Rapport de Projet

structures de données et algorithmes

B G L
O L E

Stefan Radovanovic
Groupe 102

Zakaria Sellam
Groupe 102

Université de Paris

B G L
O L E

Table des Matières

Page 3 – Introduction

Page 4 – Graphe des Dépendances

Page 5 à 21 – Tests Unitaires

Page 22 – Bilan

Page 22 à 64 – Codes Sources

⇔

Page 23 à 29 – Main

Page 30 à 31 – Mot

Page 32 à 34 – Score

Page 35 à 38 – ConteneurMot

Page 39 à 40 – Inser

Page 41 à 46 – ListeCanonique

Page 47 à 48 – Affichage

Page 49 à 53 – Stockage

Page 54 à 58 – ListeCompare

Page 59 à 64 – Plateau

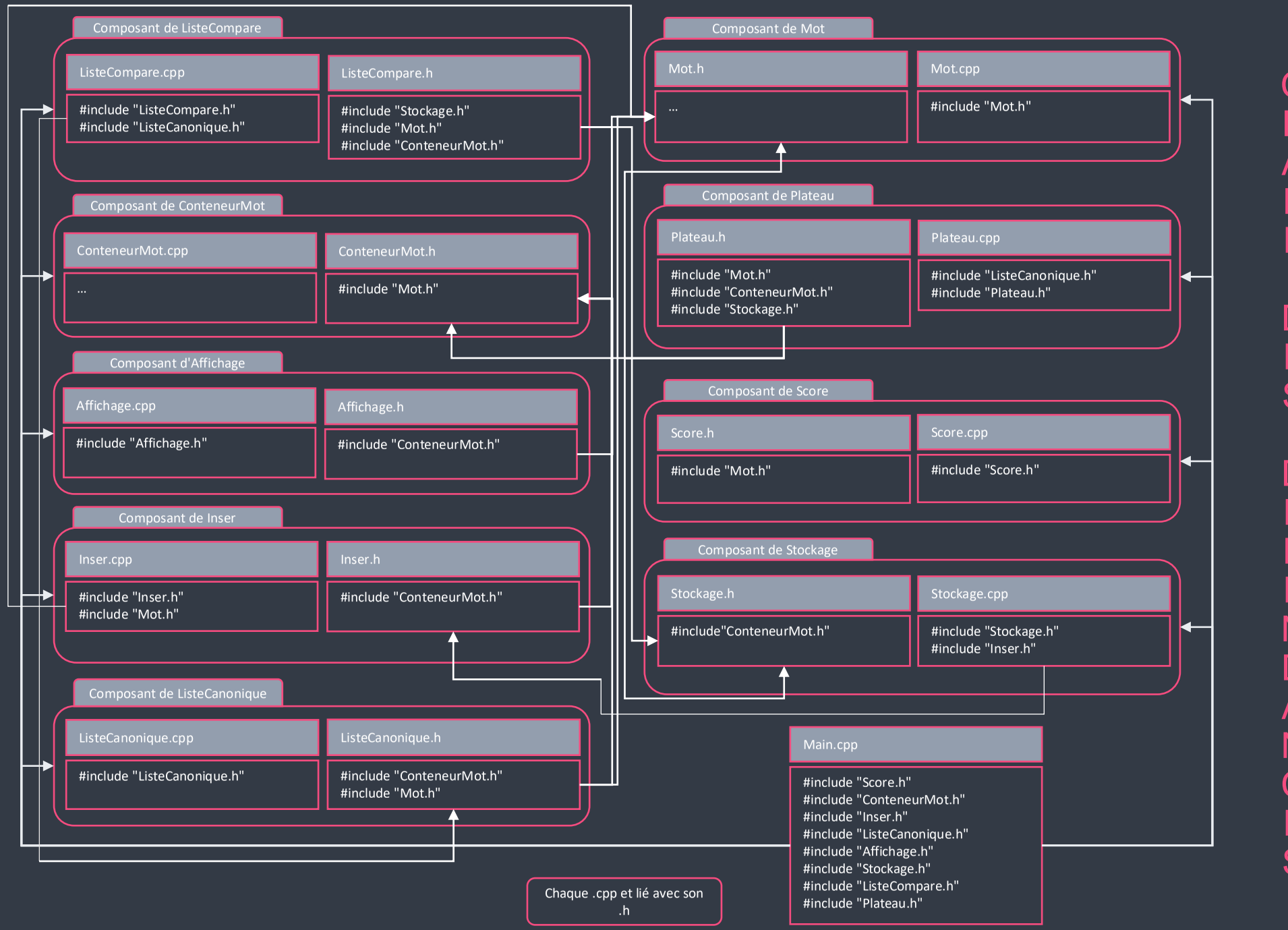
B G L
O L E

Introduction

Initialement conçu par Alan Turoff, le Boggle est un jeu qui se joue avec un plateau de lettre. Chaque joueur doit alors trouver le plus de mot possible à partir d'une lettre du tableau et de ses lettres adjacentes.

Ce Jeu est tout d'abord un jeu de société physique, ce qui représente tout l'enjeu du projet qui cherche à le numériser. Vous suivrez ainsi étapes par étapes la production de ce projet, en commençant d'abord par le graphe de dépendance de chaque fichier source, puis nous passerons en revue les tests unitaires de notre code pour enfin finir sur un bilan général du projet.

le résultat n'est qu'une esquisse de ce que devrait être une application qui fait tourner le jeu, ici nous nous concentrons sur la partie algorithmique.


Composant de ListeCompare
ListeCompare.cpp
#include "ListeCompare.h"
#include "ListeCanonique.h"
ListeCompare.h
#include "Stockage.h"
#include "Mot.h"
#include "ConteneurMot.h"
Composant de Mot
Mot.h
...
Mot.cpp
#include "Mot.h"
Composant de ConteneurMot
ConteneurMot.cpp
...
ConteneurMot.h
#include "Mot.h"
Composant de Plateau
Plateau.h
#include "Mot.h"
#include "ConteneurMot.h"
#include "Stockage.h"
Plateau.cpp
#include "ListeCanonique.h"
#include "Plateau.h"
Composant d'Affichage
Affichage.cpp
#include "Affichage.h"
Affichage.h
#include "ConteneurMot.h"
Composant de Score
Score.h
#include "Mot.h"
Score.cpp
#include "Score.h"
Composant de Inser
Inser.cpp
#include "Inser.h"
#include "Mot.h"
Inser.h
#include "ConteneurMot.h"
Composant de Stockage
Stockage.h
#include"ConteneurMot.h"
Stockage.cpp
#include "Stockage.h"
#include "Inser.h"
Composant de ListeCanonique
ListeCanonique.cpp
#include "ListeCanonique.h"
ListeCanonique.h
#include "ConteneurMot.h"
#include "Mot.h"
Main.cpp
#include "Score.h"
#include "ConteneurMot.h"
#include "Inser.h"
#include "ListeCanonique.h"
#include "Affichage.h"
#include "Stockage.h"
#include "ListeCompare.h"
#include "Plateau.h"
Chaque .cpp et lié avec son .h
GRAPE DES DEPENDANCES

B G L
O L E

Tests Unitaires

Pour tester nos algorithmes, le projet nous fournit des documents texte aussi appelés in et out qui représentent un exemple attendu d'entrée d'un utilisateur ainsi que la sortie qui en découle.

Ainsi le projet fournit six tests qui représentent les six algorithmes à faire pour la réalisation du jeu.

Ajouté à cela le « Test Ultime » qu'est un test annexe au projet permettant de mettre à rude épreuve l'optimisation du code, car on y teste tout les mots du dictionnaire français. De quoi faire chauffer la machine avec des temps d'exécution pouvant monter à plus de dix secondes.

De plus, nous avons testé nos algorithmes avec un jeu d'essai personnel afin de prévenir toutes les éventualités.

B G L
O L E

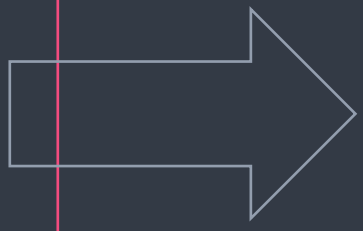
Tests Unitaires

Test 1 : Le premier test compte le nombre de score qu'apporte une liste de mots.

(rappel : un mot est limité à 30 caractères)

IN

```
CLIGNAS
EMBUE
ENBTGIE
ILE
AMENAT
TANGUE
BETNEA
NSTIE
IL
GITANS
MSEN
CLIGNES
OBEIT
LINS
LUE
ANIL
NIE
BEIGNE
LIENS
SEN
ENTEG
BOME
ICNALG
LUI
OBEIT
SEN
MSEN
OBEIT
MSEN
BEIGNE
NSTIE
NSTIE
EMBUE
ILE
LINS
LINS
*
```



OUT

```
71
points
Selon les regles
du jeu
```

B G L
O L E

Tests Unitaires

Test 1 :

```
C:\Users\X AE A-12\source\repos\testtewsdf\x64\Release>(echo 1 & type in1-2.txt) | BoggleSda.exe
71
```

Le résultat attendu est bien 71.

Test 1.bis : écriture d'un mot supérieure à 30 caractères.

```
1
MOTTRESLONGQUIFAITPLUSDETRENTELETTRES
Assertion failed: strlen(newMot) < MAXCARACTERE, file C:\Users\X AE A-12\source\repos\testtewsdf\Mot.cpp, line 22
```

Une assertion s'est bien déclenché

B G L
O L E

Tests Unitaires

Test 2 : Le second test vérifie la notion de liste canonique, c'est-à-dire une liste de mots trié par ordre alphabétique croissant et sans doublon.

IN

CLIGNAS
EMBUE
ENBTGIE
ILE
AMENAT
TANGUE
BETNEA
NSTIE
IL
GITANS
MSEN
CLIGNES
OBEIT
LINS
LUE
ANIL
NIE
BEIGNE
LIENS
SEN
ENTEG
BOME
ICNALG
LUI
OBEIT
SEN
MSEN
OBEIT
MSEN
BEIGNE
NSTIE
NSTIE
EMBUE
ILE
LINS
LINS
*

OUT

AMENAT
ANIL
BEIGNE
BETNEA
BOME
CLIGNAS
CLIGNES
EMBUE
ENBTGIE
ENTEG
GITANS
ICNALG
IL
ILE
LIENS
LINS
LUE
LUI
MSEN
NIE
NSTIE
OBEIT
SEN
TANGUE
*

Tests Unitaires

Test 2 :

```
C:\Users\X AE A-12\source\repos\testtewsdf\x64\Release>(echo 2 & type in1-2.txt) | BoggleSda.exe
AMENAT
ANIL
BEIGNE
BETNEA
BOME
CLIGNAS
CLIGNES
EMBUE
ENBTGIE
ENTEG
GITANS
ICNALG
IL
ILE
LIENS
LINS
LUE
LUI
MSEN
NIE
NSTIE
OBEIT
SEN
TANGUE
*
```

La liste est bien canonique

B G L
O L E

Tests Unitaires

Test 3 : Le troisième test vérifie pour deux listes de mots donnés les mots de la seconde liste n'apparaissant pas dans la première.

IN

BEIGNE
CLIGNAS
EMBUE
AMENAT
LUE
GITANS
IL
LIENS
LINS
CLIGNES
LUE
LUI
ILE
NIE
EMBUE
OBEIT
TANGUE
*

CLIGNAS
EMBUE
ENBTGIE
ILE
AMENAT
TANGUE
BETNEA
NSTIE
IL
GITANS
MSEN
CLIGNES
OBEIT
LINS
LUE
ANIL
NIE
BEIGNE
LIENS
SEN
ENTEG
BOME
ICNALG
LUI
OBEIT
SEN
MSEN
OBEIT
MSEN
BEIGNE
NSTIE
NSTIE
EMBUE
ILE
LINS
LINS
*

OUT

ANIL
BETNEA
BOME
ENBTGIE
ENTEG
ICNALG
MSEN
NSTIE
SEN
*

B G L
O L E

Tests Unitaires

Test 3 :

```
C:\Users\X AE A-12\source\repos\testtewsdf\x64\Release>(echo 3 & type in3.txt) | BoggleSda.exe
ANIL
BETNEA
BOME
ENBTGIE
ENTEG
ICNALG
MSEN
NSTIE
SEN
*
```

On retrouve bien les mots de la seconde liste n'apparaissant pas dans la première

B G L
O L E

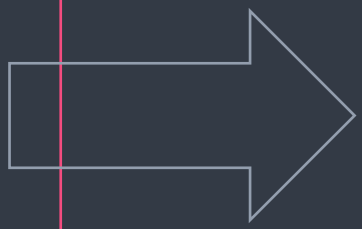
Tests Unitaires

Test 4 : Le quatrième test vérifie pour deux listes de mots donnés les mots de la seconde liste apparaissant dans la première.

IN

AMENAT
BEIGNE
CLIGNAS
CLIGNES
EMBUE
GITANS
IL
ILE
LIENS
LINS
LUE
LUI
NIE
OBEIT
TANGUE
*

CLIGNAS
EMBUE
ENBTGIE
ILE
AMENAT
TANGUE
BETNEA
NSTIE
IL
GITANS
MSEN
CLIGNES
OBEIT
LINS
LUE
ANIL
NIE
BEIGNE
LIENS
SEN
ENTEG
BOME
ICNALG
LUI
OBEIT
SEN
MSEN
OBEIT
MSEN
BEIGNE
NSTIE
NSTIE
EMBUE
ILE
LINS
LINS
*



OUT

AMENAT
BEIGNE
CLIGNAS
CLIGNES
EMBUE
GITANS
IL
ILE
LIENS
LINS
LUE
LUI
NIE
OBEIT
TANGUE
*

B G L
O L E

Tests Unitaires

Test 4 :

```
C:\Users\X AE A-12\source\repos\testtewsdf\x64\Release>(echo 4 & type in4.txt) | BoggleSda.exe
AMENAT
BEIGNE
CLIGNAS
CLIGNES
EMBUE
GITANS
IL
ILE
LIENS
LINS
LUE
LUI
NIE
OBEIT
TANGUE
*
```

On retrouve bien les mots de la seconde liste apparaissant dans la première

Il est possible que nous ayons raté une subtilité de l'énoncé, car les tests 3 et 4 étant presque identiques nous n'avions qu'à effectuer un changement de signe pour confirmer le test 4.

Tests Unitaires

Test 5 : Le cinquième test vérifie pour plusieurs listes de mots, les mots présents dans au moins deux de ces listes. Il exige également ces mots dans un ordre canonique.

IN

CLIGNAS
EMBUE
ENBTGIE
ILE
*

ILE
MSEN
LINS
OBEIT
MSEN
BEIGNE
NSTIE
EMBUE
ILE
LINS
*

*

AMENAT
TANGUE
BETNEA
NSTIE
IL
*

ENTEG
BOME
ICNALG
TANGUE
LUI
ICNALG
OBEIT
SEN
*

GITANS
MSEN
TANGUE
ENBTGIE
CLIGNES
OBEIT
LINS
*

LUE
ANIL
NSTIE
NIE
OBEIT
BEIGNE
LIENS
SEN
*

OUT

BEIGNE
EMBUE
ENBTGIE
ILE
LINS
MSEN
NSTIE
OBEIT
SEN
TANGUE
*

On teste aussi les listes vides, c'est-à-dire les listes composées d'une étoiles « * » pour indiquer la fin des entrées de l'utilisateur.

Tests Unitaires

Test 5 :

```
C:\Users\X AE A-12\source\repos\testtewsdf\x64\Release>(echo 5 & type in5.txt) | BoggleSda.exe
BEIGNE
EMBUE
ENBTGIE
ILE
LINS
MSEN
NSTIE
OBEIT
SEN
TANGUE
*
```

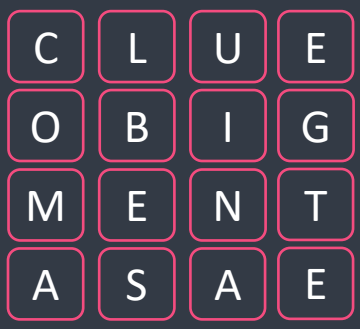
On retrouve bien les mots présents dans au moins deux des sept listes de mots.

Tests Unitaires

Test 6 : Le sixième test vérifie pour une liste de mots les mots que l'on peut trouver sur un plateau de lettre quatre fois quatre avec les règles du Boggle.

IN

CLIGNAS
EMBUE
ENBTGIE
ILE
AMENAT
TANGUE
SAINT
BETNEA
NSTIE
IL
GITANS
MSEN
CLIGNES
OBEIT
LUBIE
LINS
LUE
ANIL
NIE
GITANE
BEIGNE
LIENS
SEN
ENTEG
BOME
ICNALG
LUI
OBEIT
SEN
MSEN
OBEIT
MSEN
BEIGNE
NSTIE
COLOMBE
NSTIE
EMBUE
ILE
LINS
LINS
*



OUT

AMENAT
ANIL
BEIGNE
BOME
CLIGNAS
CLIGNES
EMBUE
GITANE
GITANS
IL
LIENS
LINS
LUBIE
LUE
LUI
MSEN
NIE
OBEIT
SEN
TANGUE
*

B G L
O L E

Tests Unitaires

Test 6 :

```
C:\Users\X AE A-12\source\repos\testtewsdf\x64\Release>(echo 6 & type in6.txt) | BoggleSda.exe
AMENAT
ANIL
BEIGNE
BOME
CLIGNAS
CLIGNES
EMBUE
GITANE
GITANS
IL
LIENS
LINS
LUBIE
LUE
LUI
MSEN
NIE
OBEIT
SEN
TANGUE
*
```

On retrouve tous les mots pouvant s'écrire sur le plateau.

Anecdote:
Pour cette partie, l'algorithme a été donné en pseudo code,
Le problème étant que nous n'étions pas au courant (mauvaise lecture
Document de notre part) et nous avons passé trois bons jours dessus sans savoir quoi faire,
quant on nous a informé l'existence de l'algorithme cela à fait l'effet
D'une lumière qui éclaire les ténèbres qu'étaient notre désespoir (un peu de poésie pour conclure le projet).

Tests Unitaires

Test 6.bis : Ce test vérifie la prise en charge d'une mauvaise saisit de tableau.

```
6
CLA DLSD DSAD DSAA
Assertion failed: lettre == ' ', file C:\Users\X AE A-12\source\repos\testtewsdf\Plateau.cpp, line 44
```

Il manque une lettre à la première ligne du plateau donc l'assertion s'est levé.

```
6
ASERT QWER QWER QWER
Assertion failed: lettre == ' ', file C:\Users\X AE A-12\source\repos\testtewsdf\Plateau.cpp, line 44
```

Pareil quant il y a une lettre en trop.

Tests Unitaires

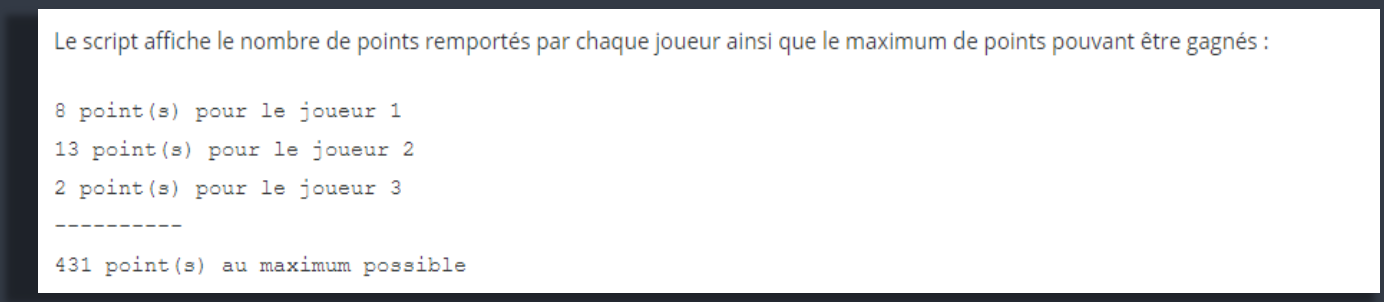
DISCLAIMER: Nous n'avons pas réussi a faire tourner le test ultime Avec la configuration du test6.bis, nous avons choisi de le retirer du projet et de le mettre en annexe ici.

```
void initPlateau(plateau& p)
{
//videz le buffer avant d'initialiser le tableau avec scanf

char lettre;
scanf("%c", &lettre);
for (int i = 0; i < LIGNE; ++i)
{

//etant donné que "cin" ne s'arrete pas aux espaces nous utilisons scanf pour
pouvoir verifier notre assertion
//si en dehors d'un ligne le char n'est pas un espace cela indique un tableau erroné
if (i > 0)
{
scanf("%c", &lettre);
assert(lettre == ' ');
}

for (int y = 0; y < COLONNE; ++y)
{

//une grille s'initialise dans le buffer sous cette forme : AAAA AAAA AAAA AAAA
scanf("%c", &lettre);
if(y >= 0)
p.tab[i][y].lettre = lettre;

}

}
}
```

B G L
O L E

Tests Unitaires

Test Ultime: le test ultime est un test qui s'exécute avec un ensemble de commande stocké dans un .bat (batch). Avec trois listes et un ods4 (liste de tous les mots de la langue française) l'objectif est de simulé une vrai partie de Boggle.


Le script affiche le nombre de points remportés par chaque joueur ainsi que le maximum de points pouvant être gagnés :

```
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
```

Confirmation du test ultime sur le Moodle de l'école

B G L
O L E

Tests Unitaires

Test Ultime:

```
C:\Users\X AE A-12\Desktop\Nouveau dossier\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 0:0:0.48 (0.48s total)

C:\Users\X AE A-12\Desktop\Nouveau dossier\ultime>
```

Les résultats sont identiques aux résultats attendus

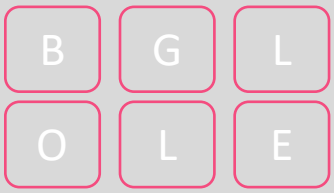
PS: Note spécial pour le temps, il s'avère que nous avons eu le meilleurs de l'école.

Cela s'explique en partie par l'utilisation d'un processeur AMD Ryzen 5600x Celui-ci est un des meilleurs du marché rapport qualité prix dans le domaine multi-tâche. Cela n'enlève en rien les efforts d'optimisation ☺.

B G L
O L E

Bilan du Projet

Ce projet correspond très bien au niveau d'un étudiant en première année d'informatique, dans la mesure où tout ce que nous avons vu en SDA est utilisé ici. Nous avons eu néanmoins un certain nombre d'écueils avant d'arriver à la fin. Point positif par rapport au projet d'IAP, nous avons eu la sensation d'être plus guidé durant ce projet, ce qui est totalement paradoxale, mais qui montre que notre niveau de compréhension en programmation s'est nettement amélioré. De plus, s'appuyer sur quelque chose de réel pour l'énoncé du projet nous a beaucoup aidée à visualiser le problème. Point négatif, avec les recherches que nous avons faites sur le C++ durant le projet, nous avons trouvé dommage de se limiter aux librairies du C et les quelques fonctions que nous avons vu du C++. Mais cela n'est pas de la faute du projet, mais plus tôt du programme qui ne couvre que la partie émergée de l'iceberg. Le derniers points que nous aimerions souligner est le retrait des recettes (les tests avec le prof), certes cela nous avait garantie une bonne note mais le stress qui va avec n'en valait pas la chandelle.



Codes Sources : Main

```
/**
* @file Main.cpp
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief Ici se trouve tout les jeux de test liées aux etapes du projet
*/

#include <iostream>

using namespace std;

#include "Mot.h"
#include "Score.h"
#include "ConteneurMot.h"
#include "Inser.h"
#include "ListeCanonique.h"
#include "Affichage.h"
#include "Stockage.h"
#include "ListeCompare.h"
#include "Plateau.h"
```

```
void exo1()
{
//affichage du score total d'une liste de mot
cout << getScore() << endl;

}

void exo2()
{
//initialisation
conteneurMot cMot;
initialiser(cMot);

//saisi des mots
inserMot(cMot);

//rendre la liste canonique
rendreCanonique(cMot);

//affichage
affichage(cMot);

//desallocation
suppInser(cMot);
detruire(cMot);
}
```

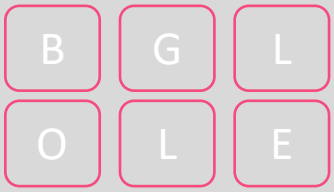
```
void exo3()
{
//initialisation du stockage de mot
stockageConteneur stockage;
initStockage(stockage);


//saisi des mots

for (unsigned int i = 0; i < 2; ++i) //2 listes
{

ajoutConteneur(stockage);
inserMot(stockage.cStockage[i]);

}

//rendre canonique nos 2 listes
for (unsigned int i = 0; i < stockage.nbStock; ++i) //
{
rendreCanonique(stockage.cStockage[i]);
}

// création d'un dernier conteneur
ajoutConteneur(stockage);

// ecriture des mots differents dans le dis conteneur
trouverDiff(stockage, 1, 0);

// affichage de la derniere liste
affichage(stockage.cStockage[stockage.nbStock - 1]);

//desalocation du stockage, des conteneurs stockés ainsi que des mots presents
formater(stockage);
}
```

```
void exo4()
{
//initialisation du stockage de mot
stockageConteneur stockage;
initStockage(stockage);


//saisi des mots

for (unsigned int i = 0; i < 2; ++i) //2 listes
{

ajoutConteneur(stockage);
inserMot(stockage.cStockage[i]);

}

//on rend canonique nos 2 listes
for (unsigned int i = 0; i < stockage.nbStock; ++i)
{
rendreCanonique(stockage.cStockage[i]);
}


//création d'un dernier conteneur
ajoutConteneur(stockage);

//ecriture des mots differents dans le dis conteneur
trouverEgalite(stockage, 0, 1);

//affichage de la derniere liste
affichage(stockage.cStockage[stockage.nbStock - 1]);

//desalocation du stockage, des conteneurs stockés ainsi que des mots presents
formater(stockage);
}
```

```
void exo5()
{
//initialisation du stockage de mot
stockageConteneur stockage;
initStockage(stockage);

//saisi des mots
int i = 0;
bool listeVide = false;
do
{

ajoutConteneur(stockage);
inserMot(stockage.cStockage[i]);
listeVide = estVide(stockage.cStockage[i]);
++i;

} while (!listeVide);

//on rend canonique nos listes
for (unsigned int i = 0; i < stockage.capacite; ++i) //
{
rendreCanonique(stockage.cStockage[i]);
}

//création d'un dernier conteneur
ajoutConteneur(stockage);

//recherche des mots presents dans au moins 2 listes
 comparerListe(stockage);

//affichage de la derniere liste
affichage(stockage.cStockage[stockage.nbStock - 1]);

//desalocation du stockage, des conteneurs stockés ainsi que des mots presents
formater(stockage);
}
```

```
void exo6()
{
//initialisation du stockage de mot
stockageConteneur stockage;
initStockage(stockage);

//initialisation du plateau
plateau p;
initPlateau(p);


//ajout d'un conteneur
ajoutConteneur(stockage);

//saisi des mots
inserMot(stockage.cStockage[0]);
//on rend canonique la liste
rendreCanonique(stockage.cStockage[0]);

//ajout d'un dernier conteneur
ajoutConteneur(stockage);

//recherche des mots de la liste dans le tableau
trouverMot(stockage, stockage.cStockage[0], p);

//affichage des mots trouvés
affichage(stockage.cStockage[stockage.nbStock - 1]);

//desalocation du stockage, des conteneurs stockés ainsi que des mots presents
formater(stockage);
}
```

```
int main(int argc, char** argv)
{
//exemple de commande (echo 1 & type in1-2.txt) | projetSda_Boggle.exe
int num;
cin >> num;
switch (num) {
case 1:
exo1(); break;
case 2:
exo2(); break;
case 3:
exo3(); break;
case 4:
exo4(); break;
case 5:
exo5(); break;
case 6:
exo6(); break;
}
return 0;
}
```



Codes Sources : Mot

```
#pragma once
/**
* @file Mot.h
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief saisi des mots de l'utilisateur
*/


#define MAXCARACTERE 30//30 caracteres max pour un mot

/** @brief definition d'un type Mot à allouer en memoire dynamique
 *avec pour capacité : 30 caracteres.
 */
typedef char* mot;


/**
* @brief initialisation d'un type mot en memoire dynamique pour recuperer
* la chaine de caractere de l'utilisateur
* @see supprimer (ConteneurMot.cpp), pour la desallocation des Mot
* @return le mot saisi par l'utilisateur
* @pre chaine de caractere < 30 caracteres
*/
mot motSaisir();

/**
* @brief desallocation d'un mot en memoire dynamique
* @see dejaSupp, pour verifier si le mot est deja supprimer
* @param[in, out] m : le mot
*/
void supprimer(mot& m);
```

```
/**
* @file Mot.cpp
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief saisi des mots de l'utilisateur
*/


#include <iostream>
#include <cassert>

using namespace std;

#include "Mot.h"

mot motSaisir()
{
//allocation d'un nouveau mot
mot newMot = new char[MAXCARACTERE];
cin >> newMot;
assert(strlen(newMot) < MAXCARACTERE);
return newMot;
}

void supprimer(mot& m)
{
if (m != nullptr || m[0] != '\0')//on verifie d'abord si le mot n'a pas deja été supprimer ou n'est pas alloué
//en regardant si la premiere lettre du mot est vide ( '\0' pour un char vide )
{
m = nullptr;
delete m;
}
}
```

Codes Sources : Score

```
#pragma once
/**
* @file Score.h
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 23/12/2020
* @brief calcul du score d'un mot, ainsi que le score total d'une liste de mot
*/

#include "Mot.h"


/**
* @brief calcul de la valeur d'un mot selon les regles du boggle
* @param[in] motSaisi : le mot
* @return le score qu'apporte un mot
*/
int calcScore(const mot& motSaisi);

/**
* @brief obtient le score totale d'une saisie de mots
* @return le score des mots donnés
*/
int getScore();
```

```
/**
* @file Score.cpp
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 23/12/2020
* @brief calcul du score d'un mot, ainsi que le score total d'une liste de mot
*/

#include <iostream>

using namespace std;

#include "Score.h"


int calcScore(const mot& motSaisi)
{
int score = 0;
int nbLettre = strlen(motSaisi);

//condition selon les regles du jeu
if (nbLettre == 3 || nbLettre == 4)
score = 1;
else if (nbLettre == 5)
score = 2;
else if (nbLettre == 6)
score = 3;
else if (nbLettre == 7)
score = 5;
else if (nbLettre >= 8)
score = 11;

return score;
}
```

```
int getScore()
{
int score = 0;
bool endListe;
do
{
mot motSaisi = motSaisir();
endListe = (strcmp(motSaisi, "*") == 0);

if (!endListe)
score += calcScore(motSaisi);

supprimer(motSaisi);//supression du mot saisi
} while (!endListe);
return score;
}
```

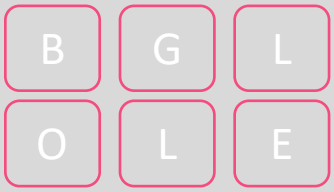
Codes Sources : ConteneurMot

```
#pragma once
/**
* @file ConteneurMot.h
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief creation d'un conteneur de mot
*/
#include "Mot.h"

/*@brief conteneur de Mot en memoire dynamique*/
struct conteneurMot
{
mot* listeMot;
unsigned int capacite;
unsigned int pExtension;
unsigned int nbMot;
};

/**
 * @brief Initialise un conteneur avec une liste de mot
 * Allocation en mémoire dynamique du conteneur
 * la capacite est par défaut à 1
 * le pas d'extension est par défaut à 2
 * @see detruire, pour sa désallocation en fin d'utilisation
 * @param[in,out] cMot : le conteneur d'items
 */
void initialiser(conteneurMot& cMot);

/**
 * @brief Désalloue un conteneur avec une liste de mot en mémoire dynamique
 * @see initialiser, le conteneur de mot a déjà été alloué
 * @see supprimer, pour la desallocation des mots
 * @param[in,out] cMot : le conteneur de mot
 */
void detruire(conteneurMot& cMot);
```

```
/**
 * @brief Lecture d'un mot d'un conteneur de mot
 * @param[in] cMot : le conteneur de mot
 * @param[in] i : la position du mot dans le conteneur
 * @return le mot à la position i
 * @pre i < cMot.nbMot && i >= 0
 */
mot lire(const conteneurMot& cMot, const unsigned int i);

/**
 * @brief Ecrire un mot dans un conteneur de mot
 * @param[in,out] cMot : le conteneur de mot
 * @param[in] newMot : le mot à écrire
 */
void ecrire(conteneurMot& cMot, const mot& elem);

/**
 * @brief determine si une liste de mot est une liste vide
 * quand le premier mot de la liste est une etoile "*"
 * @param[in] cMot : le conteneur de mot
 * @return booleen si c'est une liste vide
 */
bool estVide(const conteneurMot& cMot);
```

```cpp
/**
* @file ConteneurMot.cpp
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief creation d'un conteneur de mot
*/
#include <iostream>
#include <cassert>

using namespace std;

#include "ConteneurMot.h"


void initialiser(conteneurMot& cMot)
{
//allacotion d'une liste de capacite = 1 (extensible avec le pas d'extension)
cMot.listeMot = new mot[1];
cMot.capacite = 1;
//pas d'extension defini à 2
cMot.pExtension = 2;
cMot.nbMot = 0;
}


void detruire(conteneurMot& cMot)
{
//desallocation du conteneur de mot
//@see supprimer, pour la desallocation des mots
cMot.listeMot = nullptr;
delete[] cMot.listeMot;
cMot.capacite = 0;
cMot.nbMot = 0;
}
mot lire(const conteneurMot& cMot,const unsigned int i)
{
assert(i < cMot.nbMot && i >= 0);
return cMot.listeMot[i];
}
```

```
void ecrire(conteneurMot& cMot, const mot& newMot)
{
if (cMot.nbMot == cMot.capacite)
{

//la capacite double quand le conteneur est à saturation
unsigned int newTaille = cMot.capacite * cMot.pExtension;

//Allouez en mémoire dynamique un nouveau tableau (newT)
//à cette nouvelle taille
mot* newT = new mot[newTaille];

//copie des items déjà stockés dans le conteneur
for (unsigned int i = 0; i < cMot.nbMot; ++i)
{
newT[i] = cMot.listeMot[i];
}

//Désallocation de l'ancienne liste
//les mots de ne sont pas desaloué car gardés dans la nouvelle liste (pointeur sur adresse, char*)
delete[] cMot.listeMot;

//Actualiser la mise à jour du conteneur en mémoire dynamique
cMot.listeMot = newT;

//Actualisez la taille du conteneur
cMot.capacite = newTaille;

}

cMot.listeMot[cMot.nbMot] = newMot;
++cMot.nbMot;
}


bool estVide(const conteneurMot& cMot)
{
return strcmp(cMot.listeMot[0], "*") == 0;
}
```



Codes Sources : Inser

```
#pragma once
/**
* @file Inser.h
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 26/12/2020
* @brief gestion des entrées de l'utilisateur
*/

#include "ConteneurMot.h"

/**
* @brief enregistre les entrées dans une liste, fini par "*"
* @param[in, out] cMot : le conteneur de mots
*/
void inserMot(conteneurMot& cMot);

/**
* @brief supprime tout les mots du conteneur
* @see supprimer (Mot.h), pour la suppression des mots
* @param[in,out] cMot : le conteneur de mots
*/
void suppInser(conteneurMot& cMot);
```

```
/**
* @file Inser.cpp
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 26/12/2020
* @brief gestion des entrées de l'utilisateur
*/
#include <iostream>

using namespace std;

#include "Inser.h"
#include "Mot.h"

void inserMot(conteneurMot& cMot)
{
bool endListe;
do
{
mot motSaisi = motSaisir();
endListe = (strcmp(motSaisi, "*") == 0);//"*" marque la fin d'une liste
ecrire(cMot, motSaisi);
} while (!endListe);
}

void suppInser(conteneurMot& cMot)
{
for (unsigned int i = 0; i < cMot.nbMot; ++i)
{

supprimer(cMot.listeMot[i]);//désallocation des mots entrées

}
}
```

Codes Sources : ListeCanonique

```
#pragma once
/**
* @file ListeCanonique.h
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief rend les listes de mots canonique (ordre alphabetique croissant et sans doublon)
*/

#include "ConteneurMot.h"
#include "Mot.h"


/**
* @brief decalle le tableau d'une case en cas de suppresion de mot pour combler le vide
* @param (in,out) cMot : Conteneur avec une liste de mot
* @param (in) idx : indice du tableau
*/
void replacer(conteneurMot& cMot, const int idx);

/**
* @brief detect et supprime les doublons d'une liste de mot,
* la liste de mot doit d'abord etre trier par ordre alphabetique croissant !!
* @param (in,out) cMot : Conteneur avec une liste de mot
*/
void doublon(conteneurMot& cMot);

/**
* @brief compare deux mots en fonction de leur ordre dans l'alphabet
* @return si oui ou non les mots a et b sont dans l'ordre alphabetique
*/
bool ordreAlphabetique(const mot a, const mot b);
```

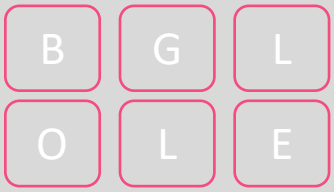
```
/**
* @brief rend la liste de mot canonique (trier + doublon)
* @param (in,out) cMot : Conteneur avec une liste de mot
*/
void rendreCanonique(conteneurMot& cMot);

/**
* @brief permutation entre deux mots
* @param (in, out) motA : mot remplacé
* @param (in, out) motB : mot remplacé
*/
void permut(mot& motA, mot& motB);

/**
* @brief compare les elements d'une partie du tableau (selon d et f)
* @see ordreAlphabetique pour les criteres de comparaison
* @param (in,out) cMot : Conteneur avec une liste de mot
* @param (in) d : indice de debut
* @param (in) f : indice de fin
* @return l'indice pivot
*/
int repartition(conteneurMot& cMot, int d, int f);


/**
* @brief tri recursif avec le systeme de pivot
* @param (in,out) cMot : Conteneur avec une liste de mot
* @param (in) d : indice de debut
* @param (in) f : indice de fin
*/
void triRapide(conteneurMot& cMot, int d, int f);
```

```
/**
* @file ListeCanonique.cpp
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief rend les listes de mots canonique (ordre alphabetique croissant et sans doublon)
*/
#include <iostream>

using namespace std;

#include "ListeCanonique.h«

void replacer(conteneurMot& cMot, const int idx)
{
//decalage des elements de la liste pour combler le vide
for (unsigned int i = idx; i < cMot.nbMot - 1; ++i)
cMot.listeMot[i] = cMot.listeMot[i + 1];

//on suppose qu'apres un decalage le nombre de mot de la liste a diminué
--cMot.nbMot;
}

void doublon(conteneurMot& cMot)
{
///!\ on assume que la liste de mot a deja été trié par ordre alphabetique croissant avant !
for (unsigned int i = 0; i < cMot.nbMot - 1; ++i)
{

//on compare avec l'element suivant car dans un tableau trié ce
//ne peut etre qu'un element different dans le cas contraire
//les doublons seront supprimé tant que le prochain mot est different
if (!(strcmp(cMot.listeMot[i], "*") == 0) &&
strcmp(cMot.listeMot[i], cMot.listeMot[i + 1]) == 0)
{

//desallocation du doublon
supprimer(cMot.listeMot[i + 1]);
replacer(cMot, i + 1);
--i;//permet de continuer les comparaisons avec le meme mot }
}
}
```

```
bool ordreAlphabetique(const mot a, const mot b)
{
if (strcmp(a, "*") == 0)//pour conserver l'etoile "*" en dernieres position
return false;//
else if (strcmp(b, "*") == 0)//s
return true;//

//taille du mot le plus court pour la boucle for
unsigned int taille = (strlen(a) >= strlen(b)) ? strlen(b) : strlen(a);

for (unsigned int i = 0; i < taille; ++i)
{

//comparaison entre deux lettres avec leur code ascii
if (a[i] == b[i])
{
continue;
}
return a[i] < b[i];

}
return strlen(a) < strlen(b);//exemple : ABBBA et ABBB
//doit etre stocké de cette facon : ABBB, ABBBA
}


void permut(mot& motA, mot& motB)
{
mot tmp = motA;
motA = motB;
motB = tmp;
}
```

```
int repartition(conteneurMot& cMot, int d, int f)
{
mot pivot = cMot.listeMot[f];
int i = (d - 1);

for (unsigned int j = d; j < f; ++j)
{
//on compare pour savoir s'il l'orde alphabetique est respecté
if (!ordreAlphabetique(pivot, cMot.listeMot[j]))
{
++i; //on imcremente l'indice de "debut"
permut(cMot.listeMot[i], cMot.listeMot[j]);
}

}
permut(cMot.listeMot[i + 1], cMot.listeMot[f]);
return i + 1;
}

void triRapide(conteneurMot& cMot, int d, int f)
{
if (d < f)
{
//permet d'avoir le bon pivot pour la suite des comparaisons
int pi = repartition(cMot, d, f);

triRapide(cMot, d, pi - 1);
triRapide(cMot, pi + 1, f);
}
}
```

```
void rendreCanonique(conteneurMot& cMot)
{
//trie necessaire s'il y a plus d'un mot
if (cMot.capacite > 1 && cMot.nbMot < 369085)//  369085 = le nombre de mot du dictionnaire
//soit une liste qu'est deja canonique :)
{

unsigned int fin = cMot.nbMot - 1;// simplifier la lecture
unsigned int debut = 0;//
triRapide(cMot, debut, fin);
doublon(cMot);
}
}
```



Codes Sources : Affichage

```
#pragma once
/**
* @file Affichage.h
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 26/12/2020
* @brief affichage de liste de mot
*/

#include "ConteneurMot.h"



/**
* @brief affiche les mots d'une liste
* @param [in] cMot : Conteneur avec une
liste de mot
*/
void affichage(const conteneurMot& cMot);
```

```
/**
* @file Affichage.cpp
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 26/12/2020
* @brief affichage de liste de mot
*/
#include <iostream>

using namespace std;

#include "Affichage.h"




void affichage(const conteneurMot& cMot)
{

for (unsigned int i = 0; i < cMot.nbMot; ++i)
{
cout << lire(cMot, i) << endl;
}

}
```

Codes Sources : Stockage

```
#pragma once
/**
* @file Stockage.h
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief definition d'un stockage pour les conteneurs de mots
*/


#include "ConteneurMot.h"


/*@brief structure d'un stockage de conteneur de Mot en memoire dynamique*/
struct stockageConteneur
{
conteneurMot* cStockage;
int capacite;
int nbStock;
};
```

```
/**
 * @brief Initialise un stockage de conteneur en memoire dynamique
 * la capacite est definit à 1 par défaut
 * @see formater, pour sa désallocation en fin d'utilisation
 * @param[in,out] stockage : le stockage de liste de mot
 */
void initStockage(stockageConteneur& stockage);

/**
* @brief Ajoute un conteneur de mot dans le stockage
* @see ConteneurMot.h, pour l'initialisation d'un conteneur
* @param [in, out] stocakge : le stockage de conteneur de mots
*/
void ajoutConteneur(stockageConteneur& stockage);

/**
* @brief desallocation de tout les conteneurs de mots et leurs mots
* @see ConteneurMot.h, Mot.H, Inser.h
* pour la desallocation d'un conteneur, d'un mot, d'une liste de mots
* @param [in, out] stocakge : le stockage de conteneur de mots
*/
void formater(stockageConteneur& stockage);
```

```
/**
* @file Stockage.cpp
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief definition d'un stockage pour les conteneurs de mots
*/

#include <iostream>
#include <cassert>

using namespace std;

#include "Stockage.h"
#include "Inser.h"


void initStockage(stockageConteneur& stockage)
{
//la capacité du stockage est definit a 1
stockage.cStockage = new conteneurMot[1];
stockage.capacite = 1;

stockage.nbStock = 0;
}
```

```
void ajoutConteneur(stockageConteneur& stockage)
{
if (stockage.nbStock == stockage.capacite)
{
//la capacite augmente de 1
unsigned int newCapa = stockage.capacite + 1;

//allocation d'un nouveau tableau de conteneur
conteneurMot* newC = new conteneurMot[newCapa];

//copie de l'ancien stockage dans un nouveau plus grand
for (int i = 0; i < stockage.capacite; ++i)
{
newC[i] = stockage.cStockage[i];
}
//supression de l'ancien stockage
delete[] stockage.cStockage;

//mis a jour du stockage et de la capacite
stockage.cStockage = newC;
stockage.capacite = newCapa;

}
//ajout du conteneur de mot dans le stockage
conteneurMot cMot;
initialiser(cMot);
stockage.cStockage[stockage.nbStock] = cMot;
++stockage.nbStock;
}
```

```
void formater(stockageConteneur& stockage)
{
//desallocation de tout les conteneurs
for (unsigned int i = 0; i < stockage.nbStock; ++i)
{
suppInser(stockage.cStockage[i]);//desallocation de la liste de mots
detruire(stockage.cStockage[i]);//desallocation du conteneur de mots
}
//desallocation du stockage
stockage.cStockage = nullptr;
delete[] stockage.cStockage;
stockage.capacite = 0;
}
```

Codes Sources : ListeCompare

```
#pragma once
/**
* @file Comparateur.h
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief checking de nos listes de mots pour en extraire une liste selon
* nos besoins (avec les mots en double ou non etc)
*/
#include "Stockage.h"
#include "ConteneurMot.h"
#include "Mot.h"

/**
* @brief verifie si un mot testé est present
* dans une liste de mot
* @param[in] cMot : le conteneur de mot
* @param[in] motTest : le mot testé
* @return la presence du mot dans la liste
*/
bool trouverCopie(const conteneurMot& cMot, mot motTest);

/**
* @brief trouve les mots d'une liste n'apparaissant pas dans la seconde
* @param[in,out] stockage : stockage des conteneurs et leurs mots
* @param[in] idx1 : indice de la premiere liste
* @param[in] idx2 : indice de la seconde liste
* @pre idx1 < stockage.nbStock&& idx1 - 1 >= 0) || (idx2 < stockage.nbStock - 1 && idx2 >= 0 || idx1 != idx2
* ces indices doivent etre les listes de mots de l'utilisateur et ne doivent pas etre les memes
*/
void trouverDiff(stockageConteneur& stockage, int idx1, int idx2);
```

```
/**
* @brief trouve les mots d'une liste apparaissant dans la seconde
* @param[in,out] stockage : stockage des conteneurs et leur mots
* @param[in] idx1 : indice de la premiere liste
* @param[in] idx2 : indice de la seconde liste
* @pre idx1 < stockage.nbStock&& idx1 - 1 >= 0) || (idx2 < stockage.nbStock - 1 && idx2 >= 0 || idx1 != idx2
* ces indices doivent etre les listes de mots de l'utilisateur et ne doivent pas etre les memes
*/
void trouverEgalite(stockageConteneur& stockage, int idx1, int idx2);

/**
* @brief cherche tout les mots ecris au moins deux listes de mots
* @param[in,out] stockage : stockage des conteneurs et leurs mots
*/
void comparerListe(stockageConteneur& stockage);
```

```
*/**
* @file Comparateur.cpp
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief checking de nos listes de mots pour en extraire une liste selon
* nos besoins (avec les mots en double ou non etc)
*/
#include <iostream>
#include <cassert>

using namespace std;

#include "ListeCompare.h"
#include "ListeCanonique.h"




bool trouverCopie(const conteneurMot& cMot, mot motTest)
{
for (unsigned int i = 0; i < cMot.nbMot; ++i)
{
if (strcmp(cMot.listeMot[i], motTest) == 0)
return true;
}
return false;
}
```

```
void trouverDiff(stockageConteneur& stockage, int idx1, int idx2)
{
assert((idx1 < stockage.nbStock&& idx1 - 1 >= 0) ||
(idx2 < stockage.nbStock - 1 && idx2 >= 0) ||
idx1 != idx2);

//comparaison de tout les mots de la premiere liste avec la deuxieme
for (unsigned int i = 0; i < stockage.cStockage[idx1].nbMot; ++i)
{

//on cherche s'il n'existe aucun mot identique à celui qu'on test pour l'ecrire dans une troisieme liste
if (strcmp(stockage.cStockage[idx1].listeMot[i], "*") == 0 ||
!trouverCopie(stockage.cStockage[idx2], stockage.cStockage[idx1].listeMot[i]))
{

//on cherche a savoir si le mot trouvé n'a pas deja été ecris avant de l'ecrire dans le dernier conteneur
if(!trouverCopie(stockage.cStockage[stockage.nbStock - 1], stockage.cStockage[idx1].listeMot[i]))
ecrire(stockage.cStockage[stockage.nbStock - 1], stockage.cStockage[idx1].listeMot[i]);

}
}
}
void trouverEgalite(stockageConteneur& stockage, int idx1, int idx2)
{
assert((idx1 < stockage.nbStock&& idx1 - 1 >= 0) ||
(idx2 < stockage.nbStock - 1 && idx2 >= 0) ||
idx1 != idx2);

for (int i = 0; i < stockage.cStockage[idx1].nbMot; ++i)
{

//on cherche s'il existe au moins un mot identique à celui qu'on test pour l'ecrire dans une troisieme liste
if (strcmp(stockage.cStockage[idx1].listeMot[i], "*") == 0 ||
trouverCopie(stockage.cStockage[idx2], stockage.cStockage[idx1].listeMot[i]))
{

//on cherche a savoir si le mot trouvé n'a pas deja été ecris avant de l'ecrire dans le dernier conteneur
if(!trouverCopie(stockage.cStockage[stockage.nbStock - 1], stockage.cStockage[idx1].listeMot[i]))
ecrire(stockage.cStockage[stockage.nbStock - 1], stockage.cStockage[idx1].listeMot[i]);
}
}
}
```

```
void comparerListe(stockageConteneur& stockage)
{
//on compare toutes nos listes (sauf la derniere crée car elle stock les mots trouvés
for (unsigned int i = 0; i < stockage.capacite - 1; ++i)
{
for (unsigned int y = 0; y < stockage.capacite - 1; ++y)
{

//les listes ne doivent pas etre identique et vide
if (y != i && !estVide(stockage.cStockage[i]) && !estVide(stockage.cStockage[y]))
{
//ecriture des mots trouvés dans notre derniere liste
trouverEgalite(stockage, i, y);
}


}
}

unsigned int fin = stockage.cStockage[stockage.nbStock - 1].nbMot - 1;// simplifier la lecture
unsigned int debut = 0;//
triRapide(stockage.cStockage[stockage.nbStock - 1], debut, fin);
}
```

Codes Sources : Plateau

```
#pragma once
/**
* @file Plateau.h
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief gestion du plateau de jeu
*/

#include "Mot.h"
#include "ConteneurMot.h"
#include "Stockage.h"

#define COLONNE 4//la grille est en 4x4
#define LIGNE 4//


/* @brief type position sur un plan 2d avec abscisse et ordonné */
struct position
{
int abs;//indice de ligne
int ord;//indice de colonne
};

/* @brief type element pour les composants de la grille */
struct element
{
char lettre;
bool visit;//case visitée ou non
};

/* @brief type plateau qui crée une grille 4x4 "d'elements" */
struct plateau
{
element tab[LIGNE][COLONNE];
};
```

```
/**
* @brief Initialise un plateau de 16 lettres
* toutes les cases sont indiqués comme "non visitée" (false)
* @param[in, out] p : le plateau
*/
void initPlateau(plateau& p);

/**
* @brief marque toutes les cases de la grille "non visitée" (false)
* @param[in, out] p : le plateau
*/
void falseTag(plateau& p);

/**
* @brief verifie sur les coordonnées
* @param[in] coord : coordonnées d'une case
* @return si oui ou non les coordonnées sont en dehors des limites de la grille
*/
bool horsLimite(const position& coord);

/**
* @brief recherche si le mot données dans le plateau (selon les regles du boggle)
* @param [in] m : le mot testé
* @param [in] p : le plateau
* @return true si le mot peut etre ecris sur le tableau, false sinon
*/
bool recherche(const mot& m, plateau& p);

/**
* @brief recherche recursive de chaque lettre d'un mot
* ici quand une lettre est trouvé sur le tableau
* on test toutes les cases adjacentes pour trouver la lettre suivante
* la fonction continue jusqu'à avec testé tout les chemins possibles
* @param [in] m : le mot
* @param [in] pos : indice de position dans un mot
* @param [in] coord : les coordonnées d'une case
* @param [in] p : le plateau
* @return true si le mot a ete trouvé, false sinon
*/
bool sousRecherche(const mot& m, int pos, position coord, plateau& p);
/**
* @brief verifie les mots d'une liste present dans le plateau puis les ecris
* dans une autre liste
* @param [in, out] stockage : stockage de conteneur de mot
* @param [in] cMot : conteneur de mot
* @param [in] p : le plateau
*/
void trouverMot(stockageConteneur& stockage, const conteneurMot& cMot, plateau& p);
```

```
/**
* @file Plateau.cpp
* Projet Sda
* @author Stefan Radovanovic Zakaria Sellam
* @version 24/12/2020
* @brief gestion du plateau de jeu
*/

#include <iostream>
#include <cassert>

using namespace std;

#include "ConteneurMot.h"
#include "Plateau.h"
#include "Mot.h"
#include "ListeCanonique.h"

//Visualisation d'un plateau 4*4 avec les coordonnées correspondantes
//0,0 | 0,1 | 0,2 | 0,3
//----|-----|-----|----
//1,0 | 1,1 | 1,2 | 1,3
//----|-----|-----|----
//2,0 | 2,1 |2,2| 2,3
//----|-----|-----|----
//3,0 | 3,1 | 3,2 | 3,3

void initPlateau(plateau& p)
{
for (int i = 0; i < LIGNE; ++i)
{
for (int y = 0; y < COLONNE; ++y)
{
//une grille s'initialise dans le buffer sous cette forme : AAAA AAAA AAAA AAAA
//les espaces indiques un saut de ligne dans la grille
char lettre;
cin >> lettre;
p.tab[i][y].lettre = lettre;

}
}
}
```

```
void falseTag(plateau& p)
{
for (int x = 0; x < LIGNE; ++x)
{
for (int y = 0; y < COLONNE; ++y)
{
p.tab[x][y].visit = false;
}
}
}

bool horsLimite(const position& coord)
{
//utilisée lors des tests de case adjacentes
//les coordonnées peuvent sortir de la grille
return coord.abs >= LIGNE || coord.abs < 0 ||
coord.ord >= COLONNE || coord.ord < 0;

}

bool recherche(const mot& m, plateau& p)
{
falseTag(p);//les cases doivent etre initialiser par "false" avant chaque recherche
for (int x = 0; x < LIGNE; ++x)
{
for (int y = 0; y < COLONNE; ++y)
{
position coord = { x, y };//la recherche commence a cette position de depart
if (sousRecherche(m, 0, coord, p))//et avec la premiere lettre du mot
return true;
}
}
return false;
}
```

```
bool sousRecherche(const mot& m, int pos, position coord, plateau& p)
{

if (pos > strlen(m) - 1)//quand l'indice de position dans le mot à atteint la fin du
return true;//mot cela veut dire que le mot a été trouvé

else if (horsLimite(coord))
return false;

else if(m[pos] != p.tab[coord.abs][coord.ord].lettre)
return false;

else if(p.tab[coord.abs][coord.ord].visit == true)//la case a deja été visitée
return false;

else p.tab[coord.abs][coord.ord].visit = true;

position coordTmp = coord;//sauvegarde des coordonnées initiales

//algo qui donne toutes les cases adjacentes
for (int x = -1; x <= 1; x++)
{
for (int y = -1; y <= 1; y++)
{
coord = coordTmp;
if (x != 0 || y != 0)
{

coord.abs += x;
coord.ord += y;

if(sousRecherche(m, pos + 1, coord, p))//recursion, on recherche la lettre suivante
return true;

}
}
}
coord = coordTmp;
p.tab[coord.abs][coord.ord].visit = false;
return false;
}
```

```
void trouverMot(stockageConteneur& stockage, const conteneurMot& cMot, plateau& p)
{
for (int i = 0; i < cMot.nbMot; ++i)
{
if (strcmp(cMot.listeMot[i], "*") == 0 || recherche(cMot.listeMot[i], p))
{

//ecriture dans le dernier conteneur du stockage
//(prévu pour accueillir les mots trouvés)
ecrire(stockage.cStockage[stockage.nbStock - 1], cMot.listeMot[i]);
}
}

}
```

B G L
O L E

FIN

Merci de votre lecture !

Université de Paris