() {

    int num;
    cin >> num;
    switch (num) {
        case 1:
            ex1();
            break;
        case 2:
            ex2();
            break;
        case 3:
            ex3();
            break;
        case 4:
            ex4();
            break;
        case 5:
            ex5();
            break;
        case 6:
            ex6();
            break;
        default:
            cout << "numero d'exercice inconnu";
    }

    return 0;
}
```

exercices.h

```
/**
 * @file exercices.h
 * @brief Fonctions pour les six exercices du projet
 */

#ifndef SDA_PROJET_EXERCICES_H
#define SDA_PROJET_EXERCICES_H

void ex1();
void ex2();
void ex3();
void ex4();
void ex5();
void ex6();

#endif //SDA_PROJET_EXERCICES_H
```



```

/**
 * @file exercices.cpp
 * @brief Fonctions pour les six exercices du projet
 */

#pragma warning(disable:4996)
#include "exercices.h"
#include "utils.h"
#include "PlateauMots.h"
#include <iostream>
#include <cstring>
using namespace std;

void ex1() {
    ListeMots liste;
    get_mots(liste);
    int pts = 0;
    for (int j = 0; j < longueur(liste); ++j)
        pts += evaluer(lire(liste, j));
    cout << pts;
    detruire(liste);
}

void ex2() {
    ListeMots liste;
    get_mots(liste);
    canonique(liste);
    afficher(liste);
    detruire(liste);
}

void ex3() {
    ListeMots liste1;
    get_mots(liste1);
    ListeMots liste2;
    get_mots(liste2);

    // suppression des doublons
    canonique(liste1);
    canonique(liste2);

    ListeMots uniques;

    for (int i = 0; i < longueur(liste2); ++i)
        if (!chercher(liste1, lire(liste2, i)))
            inserer(uniques, longueur(uniques), lire(liste2, i));
}

```

```

    afficher(uniques);

    detruire(liste1);
    detruire(liste2);
    detruire(uniques);
}

void ex4() {
    ListeMots liste1;
    get_mots(liste1);
    ListeMots liste2;
    get_mots(liste2);

    // suppression des doublons
    canonique(liste1);
    canonique(liste2);

    ListeMots communs;

    for (int i = 0; i < longueur(liste2); ++i)
        if (chercher(liste1, lire(liste2, i)))
            inserer(communs, longueur(communs), lire(liste2, i));

    afficher(communs);

    detruire(liste1);
    detruire(liste2);
    detruire(communs);
}

void ex5() {
    ListeListesMots listes;
    get_listes(listes);

    /* on "canonise" l'ensemble des listes pour en supprimer les doublons
    internes, qui ne comptent pas dans la recherche */
    for (int i = 0; i < listes.nb; ++i)
        canonique(listes.listes[i]);

    // on les unifie en une seule grande liste
    ListeMots liste1 = fusionner(listes);

    ListeMots liste2 = copier(liste1);
    canonique(liste2); // suppression des doublons inter-listes dans la copie

    /* la liste des mots présents dans au moins 2 listes est la soustraction
    entre la liste1 (unifiée) et la liste2 (une occurrence de chaque mot) */

```

```

    sub(liste1, liste2);

    canonique(liste1);
    afficher(liste1);

    detruire(listes);
    detruire(liste1);
    detruire(liste2);
}

void ex6() {

    PlateauMots p;
    int l = 0, x = 0;
    while (l < p.nbCases) {
        int y = 0;
        Mot mot;
        cin >> mot;
        while (y < 4) { // on stocke les lettres par 4
            p.cases[l].coord.x = x;
            p.cases[l].coord.y = y;
            p.cases[l].lettre = mot[y];
            ++y, ++l;
        }
        ++x;
    }

    ListeMots liste;
    get_mots(liste);
    canonique(liste);

    for (int i = 0; i < longueur(liste); ++i) {
        Mot mot;
        strcpy(mot, lire(liste, i));
        if (recherche(p, mot))
            cout << mot << endl;
    }
    cout << "*" << endl;
    delete[] p.cases;
    detruire(liste);
}

```

utils.h

```
/**
 * @file utils.h
 * @brief Fonctions utilitaires pour la réalisation des exercices du projet
 */

#ifndef SDA_PROJET_UTILS_H
#define SDA_PROJET_UTILS_H

#include "ListeListesMots.h"

/**
 * Lit et stocke les mots en entrée dans une liste de mots.
 * @param liste : la liste de mots
 */
void get_mots(ListeMots& liste);

/**
 * Lit un mot et renvoie son nombre de points correspondant.
 * @param mot : le mot à évaluer
 * @return le nombre de points correspondant
 */
int evaluer(const Mot mot);

/**
 * Affiche une liste de mots.
 * @param liste : la liste de mots
 */
void afficher(const ListeMots& liste);

/**
 * Trie alphabétiquement et supprime les doublons d'une liste de mots.
 * @param liste : la liste de mots
 */
void canonique(ListeMots& liste);

/**
 * Cherche un mot dans une liste de mots.
 * @param liste : la liste de mots
 * @param mot : le mot à chercher
 * @return vrai ou faux (selon que le mot ait été trouvé ou non)
 */
bool chercher(const ListeMots& liste, const Mot mot);

/**
 * Cherche un mot dans une liste de mots et renvoie sa position si trouvé.
 * @param liste : la liste de mots
 */
```

```

* @param mot : le mot à chercher
* @param index : variable dans laquelle stocker la position du mot dans la
* liste si trouvé (vaut -1 sinon)
* @return vrai ou faux (selon que le mot ait été trouvé ou non)
*/
bool chercher(const ListeMots& liste, const Mot mot, int& index);

/**
* Soustraie une liste à une autre liste : les occurrences du mot de la seconde
* liste dans la première seront toutes supprimées.
* @param liste1 : la liste de mots à modifier
* @param liste2 : la liste de mots dont on veut supprimer les mots dans la
* première
*/
void sub(ListeMots& liste1, const ListeMots& liste2);

#endif //SDA_PROJET_UTILS_H

```

utils.cpp

```

/**
* @file utils.cpp
* @brief Fonctions utilitaires pour la réalisation des exercices du projet
*/

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

void get_mots(ListeMots& liste) {
    while (true) {
        Mot mot;
        cin >> mot;
        if (!strcmp(mot, ""))
            break;
        inserer(liste, longueur(liste), mot);
    }
}

int evaluer(const Mot mot) {
    int l = strlen(mot);
    if (l <= 2)
        return 0;
}

```

```

        else if (l == 3 || l == 4)
            return 1;
        else if (l == 5)
            return 2;
        else if (l == 6)
            return 3;
        else if (l == 7)
            return 5;
        else
            return 11;
    }

void afficher(const ListeMots& liste) {
    for (int i = 0; i < longueur(liste); ++i)
        cout << lire(liste, i) << endl;
    cout << "*" << endl;
}

void canonique(ListeMots& liste) {
    // Méthode : comparaison deux à deux
    int i = 0, j = 1;
    while (j < longueur(liste)) {
        int k = strcmp(lire(liste, i), lire(liste, j));
        // si même mot
        if (k == 0) {
            supprimer(liste, j); // suppression du doublon
            continue; // on refait un tour sur les mêmes positions dans la liste
        }
        // si mot1 > mot2 alphabétiquement
        else if (k > 0) {
            Mot tmp;
            strcpy(tmp, lire(liste, i));
            supprimer(liste, i);
            inserer(liste, j, tmp);
            canonique(liste);
        }
        ++i, ++j;
    }
}

bool chercher(const ListeMots& liste, const Mot mot) {
    int i; // dummy
    return chercher(liste, mot, i);
}

bool chercher(const ListeMots& liste, const Mot mot, int& index) {
    for (int j = 0; j < longueur(liste); ++j)

```

```

        if (!strcmp(mot, lire(liste, j))) {
            index = j;
            return true;
        }
        index = -1;
        return false;
    }

void sub(ListeMots& liste1, const ListeMots& liste2) {
    assert(liste1.nbMots >= liste2.nbMots);

    for (int i = 0; i < longueur(liste2); ++i) {
        int j;
        Mot courant;
        strcpy(courant, lire(liste2, i));
        if (chercher(liste1, courant, j))
            supprimer(liste1, j);
    }
}

```

Mot.h

```

/**
 * @file Mot.h
 * @brief Définition de Mot comme chaîne de caractères
 */

#ifndef SDA_PROJET_MOT_H
#define SDA_PROJET_MOT_H

#define MOT_MAX 30
typedef char Mot[MOT_MAX + 1];

#endif //SDA_PROJET_MOT_H

```

ListeMots.h

```

/**
 * @file ListeMots.h
 * @brief Composant de liste de mots de capacité extensible.
 */

#ifndef SDA_PROJET_LISTEMOTS_H
#define SDA_PROJET_LISTEMOTS_H

```

```

#include "Mot.h"

#define LISTE_CAPACITE 10
#define LISTE_PAS 2

struct ListeMots {
    unsigned int capacite = LISTE_CAPACITE;
    unsigned int pas = LISTE_PAS;
    Mot* mots = new Mot[LISTE_CAPACITE];
    unsigned int nbMots = 0;
};

/**
 * @brief Désalloue un conteneur d'items en mémoire dynamique.
 * @see initialiser.
 * @param[in,out] c Le conteneur d'items.
 */
void detruire(ListeMots& l);

/**
 * Renvoie le nombre de mots d'une liste de mots.
 * @param l : la liste de mots
 * @return la longueur de la liste
 */
unsigned int longueur(const ListeMots& l);

/**
 * @brief Lecture d'un item d'un conteneur d'items.
 * @param[in] c Le conteneur d'items.
 * @param[in] i La position de l'item dans le conteneur.
 * @return L'item à la position i.
 * @pre i < c.capacite
 */
Mot& lire(const ListeMots& l, unsigned int i);

/**
 * @brief Ecrire un item dans un conteneur d'items.
 * @param[in,out] c Le conteneur d'items.
 * @param[in] i La position où ajouter/modifier l'item.
 * @param[in] it L'item à écrire.
 */
void ecrire(ListeMots& l, unsigned int i, const Mot& it);

/**
 * Insère un mot dans une liste à une position donnée.
 * @param l : la liste de mots
 * @param pos : la position dans la liste

```



```

    * @param it : le mot à insérer
    */
void inserer(ListeMots& l, unsigned int pos, const Mot& it);

/**
 * Supprime le mot dans une liste à la position donnée.
 * @param l : la liste de mots
 * @param pos : la position
 */
void supprimer(ListeMots& l, unsigned int pos);

/**
 * Renvoie une copie de la liste passée en paramètre.
 * @param l : la liste de mots à copier
 * @return une copie de cette liste sous forme d'une nouvelle liste de mots
 */
ListeMots copier(const ListeMots& l);

#endif //SDA_PROJET_LISTEMOTS_H

```

ListeMots.cpp

```

/**
 * @file ListeMots.cpp
 * @brief Composant de liste de mots de capacité extensible
 */

#pragma warning(disable:4996)
#include "ListeMots.h"
#include <cassert>
#include <cstring>

void detruire(ListeMots& l) {
    delete[] l.mots;
    l.mots = nullptr;
}

unsigned int longueur(const ListeMots& l) {
    return l.nbMots;
}

Mot& lire(const ListeMots& l, unsigned int pos) {
    assert(pos < l.nbMots);
    return l.mots[pos];
}

```

```

void ecrire(ListeMots& l, unsigned int pos, const Mot& it) {
    if (pos >= l.capacite) {
        unsigned int newTaille = (pos + 1) * l.pas;
        Mot* newT = new Mot[newTaille];
        for (unsigned int i = 0; i < l.capacite; ++i)
            strcpy(newT[i], l.mots[i]);
        delete[] l.mots;
        l.mots = newT;
        l.capacite = newTaille;
    }
    strcpy(l.mots[pos], it);
}

void inserer(ListeMots& l, unsigned int pos, const Mot& it) {
    assert(pos <= l.nbMots);
    for (unsigned int i = l.nbMots; i > pos; --i)
        ecrire(l, i, lire(l, i - 1));
    ecrire(l, pos, it);
    ++l.nbMots;
}

void supprimer(ListeMots& l, unsigned int pos) {
    assert((l.nbMots != 0) && (pos < l.nbMots));
    for (unsigned int i = pos; i < l.nbMots - 1; ++i)
        ecrire(l, i, lire(l, i + 1));
    --l.nbMots;
}

ListeMots copier(const ListeMots& l) {
    ListeMots c;
    for (int i = 0; i < l.nbMots; ++i)
        inserer(c, longueur(c), l.mots[i]);
    return c;
}

```

ListeListesMots.h

```

/**
 * @file ListeListesMots.h
 * @brief Liste de listes de mots
 */

#ifndef SDA_PROJET_LISTELISTESMOTS_H
#define SDA_PROJET_LISTELISTESMOTS_H

#include "ListeMots.h"

```

```

struct ListeListesMots {
    int capacite = 10;
    ListeMots* listes = new ListeMots[capacite];
    int nb = 0;
};

/**
 * Lit les listes de mots en entrée et les stocke.
 * @param listes : la liste de listes de mots
 */
void get_listes(ListeListesMots& listes);

/**
 * Vérifie si une liste de listes de mots a besoin d'être étendue et, le cas
 * échéant, réalise l'opération.
 * @param listes : la liste de listes de mots
 */
void etendre(ListeListesMots& listes);

/**
 * Détruit une liste de liste de mots.
 * @param listes : la liste de listes de mot
 */
void detruire(ListeListesMots& listes);

/**
 * Fusionne les listes contenues dans la liste de listes en une seule liste de
 * mots.
 * @param listes : la liste de listes de mots
 * @return la fusion de toutes les listes sous une seule liste de mots
 */
ListeMots fusionner(ListeListesMots& listes);

#endif //SDA_PROJET_LISTELISTESMOTS_H

```

ListeListesMots.cpp

```

/**
 * @file ListeListesMots.cpp
 * @brief Liste de listes de mots
 */

#pragma warning(disable:4996)
#include "ListeListesMots.h"
#include <iostream>
#include <cstring>
#include <cassert>

```

```

using namespace std;

void get_listes(ListeListesMots& listes) {
    while (true) {
        char courant[MOT_MAX + 1];
        char precedent[MOT_MAX + 1];
        cin >> courant;
        if (!strcmp(courant, "")) {
            if (!strcmp(precedent, courant)) // si le mot précédent est aussi
un "",
                break; // fin de la liste de listes
            strcpy(precedent, courant);
            ++listes.nb;
            continue;
        }
        etendre(listes); // vérifie si la liste (de listes) doit être étendue
        inserer(listes.listes[listes.nb], longueur(listes.listes[listes.nb]),
courant);
        strcpy(precedent, courant);
    }
}

void etendre(ListeListesMots& listes) {
    if (listes.nb == listes.capacite) {
        int newTaille = listes.capacite * 2;
        auto* newListe = new ListeMots[newTaille];
        for (int i = 0; i < listes.capacite; ++i)
            newListe[i] = listes.listes[i];
        delete[] listes.listes;
        listes.listes = newListe;
        listes.capacite = newTaille;
    }
}

void detruire(ListeListesMots& listes) {
    for (int i = 0; i < listes.nb; ++i)
        detruire(listes.listes[i]);
    delete[] listes.listes;
    listes.listes = nullptr;
}

ListeMots fusionner(ListeListesMots& listes) {
    ListeMots f;
    for (int i = 0; i < listes.nb; ++i)
        for (int j = 0; j < listes.listes[i].nbMots; ++j)
            inserer(f, longueur(f), listes.listes[i].mots[j]);
    return f;
}

```

PlateauMots.h

```
/**
 * @file PlateauMots.h
 * @brief Plateau de mots
 */

#ifndef SDA_PROJET_PLATEAUMOTS_H
#define SDA_PROJET_PLATEAUMOTS_H

#include "Mot.h"

struct Coord {
    int x;
    int y;
};

struct Case {
    Coord coord;
    char lettre;
    bool estVisitee;
};

struct PlateauMots {
    int x_max = 4;
    int y_max = 4;
    int nbCases = x_max * y_max;
    Case* cases = new Case[nbCases];
};

/**
 * Renvoie la case du plateau de mots correspondant à une position donnée. Les
 * coordonnées doivent être dans les bornes du plateau.
 * @param p : le plateau de mots
 * @param c : la coordonnée de la case
 * @return la case
 */
Case& at(const PlateauMots& p, const Coord& c);

/**
 * Recherche un mot sur un plateau de mots selon les règles du Boogle et indique
 * s'il a été trouvé.
 * @param p : le plateau de mots
 * @param mot : le mot à chercher
 * @return vrai ou faux (selon que le mot ait été trouvé ou non)
 */
bool recherche(const PlateauMots& p, const Mot mot);
```

```

/**
 * Algorithme récursif de recherche de la présence d'un mot sur un plateau de
 * mots (Boogle).
 * @see recherche
 * @param p : le plateau de mots
 * @param mot : le mot à chercher
 * @param pos : la position de début du mot dans le mot qu'on essaie de placer
 * @param c : la coordonnée à partir de laquelle on essaie de placer le mot
 * @return vrai ou faux (selon que le mot ait été trouvé ou non)
 */
bool sous_recherche(const PlateauMots& p, const Mot mot, int pos, Coord& c);

/**
 * Renvoie les huit coordonnées adjacentes à une coordonnée donnée.
 * @param c la coordonnée
 * @return le tableau de ses huit coordonnées adjacentes
 */
Coord* adjacentes(Coord c);

#endif //SDA_PROJET_PLATEAUMOTS_H

```

PlateauMots.cpp

```

/**
 * @file PlateauMots.cpp
 * @brief Plateau de mots
 */

#include "PlateauMots.h"
#include <cassert>
#include <cstring>

Case& at(const PlateauMots& p, const Coord& c) {
    assert(c.x >= 0 && c.x < p.x_max && c.y >= 0 && c.y < p.y_max);
    int i = 0;
    for (; i < p.nbCases; ++i)
        if (p.cases[i].coord.x == c.x && p.cases[i].coord.y == c.y)
            break;
    return p.cases[i];
}

bool recherche(const PlateauMots& p, const Mot mot) {
    for (int i = 0; i < p.nbCases; ++i)
        p.cases[i].estVisitee = false;
}

```

```

    for (int i = 0; i < p.nbCases; ++i) {
        Coord c = { p.cases[i].coord.x, p.cases[i].coord.y };
        if (sous_recherche(p, mot, 0, c))
            return true;
    }
    return false;
}

bool sous_recherche(const PlateauMots& p, const Mot mot, const int pos, Coord&
c) {
    if (pos >= strlen(mot))
        return true;
    if (!(c.x >= 0 && c.x < p.x_max && c.y >= 0 && c.y < p.y_max))
        return false;
    Case& ca = at(p, c); // les coordonnées correspondent à une case du platea
u, on la récupère
    if (ca.lettre != mot[pos])
        return false;
    if (ca.estVisitee)
        return false;
    ca.estVisitee = true;

    Coord* c_adjacentes = adjacentes(c);

    for (int i = 0; i < 8; ++i)
        if (sous_recherche(p, mot, pos + 1, c_adjacentes[i])) {
            delete[] c_adjacentes;
            return true;
        }
    ca.estVisitee = false;
    delete[] c_adjacentes;
    return false;
}

Coord* adjacentes(const Coord c) {
    auto* c_adjacentes = new Coord[8];

    // HAUT GAUCHE
    c_adjacentes[0].x = c.x - 1;
    c_adjacentes[0].y = c.y - 1;
    // HAUT
    c_adjacentes[1].x = c.x - 1;
    c_adjacentes[1].y = c.y;
    // HAUT DROITE
    c_adjacentes[2].x = c.x - 1;
    c_adjacentes[2].y = c.y + 1;
    // GAUCHE
    c_adjacentes[3].x = c.x;

```

```
c_adjacentes[3].y = c.y - 1;
// DROITE
c_adjacentes[4].x = c.x;
c_adjacentes[4].y = c.y + 1;
// BAS GAUCHE
c_adjacentes[5].x = c.x + 1;
c_adjacentes[5].y = c.y - 1;
// BAS
c_adjacentes[6].x = c.x + 1;
c_adjacentes[6].y = c.y;
// BAS DROITE
c_adjacentes[7].x = c.x + 1;
c_adjacentes[7].y = c.y + 1;

return c_adjacentes;
}
```