ABDELJEBBAR mohamed amine
OUTREBON Séraphin
Groupe 108

SAE S1.02 : Comparaison d'approches algorithmiques – Lexicon –

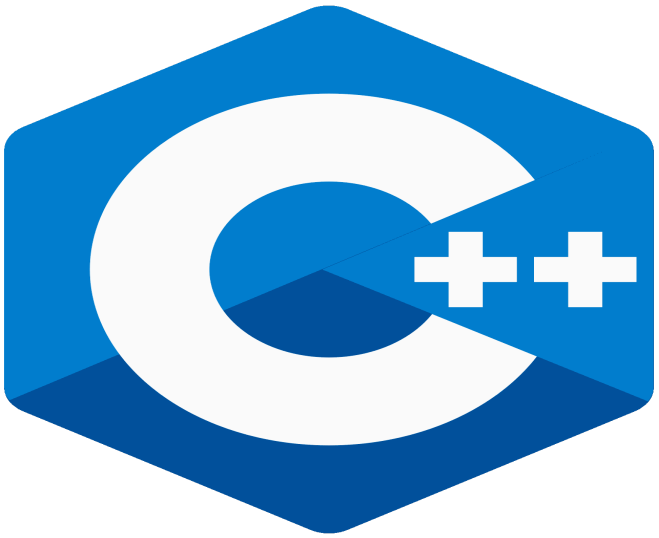


TABLE DES MATIÈRES:

En quoi consiste le projet? :

Le projet consiste à développer un logiciel complet pour le jeu “Lexicon”.Le jeu vise à se débarrasser des cartes rapidement. Le premier joueur à réussir ,termine le tour. Les autres joueurs accumulent des points équivalents aux chiffres sur leurs cartes restantes. Le premier à atteindre 100 points se retire, suivi des autres, jusqu'à ce qu'un seul joueur reste comme le gagnant.
L'application doit permettre à 2 à 4 joueurs de disputer une partie tout en respectant les règles officielles du jeu.
Les fonctionnalités principal de notre programme englobent:

1-mélange et distribution du Paquet de Cartes :

- Assurer le mélange d’un paquet de 51 cartes chacune associé à une lettre et un nombre de points
- Distribution équitable de 10 cartes a chaque joueurs
- Formation du talon avec le reste des cartes et première carte exposée.

2-Tours de Jeu Interactifs :

- Chaque joueur réalise une action parmi les cinq options à son tour.

les commande pour chaque Actions incluent:

“T”: Piocher une carte du talon et poser une des cartes en main sur l'exposé.
“E”:utiliser une carte en main pour remplacer la carte exposée.
“P”: le joueur peut former un mot avec les lettres de ces cartes.Si il est valide ,ce mot est ensuite posé sur la table et les cartes utilisées sont retirées de la main du joueur.
“R”:le joueur peut remplacer un mot existant sur la table par un nouveau mot. Après la commande le joueur peut entrer un numéro qui désigne l'ordre de la lettre qu'il veut changer.Le nouveau mot doit pouvoir être construit en remplaçant des lettres par d'autres lettres détenues par le joueur.
“C”: Le joueur a la possibilité de compléter un mot existant sur la table avec de nouvelles lettres. Le numéro spécifié désigne le mot à compléter, et le nouveau mot doit pouvoir être construit en insérant des lettres détenues par le joueur.

3-Affichage en Temps Réel :

- Affichage continu de la situation du jeu, comprenant le joueur actif, la carte exposée, les cartes en main, et la liste des mots posés.

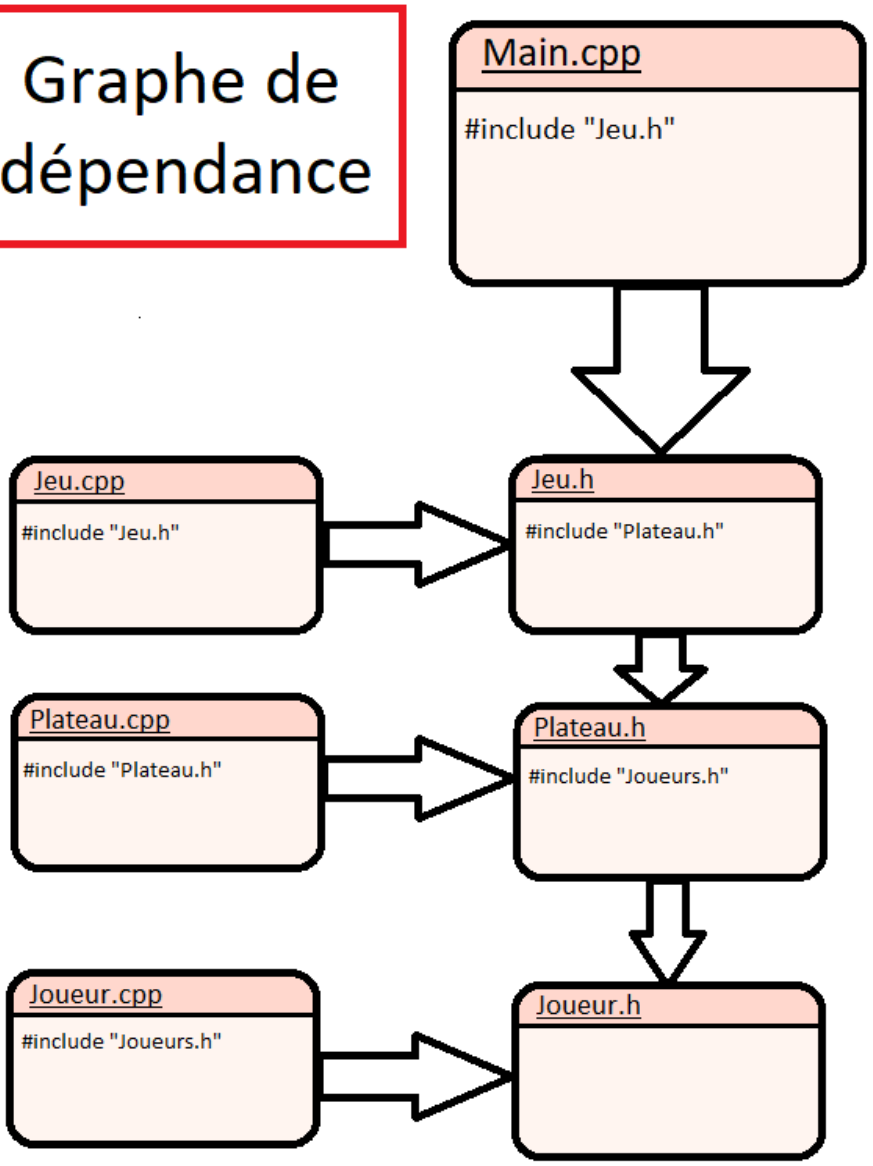
4-Vérifier la Validité des mots:

- Implémenter un dictionnaire pour vérifier si les mots proposés sont validés.
- Appliquer une pénalité en cas de mots invalides(+3 points).

5-Gérer la fin de la partie:

- Mettre fin automatiquement au programme lorsqu' il reste moins de deux joueurs.

Graphe de dépendance



Tests unitaires:

Pour les tests unitaires, nous les testons grâce à une fonction au-dessus du main appelé “Tests_unitaires”, lorsque nous voulons lancer ces test, nous mettons en paramètre **Partie(l)** ainsi que la boucle qui gère les tours en commentaire pour avoir un contrôle sur le jeu. Ensuite nous enlevons les différentes lignes mis entre commentaire pour faire en sorte de ne faire qu’un seul tour. Et enfin il n’y a plus qu'à choisir quelle partie des tests on veut tester. Dans cette fonction, nous avons divisé les différents tests entre eux pour ne pas causer de conflit, c’est pour ça qu’il faut choisir laquelle on veut tester et l’enlever du mode commentaires (enlever ça : /*#code*/). Chaque partie commence par un commentaire expliquant son objectif.

Également, concernant la réussite des différents tests unitaires que nous avons créés. Aucun test n'échoue si les conditions expliquées plus haut (mettre en commentaire certaines choses et enlever des choses du mode commentaire) est respecté.

```
void Test_unitaires(Liste& l,Tas& pioche,Tas& expose,Plateau&
plateau,Dictionnaire & dictionnaire, int
indice_lettre_dico[])//Fonction regroupant tout le test
unitaires
{
   //Vérifie que la liste enjeu s'actualise bien, que les
joueurs sont bien supprimé et que le tour commence bien avec
le bon nombre
   /*
   unsigned int i = l.enjeu[0]->numero;
   next(l);
   assert(i+1 == l.enjeu[0]->numero);
   l.enjeu[0]->NB_points+=100;
   unsigned int tmp = 2;
   unsigned int* tmp1 = &tmp;
   elimination(l,l.Taille_enjeu,tmp1);
   assert(*l.Taille_enjeu == 3);
   assert(l.enjeu[0]->numero == 4);
   cout << "bon" << endl;*/
   //Veriffie si les lettres sont bien comptées
   /*for(unsigned int i = 0; i < l.enjeu[0]->nb_carte;++i)
   {
       l.enjeu[0]->Main[i].Nom = 'A';
       l.enjeu[0]->Main[i].points = 10;
   }
```

```
compte_points(l,l.Taille_enjeu);
assert(l.enjeu[0]->NB_points == 100);*/

//Veriffie si la fonction Talon fonctione bien
/*l.enjeu[0]->Main[8].Nom = 'Z';
l.enjeu[0]->Main[8].points = 2;
for(unsigned int i = 0;i < 8;++i)
{
    l.enjeu[0]->Main[i].Nom = 'A';
    l.enjeu[0]->Main[i].points = 10;
}
l.enjeu[0]->Main[9].Nom = 'A';
l.enjeu[0]->Main[9].points = 10;
char lettre = 'B';
assert(!Talon(l,pioche,expose,lettre));
lettre = 'Z';
char lettretalon = pioche.carte[0].Nom;
assert(Talon(l,pioche,expose,lettre));
assert(l.enjeu[0]->Main[9].Nom == lettretalon);
assert(expose.carte[0].Nom == 'Z');*/

//Verifie si la fonction expose fonctione bien
/*
l.enjeu[0]->Main[8].Nom = 'Z';
l.enjeu[0]->Main[8].points = 2;
for(unsigned int i = 0;i < 8;++i)
{
    l.enjeu[0]->Main[i].Nom = 'A';
    l.enjeu[0]->Main[i].points = 10;
}
l.enjeu[0]->Main[9].Nom = 'A';
l.enjeu[0]->Main[9].points = 10;
char lettre = 'B';
assert(!exposee(l,expose,lettre));
lettre = 'Z';
char lettreexpose = expose.carte[0].Nom;
assert(exposee(l,expose,lettre));
assert(l.enjeu[0]->Main[9].Nom == lettreexpose);
assert(expose.carte[0].Nom == 'Z');
*/
//Verifie si poser fonctionne bien
/*
char mot[4] = "IYL";
```

```
assert(!poser(l,plateau,mot,3,dictionnaire,indice_lettre_dico)
);
   assert(l.enjeu[0]->nb_carte == 10);
   assert(plateau.nb_mots == 0);
   l.enjeu[0]->Main[0].Nom = 'I';
   l.enjeu[0]->Main[5].Nom = 'L';
   l.enjeu[0]->Main[9].Nom = 'Y';
   l.enjeu[0]->Main[0].points = 10;
   l.enjeu[0]->Main[5].points = 8;
   l.enjeu[0]->Main[9].points = 4;

assert(poser(l,plateau,mot,3,dictionnaire,indice_lettre_dico))
;
   assert(l.enjeu[0]->NB_points == 3);
   assert(l.enjeu[0]->nb_carte == 10);
   assert(plateau.nb_mots == 0);
   char mot1[3] = "IL";

assert(poser(l,plateau,mot1,2,dictionnaire,indice_lettre_dico)
);
   assert(l.enjeu[0]->nb_carte == 8);
   assert(plateau.nb_mots == 1);*/

   //Verifie si la fonction Remplacer
   /*
   l.enjeu[0]->Main[9].Nom = 'T';
   l.enjeu[0]->Main[8].Nom = 'E';
   l.enjeu[0]->Main[9].points = 8;
   l.enjeu[0]->Main[8].points = 10;
   l.enjeu[0]->Main[0].Nom = 'E';
   l.enjeu[0]->Main[1].Nom = 'S';
   l.enjeu[0]->Main[0].points = 10;
   l.enjeu[0]->Main[1].points = 8;
   for(unsigned int i = 2; i < 8; ++i)
   {
       l.enjeu[0]->Main[i].Nom = 'A';
       l.enjeu[0]->Main[i].points = 10;
   }
   char base[3] = "ES";
   poser(l,plateau,base,2,dictionnaire,indice_lettre_dico);
   char mot[3] = "EN";
```

```
assert(!remplacer(l,plateau,dictionnaire,2,mot,indice_lettre_d
ico));

assert(!remplacer(l,plateau,dictionnaire,1,mot,indice_lettre_d
ico));
   char mot1[3] = "EA";

assert(remplacer(l,plateau,dictionnaire,1,mot1,indice_lettre_d
ico));
   assert(l.enjeu[0]->NB_points == 3);
   char mot2[3] = "ET";

assert(remplacer(l,plateau,dictionnaire,1,mot2,indice_lettre_d
ico));
   assert(l.enjeu[0]->Main[7].Nom == 'S');
   */

   //Verifie si completer fonctionne
   /*
   l.enjeu[0]->Main[9].Nom = 'T';
   l.enjeu[0]->Main[8].Nom = 'E';
   l.enjeu[0]->Main[9].points = 8;
   l.enjeu[0]->Main[8].points = 10;
   l.enjeu[0]->Main[0].Nom = 'E';
   l.enjeu[0]->Main[1].Nom = 'S';
   l.enjeu[0]->Main[0].points = 10;
   l.enjeu[0]->Main[1].points = 8;
   for(unsigned int i = 2; i < 8; ++i)
   {
       l.enjeu[0]->Main[i].Nom = 'A';
       l.enjeu[0]->Main[i].points = 10;
   }
   char base[3] = "ES";
   poser(l,plateau,base,2,dictionnaire,indice_lettre_dico);
   char mot[4] = "Kes";

assert(!completer(l,plateau,dictionnaire,1,mot,3,indice_lettre
_dico));
   char mot1[4] = "AES";

assert(completer(l,plateau,dictionnaire,1,mot1,3,indice_lettre
_dico));
   assert(l.enjeu[0]->NB_points == 3);
```

```
   char mot2[4] = "EST";

assert(completer(l,plateau,dictionnaire,1,mot2,3,indice_lettre
_dico));
   assert(l.enjeu[0]->nb_carte == 7);*/


}
```

Bilan :

En ce qui concerne les difficultés que nous avons rencontrées sur ce projet, la première chose qui nous vient à l'esprit est d'éviter toute fuite de mémoire. En effet, nous avons décidé de créer une grande partie de nos variables de manière dynamique afin de pouvoir gérer la taille des tableaux. Ayant à peine compris comment utiliser correctement les fonctions new et delete, nous avons eu beaucoup de mal à allouer correctement la mémoire tout en la libérant à la fin du programme ou lorsqu'il était nécessaire de changer la taille d'un tableau.

Une autre difficulté que nous avons rencontrée au fil du projet était de tester nos fonctions de manière adéquate. Étant donné que le jeu nécessite l'initialisation des joueurs, des tas et du plateau, il était nécessaire de construire un grand nombre de fonctions avant même de pouvoir lancer le programme, ce qui a accumulé divers problèmes. De plus, une fois que le jeu était fonctionnel (c'est-à-dire qu'on pouvait au moins effectuer une commande), il était difficile de le tester intégralement. Nous avons donc implémenté au début certaines fonctions de débogage (que nous avons retirées de la version finale), telles qu'une fonction pour afficher les cartes d'un joueur, passer le tour d'un joueur sans effectuer de commande, et avons également mis en commentaire la vérification des mots dans le dictionnaire pour faciliter les tests.

Nous avons également utilisé des tests unitaires pour assurer une bonne couverture de test pour notre projet. Enfin, une autre difficulté qui nous a plus endurcis que ralentis était la récupération des commandes des utilisateurs ainsi que la séparation des éléments utiles, comme récupérer la lettre qui lance la commande ou encadrer le mot ajouté pour ensuite le stocker sans le reste. Étant donné que nous avons décidé de ne pas utiliser de bibliothèques autres que celles nécessaires pour les nombres aléatoires et pour lire et stocker un fichier texte, nous avons dû effectuer toutes ces manipulations de chaînes de caractères manuellement.

Malgré tous ces problèmes, le programme est complet et toutes les commandes ont été implémentées. La plupart des tableaux stockant des informations, comme les différents tas, les mains des joueurs, le tableau stockant les mots sur le plateau et les mots eux-mêmes qui stockent des cartes, sont entièrement dynamiques afin d'utiliser le moins de mémoire possible. Nous avons également essayé d'optimiser la recherche dans le dictionnaire pour valider un mot ou non. Sans optimisation, si un utilisateur écrivait un mot invalide, la fonction aurait parcouru l'intégralité du dictionnaire. Pour réduire la vérification, nous avons décidé de créer un tableau d'indices au début du programme qui stocke à partir de quel indice dans le

dictionnaire la première lettre du mot change. Cela a été possible car le dictionnaire était rangé dans l'ordre alphabétique, et donc notre vérification ne parcourt que la partie du dictionnaire commençant par la première lettre du mot à vérifier (par exemple, si le mot est BHY, la vérification ne compare que les mots commençant par B à BHY).

Nous avons également décidé d'utiliser une liste de joueurs pour gérer les tours des joueurs, la liste "enjeu". Le joueur se situant au premier indice de la liste sera celui qui devra jouer. Une fois son tour terminé, nous le retirons de la liste, rapprochons d'un cran vers le début de la liste les autres joueurs, puis remettons le joueur en fin de liste. Cela nous a permis de simplifier les fonctions, car à chaque modification pendant le jeu, elle concerne toujours le joueur au début de la liste "enjeu".

Cependant, il y a des points que nous pensons pouvoir améliorer. Nous pourrions encore optimiser le parcours du dictionnaire pour rendre la vérification encore plus facile. Une autre amélioration possible serait de ne pas poser uniquement des lettres sur le plateau, mais des cartes. Dans notre programme, puisque le mot devant être posé par un utilisateur est un caractère, nous attribuons juste le nom de la lettre au plateau. Nous sommes donc obligés, dans la fonction remplacer, d'attribuer manuellement les points des lettres récupérées par le joueur. Enfin, une dernière amélioration pourrait être de mieux répartir les fonctions dans différents fichiers CPP afin d'éviter d'avoir un fichier beaucoup plus imposant que les autres.

Annexe :

Joueur.h

```
#ifndef SAE_CARTE_JOUEURS_H
#define SAE_CARTE_JOUEURS_H
#include <iostream>
#include <ctime>
#include <iomanip>
#include <fstream>
using namespace std;
enum {NB = 51, DEBUT = 10,UN = 1,MAXMOTS =
369100,MAXTAILLEMOT = 26, MAX = 100};

typedef struct {char Nom; unsigned int points;}Carte;
typedef struct {unsigned int NB_points; Carte* Main;
unsigned int nb_carte; bool Eliminer; unsigned int
numero;}Joueur;
typedef struct {Joueur** liste_joueur;int Taille;
Joueur** enjeu; int* Taille_enjeu;}Liste;

void initialiser_liste_joueur(Liste& l,int nb_joueur);
void del_joueur(Liste& l,int nb_joueur);
#endif //SAE_CARTE_JOUEURS_H
```

Joueur.cpp

```
#include "Joueurs.h"

/** * @brief initialise la liste de joueur ainsi que les
joueurs
* @param[in/out] l : la liste des joueurs
* @param[in] nb_joueur : le nombre de joueur
*/
void initialiser_liste_joueur(Liste& l,int nb_joueur){
   l.liste_joueur = new Joueur* [nb_joueur];//on
initialise la liste de joueur
   l.enjeu = new Joueur* [nb_joueur];
```

```
   for(unsigned int i = 0;i<nb_joueur;++i)//On initialise
les joueurs
   {
       Joueur* j;
       j = new Joueur ;
       j->numero = i+1;
       j->Eliminer = false;
       j->NB_points = 0;
       j->Main = new Carte [DEBUT];
       j->nb_carte =DEBUT;
       l.liste_joueur[i] = j;
       l.enjeu[i] = j;
   }
}
/**
* @brief supprime la liste de joueur ainsi que les
joueurs
* @param[in/out] l : la liste des joueurs
* @param[in] nb_joueur : le nombre de joueur
*/
void del_joueur(Liste& l,int nb_joueur)
{
   for (unsigned int i = 0;i < nb_joueur;++i)//On
supprime les joueurs
   {
       delete [] l.liste_joueur[i]->Main;
       delete l.liste_joueur[i];
   }
   delete [] l.liste_joueur;
   l.liste_joueur = nullptr;
   delete [] l.enjeu;
   l.enjeu = nullptr;

}
```

Plateau.h

```
#ifndef SAE_CARTE_PLATEAU_H
#define SAE_CARTE_PLATEAU_H
// implémentation fonctions
#include <iomanip>
```

```
#include <fstream>
#include <iostream>
#include <ctime>
#include "Joueurs.h"


typedef struct {Carte* carte; unsigned int nb_carte;}Tas;
typedef struct {Carte* carte; unsigned int
nb_carte;}Mots;
typedef struct {Mots** Mots; unsigned int
nb_mots;}Plateau;
typedef struct {char Mot[MAXTAILLEMOT];}Mot_dictionnaire;
typedef struct {Mot_dictionnaire* dictionnaire; int*
nbmots;}Dictionnaire;

int chargerDictionnaire(const char* nomFichier,
Dictionnaire& dictionnaire,int indice_lettre_dico[]);
void del_tas(Carte*& pioche);
void initialiser_carte(Tas& pioche);
void melanger(Tas& pioche);
void distribuer(Tas& pioche, Liste& l,Tas& expose);
void initialiser_plateau(Plateau& plateau);
void delet_plateau(Plateau& plateau);

#endif //SAE_CARTE_PLATEAU_H
```

Plateau.cpp

```
#include "Plateau.h"

/**
* @brief Charger le dictionnaire dans un tableau +
stcoker à quelle indice le dictionnaire change de
première lettre
* @param[in] nomFichier : le fichier qu'on va lire
* @param[in/out] dictionnaire : la variable où on va
stocker les mots du dictionnaire
* @param[in/out] indice_lettre_dico : le tableau qui va
contenir les indices de la première apparition du'une
lettre dans le dictionnaire
```

```
* @return -1 si le dictionnaire n'a pas pu être lu ou 0
si il a pu être lu
*/
int chargerDictionnaire(const char* nomFichier,
Dictionnaire& dictionnaire,int indice_lettre_dico[]) {
    ifstream in(nomFichier);
    if (!in) {//Vérifie si le dictionnaire peut être lu
        cout << "Le dictionnaire n'a pu être ouvert." <<
std::endl;
        return -1; // Erreur à l'ouverture du fichier
    }
    dictionnaire.dictionnaire = new
Mot_dictionnaire[MAXMOTS];
    char opti = 'A';
    unsigned int tmp = 1;
    while (in >> setw(MAXTAILLEMOT) >>
dictionnaire.dictionnaire[*dictionnaire.nbmots].Mot &&
           *dictionnaire.nbmots < MAXMOTS) {
        if (*dictionnaire.nbmots > 0 &&
dictionnaire.dictionnaire[*dictionnaire.nbmots-1].Mot[0]
!=
dictionnaire.dictionnaire[*dictionnaire.nbmots].Mot[0]) {
            indice_lettre_dico[tmp] =
*dictionnaire.nbmots;
            tmp+=1;
        }
        *dictionnaire.nbmots+=1;
    }
    indice_lettre_dico[tmp] = *dictionnaire.nbmots;

    in.close();
    return 0; // Chargement réussi
}
/**
* @brief supprime un tas de carte
* @param[in/ou] l : tas de carte
*/
void del_tas(Carte*& pioche)
{
    delete [] pioche;
    pioche = nullptr;
```

```
}

/**
* @brief Initialise toutes les cartes du jeu
* @param[in/out] pioche : le talon
*/
void initialiser_carte(Tas& pioche)
{
   char a[NB+1] =
{"AABBCCDDEEEEEFGGHHIIIIJKLLMNNNOOPQRRRSSSTTTUUUVWXYZ"};
   for (unsigned int i = 0;i<NB;++i)//On parcourt toute
la chaine de caractère et le tableau de carte pour donner
les nom/points
   {
       pioche.carte[i].Nom = a[i];
       if (pioche.carte[i].Nom ==
'A'||pioche.carte[i].Nom == 'E' || pioche.carte[i].Nom ==
'I')
           pioche.carte[i].points = 10;
       else if (pioche.carte[i].Nom == 'B' ||
pioche.carte[i].Nom == 'F' || pioche.carte[i].Nom == 'X'
|| pioche.carte[i].Nom == 'Z')
           pioche.carte[i].points = 2;
       else if (pioche.carte[i].Nom == 'D' ||
pioche.carte[i].Nom == 'J')
           pioche.carte[i].points = 6;
       else if (pioche.carte[i].Nom == 'G' ||
pioche.carte[i].Nom == 'Q' || pioche.carte[i].Nom == 'Y')
           pioche.carte[i].points = 4;
       else
           pioche.carte[i].points = 8;
   }
}
/**
* @brief Mélange les cartes d'un tas
* @param[in/out] pioche : un tas de carte
*/
void melanger(Tas& pioche)
{
   srand(static_cast<unsigned int>(std::time(nullptr)));
```

```
   for(unsigned int i = 0; i < pioche.nb_carte; ++i)//On
parcourt toute la pioche pour mélanger toutes les cartes
   {
       unsigned int indice = rand() %
pioche.nb_carte;//On génère un nombre aléatoire
       Carte tmp = pioche.carte[indice];
       pioche.carte[indice] = pioche.carte[i];
       pioche.carte[i] = tmp;
   }
}
/**
* @brief Distribue les cartes aux joueurs, dépose une
carte du talon dans le tas exposées
* @param[in/out] pioche : le talon
* @param[in/out] l : la liste de joueur
* @param[in/out] expose : le tas de cartes exposées
*/
void distribuer(Tas& pioche, Liste& l,Tas& expose)
{
   int newtaille = pioche.nb_carte -
(DEBUT*(*l.Taille_enjeu));
   unsigned int tmp = pioche.nb_carte-UN;
   for (unsigned int i = 0; i < l.Taille;++i)//On
parcourt toute la liste de joueur
   {
       if (l.liste_joueur[i]->nb_carte != DEBUT)//Verifie
que le nb de carte de joueur n'est pas égale à 10
       {
           Carte* nouv = new Carte[DEBUT];
           delete [] l.liste_joueur[i]->Main;
           l.liste_joueur[i]->Main = nouv;
           l.liste_joueur[i]->nb_carte = DEBUT;
       }
       if (!l.liste_joueur[i]->Eliminer)//On verifier que
le joueur n'est pas éliminé
       {
           for (unsigned int j = 0; j < DEBUT; ++j)//On
parcourt sa main pour lui distribuer des cartes
           {
               l.liste_joueur[i]->Main[j] =
pioche.carte[tmp];
```

```
                tmp-=1;

            }
        }
    }
    expose.carte[UN-1] = pioche.carte[tmp];//On donne la
première carte de la pioche à l'exposée
    Carte* nouv = new Carte[newtaille];//On réduit la
taille du talon

    for (unsigned int k = 0;k<newtaille;++k)//Creation du
tableau dynamique
    {
        nouv[k] = pioche.carte[k];
    }
    delete [] pioche.carte;//On supprime l'ancien talon
    pioche.carte = nouv;
    pioche.nb_carte = newtaille;
}
/**
* @brief Initialise le plateau
* @param[in/out] plateau: le plateau contenant les mots
posés
*/
void initialiser_plateau(Plateau& plateau)
{
    plateau.nb_mots = 0;
}
/**
* @brief Supprime le plateau
* @param[in/out] plateau: le plateau
*/
void delet_plateau(Plateau& plateau)
{
    for(unsigned int i = 0; i< plateau.nb_mots;++i)//On
supprime tous les mots de plateau
    {

        delete [] plateau.Mots[i]->carte;
        plateau.Mots[i]->carte = nullptr;
        delete plateau.Mots[i];
```

```
        plateau.Mots[i] = nullptr;
    }
    delete plateau.Mots;//On supprime le plateau
    plateau.Mots = nullptr;
}
```

Jeu.h

```
#ifndef SAE_CARTE_JEU_H
#define SAE_CARTE_JEU_H
#include "Plateau.h"
#include "Joueurs.h"
#include <iostream>
#include <ctime>
#include <iomanip>
#include <fstream>



bool estMotValide(const char* mot,const Dictionnaire&
dictinnaire,const int indice_lettre_dico[], unsigned int
taille_mot);
bool partie(const Liste& l);
void elimination(Liste& l,int* taille_enjeu,unsigned int*
commence);
void compte_points(Liste& l, int* nb);
void affiche_tour(const Liste& l);
void affiche_jeu(const Liste& l, const Tas& expo, const
Plateau& plateau);
void next(Liste& l);
bool detient(const Liste& l, char carte);
unsigned int indice_carte_inventaire(const Liste& l,
unsigned int indice, char carte);
void enleve_carte_inventaire(Liste& l, char lettre);
void supr_carte_inventaire(Liste&l, char lettre);
unsigned int avance_str(char a[], unsigned int indice);
unsigned int compte_droite(char a[], unsigned int
indice);
unsigned int taille_char(char a[]);
bool Talon(Liste& l, Tas& pioche, Tas& expo,char carte);
bool exposee(Liste& l, Tas& expo, char carte);
```

```
bool poser(Liste& l, Plateau& plateau, char mot[],
unsigned int taille,const Dictionnaire&
dictionnaire,const int indice_lettre_dico[]);
bool remplacer(Liste& l , Plateau& plateau,const
Dictionnaire& dictionnaire,unsigned int indice,char
mot[],const int indice_lettre_dico[]);
bool completer(Liste& l, Plateau& plateau,const
Dictionnaire& dictionnaire, unsigned int indice, char
mot[],unsigned int taille,const int
indice_lettre_dico[]);
void jouer( Liste& l, Tas& pioche, Tas& expo, Plateau&
plateau,const Dictionnaire& dictionnaire,const int
indice_lettre_dico[]);


#endif //SAE_CARTE_JEU_H
```

Jeu.cpp

```
#include "Jeu.h"
/**
* @brief Verrifie que le mot donné en paramètre est dans
le dictionnaire
* @param[in] mot : le mot qu'on doit vérifier
* @param[in] dictionnaire : le dictionnaire
* @param[in] indice_lettre_dico : le tableau qui contient
les indices de la première apparition du'une lettre dans
le dictionnaire
* @param[in] taille_mot : la taille du mot à vérifier
* @return true si le mot est dans le dictionnaire sinon
false
*/
bool estMotValide(const char* mot,const Dictionnaire&
dictinnaire,const int indice_lettre_dico[], unsigned int
taille_mot) {
   unsigned int indice = mot[0] - 'A';
   unsigned int tmp;
   unsigned int nb_lettre;
   for (unsigned int i = indice_lettre_dico[indice]; i <
indice_lettre_dico[indice+1]; ++i) {//On parcourt le
dictionnaire de mot entre deux indices en fonction des
lettres
```

```
        tmp = 0;
        nb_lettre = 0;
        bool bon = true;
        while (dictinnaire.dictionnaire[i].Mot[tmp] !=
'\0' && bon) {//Verifie si le mot qu'on compare n'est pas
terminé
            bon = false;
            if (dictinnaire.dictionnaire[i].Mot[tmp] ==
mot[tmp])//Vérifie si la lettre tmp du dictionnaire et du
mot a vérifié sont les mêmes
                bon = true;
            nb_lettre+=1;
            tmp+=1;
        }
        if(nb_lettre < taille_mot)//Vérifie qu'il ne reste
pas des lettres à vérifier dans le mot vérifié
            bon = false;//Le mot n'est pas dans le
dictionnaire
        if(bon)
            return true;//Le mot est dans le dictionnaire
    }
    return false;
}
/**
* @brief Vérifie qu'il y a au moins deux joueurs en vie
* @param[in] l : la liste de joueur
* @return true si il y a plus de deux joueurs en vie
sinon false
*/
bool partie(const Liste& l)
{
    if (*l.Taille_enjeu < 2)//Vérifie s'il y a assez de
joueurs en vie
        return false;//Pas assez de joueurs
    else
        return true;//Assez de joueurs
}
/**
* @brief Elimine les joueurs avec plus de 100 points et
établit l'ordre de jeu
* @param[in/out] l : la liste de joueur
```

```
* @param[in/out] taille_enjeu : la taille des joueurs en
jeu
* @param[in/out] commence : quelle joueur commence ce
tour
*/
void elimination(Liste& l,int* taille_enjeu,unsigned int*
commence)
{
   unsigned int elimine = 0;
   for (unsigned int i =
0;i<*l.Taille_enjeu;++i)//Parcourt la liste de joueur en
jeu
   {
       if (l.enjeu[i]->NB_points > MAX)//Si le joueur à
100 points ou plus, on l'élimine
       {
           l.liste_joueur[l.enjeu[i]->numero-1]->Eliminer
= true;
           l.enjeu[i]->Eliminer = true;
           elimine+=1;
       }
   }
   if(elimine > 0)//S'il y a au moins un joueur éliminé,
on change la taille du tableau qui contient les joueurs
en jeu
   {
       Joueur** tmp = new Joueur*[*taille_enjeu-elimine];
       int temp = 0;
       for (unsigned int j = 0; j < *l.Taille_enjeu; ++j)
       {
           if(!l.enjeu[j]->Eliminer)
           {
               tmp[temp] = l.enjeu[j];
               temp+=1;
           }
       }
       delete [] l.enjeu;
       l.enjeu = tmp;
       *l.Taille_enjeu = *taille_enjeu-elimine;
   }
   *commence+=1;
```

```
    if(*commence == l.Taille)
        *commence = 0;
    unsigned int temp = 0;
    for (unsigned int j = *commence; j < l.Taille;
++j)//Complète le tableau en jeu en fonction de quel
joueur doit commencer
    {
        if(!l.liste_joueur[j]->Eliminer)
        {
            l.enjeu[temp] = l.liste_joueur[j];
            temp+=1;
        }
    }
    for(unsigned int j = 0 ; j < *commence; ++j)
    {
        if(!l.liste_joueur[j]->Eliminer)
        {
            l.enjeu[temp] = l.liste_joueur[j];
            temp+=1;
        }
    }
}
/**
* @brief Compte les points de chaque joueurs encore en
vie
* @param[in/out] l : la liste de joueur
* @param[in] nb : le nombre de joueur en jeu
*/
void compte_points(Liste& l, int* nb)
{
    for(unsigned int i = 0; i < *nb;++i)//Parcourt la
liste des joueurs en jeu
    {
        for(unsigned int j = 0; j < l.enjeu[i]->nb_carte;
++j) {//Parcourt la main des joueurs
            l.enjeu[i]->NB_points +=
l.enjeu[i]->Main[j].points;
        }
    }
}
/**
```

```
* @brief Affiche les informations à la fin d'un tour
* @param[in] l : la liste de joueur
*/
void affiche_tour(const Liste& l)
{
   cout << endl << "Le tour est fini" << endl << "*
Scores" << endl;
   for (unsigned int i = 0; i < l.Taille; ++i)//Parcourt
la liste de joueur
   {
       if(l.liste_joueur[i]->NB_points < 2 &&
!(l.liste_joueur[i]->Eliminer))//Si le joueur n'est pas
éliminé et qu'il a au plus un point
           cout << "Joueur " << l.liste_joueur[i]->numero
<< " : " << l.liste_joueur[i]->NB_points << " point" <<
endl;
       else if(!l.liste_joueur[i]->Eliminer)//Si le
joueur n'est pas éliminé
           cout << "Joueur " << l.liste_joueur[i]->numero
<< " : " << l.liste_joueur[i]->NB_points << " points" <<
endl;
   }

}
/**
* @brief Affiche les informations du joueur devant jouer
ainsi que l'état du jeu
* @param[in] l : la liste de joueur
* @param[in] expo : le tas de cartes exposées
* @param[in] plateau : le plateau avec les mots déjà
composés
*/
void affiche_jeu(const Liste& l, const Tas& expo, const
Plateau& plateau)
{
   cout << "* Joueur " << l.enjeu[0]->numero << " (" <<
expo.carte[0].Nom << ") ";
   for(unsigned int i = 0;i <
l.enjeu[0]->nb_carte;++i)//Parcourt la main du joueur
   {
```

```
        cout << l.enjeu[0]->Main[i].Nom;//Affiche les
cartes
    }
    if(plateau.nb_mots > 0)//S'il y a au moins un mot de
posé
    {
        cout << endl;
        for (unsigned int j = 0; j < plateau.nb_mots;
++j)//Parcourt les mots posés
        {
            cout << j+1 << " - ";
            for (unsigned int k = 0; k <
plateau.Mots[j]->nb_carte; ++k)//Parcourt les lettres des
mots
            {
                cout <<
plateau.Mots[j]->carte[k].Nom;//Affiche les lettres
            }
            if(j != plateau.nb_mots - 1)
                cout << endl;
        }
    }
    cout << endl << "> ";
}
/**
* @brief Passe au joueur suivant qui doit jouer
* @param[in/out] l: la liste de joueur
*/
void next(Liste& l)
{
    Joueur* tmp = l.enjeu[0];
    for (unsigned int i = 1; i <
*l.Taille_enjeu;++i)//Parcourt la liste des joueurs en
vie à partir de l'indice 1 et les rapprochent du début de
liste
    {
        l.enjeu[i-1] = l.enjeu[i];
    }
    l.enjeu[*l.Taille_enjeu-1] = tmp;//Met le joueur qui a
joué à la fin du tableau
}
```

```
/**
* @brief Verifie si le joueur possède les cartes qu'il
veut déposer
* @param[in] l: le liste de joueur
* @param[in] l: la lettre à
* @return true si le joueur à la lettre sinon false
*/
bool detient(const Liste& l, char carte)
{
   for (unsigned int i = 0; i < l.enjeu[0]->nb_carte;
++i)//Parcourt la main du joueur
   {
       if(l.enjeu[0]->Main[i].Nom == carte)//Si le joueur
possède la carte
           return true;
   }
   return false;//Le joueur ne possède pas la carte
}
/**
* @brief Renvoie l'indice d'où est stocké la carte dans
la main du joueur
* @param[in] l: le liste de joueur
* @param[in] indice: le numéro du joueur
* @param[in] carte: la carte recherchée
* @return l'indice de la carte
*/
unsigned int indice_carte_inventaire(const Liste& l,
unsigned int indice, char carte)
{
   for(unsigned int i = 0; i <
l.liste_joueur[indice-1]->nb_carte; ++i)//Parcourt la
main du joueur
   {
       if(l.liste_joueur[indice-1]->Main[i].Nom ==
carte)//Si le joueur possède la carte recherchée
           return i;//Renvoie l'indice de la carte
   }

   // Aucune carte n'a été trouvée
   return 0;
```

```
}

/**
* @brief Supprime la carte demandée de l'inventaire du
joueur donné en paramètre
* @param[in/out] l: le liste de joueur
* @param[in] lettre: la carte recherchée
*/
void enleve_carte_inventaire(Liste& l, char lettre)
{
   for(unsigned int i =
indice_carte_inventaire(l,l.enjeu[0]->numero,lettre); i <
l.enjeu[0]->nb_carte; ++i)//On parcourt la main depuis
l'indice de la carte que l'on retire
       l.enjeu[0]->Main[i] = l.enjeu[0]->Main[i+1];//On
rapproche les cartes vers le début de liste

}
/**
* @brief Supprime toutes les cartes que le joueurs doit
poser de sa main
* @param[in/out] l: le liste de joueur
* @param[in] lettre: la carte recherchée
*/
void supr_carte_inventaire(Liste&l, char lettre)
{
   unsigned int i = 0;
   while(l.enjeu[0]->Main[i].Nom != lettre)//Tant que la
lettre recherchée n'a pas été trouvé dans la main du
joueur
       i+=1;
   l.enjeu[0]->nb_carte-=1;
   unsigned int newtaille = l.enjeu[0]->nb_carte;
   Carte* nouv = new Carte [newtaille];//On réduit le
tableau correspondant à la main
   for(unsigned int j = 0; j < i; ++j)//On complète le
nouveau tableau de la main
       nouv[j] = l.enjeu[0]->Main[j];
   for(unsigned int k = i; k < l.enjeu[0]->nb_carte; ++k)
{
       nouv[k] = l.enjeu[0]->Main[k+1];
```

```
        i+=1;
    }
    delete [] l.enjeu[0]->Main;
    l.enjeu[0]->Main = nouv;
}
/**
* @brief Rends l'indice du prochain caractère d'une
chaine de caractère qui n'est pas un espace
* @param[in] a[]: la chaine de caractère
* @param[in] indice: l'indice dans la chaine de caractère
où la fonction va commencer
* @return l'indice du caractère trouvé
*/
unsigned int avance_str(char a[], unsigned int indice)
{
    unsigned tmp = indice;
    while(a[tmp] == ' ')//Tant que le caractère n'est pas
un espace
        tmp+=1;
    return tmp;//On retourne l'indice du prochain carctère
à prendre en compte
}
/**
* @brief Retorune l'indice du prochain carractère qui est
soit un esace soit l'arret de la chaine de caractère
* @param[in] a[]: la chaine de caractère
* @param[in] indice: l'indice dans la chaine de caractère
où la fonction va commencer
* @return l'indice du caractère trouvé
*/
unsigned int compte_droite(char a[], unsigned int indice)
{
    while(a[indice] != ' ' && a[indice] != '\0')//Tant que
le caractère n'est pas un espace ou que ce n'est pas la
fin de la chaine de caractère
        indice+=1;
    return indice;//on retourne l'indice où s'arrête le
caractère +1
}
/**
* @brief Calcule la taille d'un mot
```

```
* @param[in/out] a[] : le mot
* @return La taille d'un mot
*/
unsigned int taille_char(char a[])
{
   unsigned int tmp = 0;
   while(a[tmp] >= 'A' && a[tmp] <= 'Z' ){//Tant que les
caractères ne sont pas des lettres majuscules
       tmp+=1;
   }
   return tmp;//On retourne la taille du mot
}
/**
* @brief Remplace une des cartes choisi de l'inventaire
du joueur par un carte du talon et pose la carte du
joueur sur le tas de cartes exposées
* @param[in/out] l: le liste de joueur
* @param[in/out] expo: le tas de cartes exposées
* @param[in] carte: la carte que le joueur souhaite se
séparer
* @return true si le joueur possède bien la carte qu'il
souhaite remplacer sinon false
*/
bool Talon(Liste& l, Tas& pioche, Tas& expo,char carte)
{
   if(!detient(l,carte)) {//Vériffie que le joueur
possède la carte à échanger
       cout << "Coup invalide, recommencez" << endl;
       return false;//Le joueur recommence le tour
   }
   else
   {
       unsigned int newtaille = expo.nb_carte + 1;
       unsigned int indice =
indice_carte_inventaire(l,l.enjeu[0]->numero,carte);
       Carte* nouv = new Carte[newtaille];//On augmente
la taille du tableau de carte exposée
       nouv[0] = l.enjeu[0]->Main[indice];
       for (unsigned int j = 0; j < expo.nb_carte; ++j)
           nouv[j+1] = expo.carte[j];
       delete [] expo.carte;
```

```
        expo.carte = nouv;
        expo.nb_carte+=1;
        unsigned int emplacement =
indice_carte_inventaire(l,l.enjeu[0]->numero,carte);
        for(unsigned int i = emplacement; i <
l.enjeu[0]->nb_carte-1; ++i)//On déplace toutes les
cartes vers le début de la liste
            l.enjeu[0]->Main[i] = l.enjeu[0]->Main[i+1];
        l.enjeu[0]->Main[l.enjeu[0]->nb_carte-1] =
pioche.carte[0];
        if(pioche.nb_carte == 1)//S'il ne reste plus
qu'une carte dans le talon
        {
            unsigned int newtaille1 = expo.nb_carte-1;
            Carte* nouv1 = new Carte[newtaille1];//On crée
un nouveau tableau de talon avec le nombre de cartes du
tas de cartes exposées - 1
            for(unsigned int i = 0 ; i < newtaille1;
++i)//On distribue toutes les cartes de l'exposée vers le
talon sauf une
                nouv1[i] = expo.carte[i+1];
            delete [] pioche.carte;
            pioche.carte = nouv1;
            pioche.nb_carte = newtaille1;
            melanger(pioche);//On mélange le nouveau talon
            Carte* nouvexpo = new Carte[UN];//Créer le
nouveau tableau du tas de cartes exposées
            nouvexpo[0] = expo.carte[0];
            delete [] expo.carte;
            expo.carte = nouvexpo;
            expo.nb_carte = UN;
            return true;
        }
        else
        {
            unsigned int newtaille1 = pioche.nb_carte-UN;
            Carte* nouv1 = new Carte[newtaille1];//On
réduit la taille du tableau du talon
            for(unsigned int k = 0; k <
pioche.nb_carte-UN; ++k)
                nouv1[k] = pioche.carte[k+1];
```

```
            delete [] pioche.carte;
            pioche.carte = nouv1;
            pioche.nb_carte-=1;
            return true;//On passe au prochain tour
        }
    }

}
/**
* @brief Remplace une des cartes choisi de l'inventaire
du joueur par la carte exposée visible et inversement
* @param[in/out] l: le liste de joueur
* @param[in/out] expo: le tas de cartes exposées
* @param[in] carte: la carte que le joueur souhaite se
séparer
* @return true si le joueur possède bien la carte qu'il
souhaite remplacer sinon false
*/
bool exposee(Liste& l, Tas& expo, char carte)
{
    if(!detient(l,carte)) {//On vérifie que le joueur
possède la carte à échanger
        return false;
    }
    else
    {
        unsigned int indice =
indice_carte_inventaire(l,l.enjeu[0]->numero,carte);
        Carte tmp = l.enjeu[0]->Main[indice];
        for(unsigned int i = indice; i <
l.enjeu[0]->nb_carte-1; ++i)//On déplace toutes les
cartes vers le début de la liste
            l.enjeu[0]->Main[i] = l.enjeu[0]->Main[i+1];
        l.enjeu[0]->Main[l.enjeu[0]->nb_carte-1] =
expo.carte[0];
        expo.carte[0] = tmp;
        return true;
    }
}
/**
```

```
* @brief Pose un mot que le joueur à écirt si il détient
toutes les cartes et si le mot est dictionnaire
* @param[in/out] l: le liste de joueur
* @param[in/out] plateau: le plateau
* @param[in] mot[]: le mot que souhaite poser le joueur
* @param[in] taille: la taille du mot que le joueur veut
poser
* @param[in] dictionnaire: Le dictionnaire
* @param[in] indice_lettre_dico: tableau des indices des
lettres du dictionnaire pour optimiser la recherche
* @return true si le joueur à toutes les cartes pour
completer son mot (qu'il soit bon ou mauvais) sinon false
*/
bool poser(Liste& l, Plateau& plateau, char mot[],
unsigned int taille,const Dictionnaire&
dictionnaire,const int indice_lettre_dico[])
{
   for(unsigned int i = 0; i < taille; ++i)//Parcourt le
mot à poser
   {
       if(!detient(l,mot[i]))//Regarde si le joueur
possède les cartes
           return false;//Le joueur recommence
   }
   if
(!estMotValide(mot,dictionnaire,indice_lettre_dico,taille
))//Vérifie si le mot est valide
   {
       cout << "Mot invalide, vous passez votre tour" <<
endl;
       return true;//Le joueur passe son tour
   }
   unsigned int newtaille = plateau.nb_mots + UN;
   if(plateau.nb_mots == 0)//Vérifie s'il n'y a pas
encore de mot posé
       plateau.Mots = new Mots * [newtaille];//Créer le
tebleau de mot
   else
   {
       Mots** nouv = new Mots * [newtaille];//Agrandit le
tableau de mot
```

```
        for (unsigned int j = 0 ; j < plateau.nb_mots;
++j)
            nouv[j] = plateau.Mots[j];
        delete [] plateau.Mots;
        plateau.Mots = nouv;
    }
    plateau.nb_mots+=1;
    Mots* a = new Mots;//Créer un mot
    a->carte = new Carte[taille];//Créer un tableau de
carte
    plateau.Mots[plateau.nb_mots-1] = a;
    plateau.Mots[plateau.nb_mots-1]->nb_carte = 0;
    for (unsigned int k = 0; k < taille; ++k)//Insère le
mot sur le plateau
    {
        plateau.Mots[plateau.nb_mots-1]->carte[k].Nom =
mot[k];
        plateau.Mots[plateau.nb_mots-1]->nb_carte+=1;
    }
    for (unsigned int m = 0; m < taille; ++m)//Supprime
les cartes de l'inventaire du joueur
        supr_carte_inventaire(l,mot[m]);

    return true;//Le joueur a fini de jouer
}
/**
* @brief Remplace les lettres d'un mot si le joueur
possède les cartes qu'il veut poser,si le mot est pas
modifié et si le mot est dans le dictionnaire
* @param[in/out] l: le liste de joueur
* @param[in/out] plateau: le plateau
* @param[in] dictionnaire: Le dictionnaire
* @param[in] indice: l'indice où est stocké le mot que le
joueur souhaite modifié
* @param[in] mot[]: le mot que le joueur souhaite
remplacer l'ancien mot
* @param[in] indice_lettre_dico: tableau des indices des
lettres du dictionnaire pour optimiser la recherche
* @return true si le joueur à toutes les cartes pour
remplacer son mot (qu'il soit bon ou mauvais),et que le
```

```
joueur n'a pas mis plus de lettre que l'ancien mot sinon
false
*/
bool remplacer(Liste& l , Plateau& plateau,const
Dictionnaire& dictionnaire,unsigned int indice,char
mot[],const int indice_lettre_dico[])
{
   if(indice > plateau.nb_mots)//Regarde si l'indice
donné n'est pas invalide
       return false;//Le joueur recommence
   indice-=1;
   if(plateau.Mots[indice]->nb_carte !=
taille_char(mot))//Vérifie que le nouveau mot à la même
taille que l'ancien
       return false;//Le joueur recommence
   unsigned int tmp1 = 0;
   for(unsigned int i = 0; i <
plateau.Mots[indice]->nb_carte;++i)//Parcourt les lettres
du mot
   {
       if(plateau.Mots[indice]->carte[i].Nom !=
mot[i])//Regarde si les lettres ne sont pas les mêmes
       {
           if(!detient(l,mot[i]))//Vérifie si le joueur
ne possède pas la carte recherchée
               return false;//Le joueur recommence
           tmp1+=1;
       }
   }

if(!estMotValide(mot,dictionnaire,indice_lettre_dico,tail
le_char(mot)))//Vérifie si le mot est valide
   {
       cout << "Mot invalide, vous passez votre tour" <<
endl;
       l.enjeu[0]->NB_points+=3;
       return true;//Le joueur passe ton tour
   }
   char tmp[tmp1+1];
   unsigned int indicetmp = 0;
```

```
   for(unsigned int i = 0; i <
plateau.Mots[indice]->nb_carte;++i)//Parcourt le mot du
plateau
   {
       if(plateau.Mots[indice]->carte[i].Nom !=
mot[i])//Si les lettres ne sont pas les mêmes
       {
           //Cette boucle remplace les l'ancien mot par
le nouveau tout en sauvegardant les lettres qui sont
retirées dans tmp
           tmp[indicetmp] =
plateau.Mots[indice]->carte[i].Nom;
           indicetmp+=1;
           plateau.Mots[indice]->carte[i].Nom = mot[i];
           enleve_carte_inventaire(l,mot[i]);
       }
   }
   for(unsigned int j = 0; j <
plateau.Mots[indice]->nb_carte && tmp1 > 0;++j)//Parcourt
le mot
   {
       if(tmp[j] !=
plateau.Mots[indice]->carte[j].Nom){//Si la lettre dans
tmp et celle du mot ne sont pas la même

l.enjeu[0]->Main[l.enjeu[0]->nb_carte-tmp1].Nom = tmp[j];
           tmp1-=1;
       }
   }
   for (unsigned int i =
0;i<l.enjeu[0]->nb_carte;++i)//Donne aux lettres
récupérées par le joueur les points
   {
       if (l.enjeu[0]->Main[i].Nom ==
'A'||l.enjeu[0]->Main[i].Nom == 'E' ||
l.enjeu[0]->Main[i].Nom == 'I')
           l.enjeu[0]->Main[i].points = 10;
       else if (l.enjeu[0]->Main[i].Nom == 'B' ||
l.enjeu[0]->Main[i].Nom == 'F' || l.enjeu[0]->Main[i].Nom
== 'X' || l.enjeu[0]->Main[i].Nom == 'Z')
           l.enjeu[0]->Main[i].points = 2;
```

```
        else if (l.enjeu[0]->Main[i].Nom == 'D' ||
l.enjeu[0]->Main[i].Nom == 'J')
            l.enjeu[0]->Main[i].points = 6;
        else if (l.enjeu[0]->Main[i].Nom == 'G' ||
l.enjeu[0]->Main[i].Nom == 'Q' || l.enjeu[0]->Main[i].Nom
== 'Y')
            l.enjeu[0]->Main[i].points = 4;
        else
            l.enjeu[0]->Main[i].points = 8;
    }

    return true;

}
/**
* @brief Complète un mot si le joueur possède les cartes
qu'il veut poser pour compléter le mot par le mot qu'a
entré le joueur, si le mot est bien compléter et si le
mot est dans le dictionnaire
* @param[in/out] l: le liste de joueur
* @param[in/out] plateau: le plateau
* @param[in] dictionnaire: Le dictionnaire
* @param[in] indice: l'indice où est stocké le mot que le
joueur souhaite modifié
* @param[in] mot[]: le mot que le joueur souhaite
remplacer l'ancien mot
* @param[in] indice_lettre_dico: tableau des indices des
lettres du dictionnaire pour optimiser la recherche
* @param[in] taille : la taille du mot que le joueur à
complété
* @return true si le joueur à toutes les cartes pour
compléter son mot (qu'il soit bon ou mauvais),et que le
joueur n'a pas modifer l'ordre des lettres de l'ancien
mot sinon false
*/
bool completer(Liste& l, Plateau& plateau,const
Dictionnaire& dictionnaire, unsigned int indice, char
mot[],unsigned int taille,const int indice_lettre_dico[])
{
    if(indice > plateau.nb_mots)//Regarde si l'indice
donné n'est pas invalide
```

```
        return false;//Le joueur recommence
    indice-=1;
    if(plateau.Mots[indice]->nb_carte >= taille)//Vérifie
si la taille du mot posée n'est pas inférieure ou égale à
la taille du nouveau mot
        return false;//Le joueur recommence
    unsigned int indice_mot_nouv[taille];
    unsigned int nouvindice = 0;
    for(unsigned int i = 0; i <
plateau.Mots[indice]->nb_carte;++i)//Parcourt le mot sur
le plateau
    {
        bool stop = true;
        for(unsigned int j = 0; j < taille &&  stop;
++j)//Parcourt le nouveau mot tant que lettres qui sont
dans le mot du plateau n'ont pas changé de place (Par
exemple : que main ne devienne pas ainms

        {
            if(plateau.Mots[indice]->carte[i].Nom ==
mot[j])
            {
                indice_mot_nouv[nouvindice] = j;
                stop = false;
                if(nouvindice != 0 &&
indice_mot_nouv[nouvindice] <
indice_mot_nouv[nouvindice-1])
                {
                    return false;//Le joueur recommence
                }
                nouvindice+=1;
            }

        }
    }
    unsigned int tmp = 0;
    for(unsigned int i = 0; i < taille; ++i)//Parcourt le
nouveau mot
    {
        if(i != indice_mot_nouv[tmp])//Si la lettre ne
fait pas partie des lettres du mot sur le plateau
```

```
        {
            if(!detient(l,mot[i]))//Vérifie si le joueur
n'a pas cette lettre
                return false;//Le joueur recommence
        }
        else
            tmp+=1;
    }

if(!estMotValide(mot,dictionnaire,indice_lettre_dico,tail
le_char(mot)))//Vérifie si le mot est valide
    {
        cout << "Mot invalide, vous passez votre tour" <<
endl;
        l.enjeu[0]->NB_points+=3;
        return true;//Le joueur passe son tour
    }
    tmp = 0;
    for(unsigned int i = 0; i < taille; ++i)//Parcourt le
nouveau mot
    {
        if(i != indice_mot_nouv[tmp])//Si la lettre ne
fait pas partie des lettres du mot sur le plateau
        {
            supr_carte_inventaire(l,mot[i]);//Supprime la
carte de la main du joueur
        }
        else
            tmp+=1;
    }
    unsigned int newtaille = taille;
    Carte* nouv = new Carte[newtaille];//Change la taille
du tableau contenant le mot du plateau
    for(unsigned int i = 0; i < newtaille; ++i)
    {
        nouv[i].Nom = mot[i];
    }
    plateau.Mots[indice]->nb_carte = newtaille;
    delete [] plateau.Mots[indice]->carte;
    plateau.Mots[indice]->carte = nouv;
    return true;
```

```
}
/**
* @brief Lance les différentes commandes en fonction de
ce qu'a mis le joueur
* @param[in/out] l: le liste de joueur
* @param[in/out] pioche: le talon
* @param[in/out] expo : le tas de cartes exposées
* @param[in/out] plateau: le plateau
* @param[in] dictionnaire: Le dictionnaire
* @param[in] indice_lettre_dico: tableau des indices des
lettres du dictionnaire pour optimiser la recherche
*/
void jouer( Liste& l, Tas& pioche, Tas& expo, Plateau&
plateau,const Dictionnaire& dictionnaire,const int
indice_lettre_dico[])
{
   char commande[MAX];//La chaine de caractère qui va
récupérer les entrées des joueurs
   unsigned int indice = 1;//L'endroit où se trouve la
flèche pour récupérer les caractères dans commande
   bool bon;//Booléen pour savoir si le tour peut
continuer
   affiche_jeu(l,expo,plateau);
   fgets(commande, MAX, stdin);
   size_t len;
   size_t length = 0;
   while (commande[length] != '\0')
   {
       length++;
   }
   len = length;
   if (len > 0 && commande[len - 1] == '\n') {
       commande[len - 1] = '\0';
   }

   if((commande[0] != 'T' && commande[0] != 'E' &&
commande[0] != 'P' && commande[0] != 'R' && commande[0]
!= 'C') || (commande[1] != ' ') )//Vérifie si la première
lettre n'est pas une des commandes ou si la chaine de
caractère s'arrête juste après
       cout << "Coup invalide, recommencez" << endl;
```

```
   else
   {
        if ( commande[0] == 'T')
        {
            indice = avance_str(commande,indice);
            if(commande[indice] == '\0' ||
(commande[indice+1] != '\0' && commande[indice+1] != '
'))//S'il n'y a pas lettre après l'espace
                cout << "Coup invalide, recommencez" <<
endl;
            else
            {
                bon =
Talon(l,pioche,expo,commande[indice]);
                if(bon)//Si le tour peut continuer
                    next(l);
                else
                    cout << "Coup invalide, recommencez"
<< endl;
            }
        }
        else if (commande[0] == 'E')
        {
            indice = avance_str(commande,indice);
            if(commande[indice] == '\0' ||
(commande[indice+1] != '\0' && commande[indice+1] != '
'))//S'il n'y a pas lettre après l'espace
                cout << "Coup invalide, recommencez" <<
endl;
            else
            {
                bon = exposee(l,expo,commande[indice]);
                if(bon)//Si le tour peut continuer
                    next(l);
                else
                    cout << "Coup invalide, recommencez"
<< endl;
            }
        }
        else if (commande[0] == 'P')
        {
```

```
            indice = avance_str(commande,indice);
            unsigned int tmp = indice;
            while(commande[tmp] != ' ' && commande[tmp] !=
'\0')//Tant que ce n'est pas un espace où la fin de la
chaine de caractère
                tmp+=1;
            if(commande[indice] == '\0' &&
commande[indice+1] != '\0')//Si il n'y a pas de mot
                cout << "Coup invalide, recommencez" <<
endl;
            else
            {
                char commande1[tmp-indice+UN];
                for (unsigned int i = 0; i < tmp-indice;
++i)//Stock le mot dans commande1
                    commande1[i] = commande[indice+i];
                bon =
poser(l,plateau,commande1,tmp-indice,dictionnaire,indice_
lettre_dico);
                if(bon)//Si le tour peut continuer
                    next(l);
                else
                    cout << "Coup invalide, recommencez"
<< endl;
            }
        }
        else if (commande[0] == 'R' || commande[0] == 'C')
        {
            indice = avance_str(commande,indice);
            unsigned int numero;//Le numéro du mot à
modifier
            unsigned int droite =
compte_droite(commande,indice);
            char tmp[droite-indice];
            unsigned int tmpindice = 0;
            if(commande[indice] != '1' && commande[indice]
!= '2' && commande[indice] != '3' && commande[indice] !=
'4' && commande[indice] != '5' && commande[indice] != '6'
&& commande[indice] != '7' && commande[indice] != '8' &&
commande[indice] != '9')//Si le caractère après les
espaces n'est pas un chiffre(à part 0)
```

```
            {
                cout << "Coup invalide, recommencez" <<
endl;
            }
            else
            {
                tmp[tmpindice] = commande[indice];//chaine
de caractère qui va contenir le nombre
                tmpindice+=1;
                indice+=1;
                bool ok = true;
                if(indice < droite)//S'il y a encore des
caractères après le premier chiffre
                {
                    for(unsigned int i = indice; i <
droite; ++i)//On stocke les chiffres dans tmp
                    {
                        if(commande[i] != '0' &&
commande[i] != '1' && commande[i] != '2' && commande[i]
!= '3' && commande[i] != '4' && commande[i] != '5' &&
commande[i] != '6' && commande[i] != '7' && commande[i]
!= '8' && commande[i] != '9')//Si le caractère après les
espaces n'est pas un chiffre
                        {
                            ok = false;
                        }
                        tmp[tmpindice] = commande[i];
                        tmpindice+=1;
                        indice+=1;
                    }
                }
                if(ok)
                {
                    cin.putback('\0');//On envoie dans le
buffer les chiffres
                    for(unsigned int i = tmpindice; i > 0;
--i){
                        cin.putback(tmp[i-1]);
                    }
                    cin >> numero;
                    if(commande[droite] != ' ')
```

```
                        cout << "Coup invalide,
recommencez" << endl;
                    else
                    {
                        indice =
avance_str(commande,indice);
                        unsigned int tmp1 = indice;
                        while(commande[tmp1] != ' ' &&
commande[tmp1] != '\0')//Tant qu'il n'y a pas un espace
ou la fin de la chaine de caractère
                            tmp1+=1;
                        char mot[tmp1-indice+1];
                        for (unsigned int i = 0; i <
tmp1-indice; ++i)//On stock le mot
                            mot[i] = commande[indice+i];
                        if(commande[0] == 'R')
                        {
                            bon =
remplacer(l,plateau,dictionnaire,numero,mot,indice_lettre
_dico);
                            while (std::cin.get() != '\0')
{
                                // vide le buffer
                            }
                            if(bon)//Si le tour peut
continuer
                                next(l);
                            else
                                cout << "Coup invalide,
recommencez" << endl;
                        }
                        else
                        {
                            bon =
completer(l,plateau,dictionnaire,numero,mot,tmp1-indice,i
ndice_lettre_dico);
                            while (std::cin.get() != '\0')
{
                                // vide le buffer
                            }
```

```
                            if(bon)//Si le tour peut
continuer
                                next(l);
                            else
                                cout << "Coup invalide,
recommencez" << endl;
                        }

                    }

                }
                else
                    cout << "Coup invalide, recommencez"
<< endl;
            }
        }
    }

}
```

Main (avec les test unitaires au dessus)

```
#include "Jeu.h"
#include "Joueurs.h"
#include "Plateau.h"
#include <iostream>
#include <cassert>

void Test_unitaires(Liste& l,Tas& pioche,Tas&
expose,Plateau& plateau,Dictionnaire & dictionnaire, int
indice_lettre_dico[])//Fonction regroupant tout le test
unitaires
{
   //Vérifie que la liste enjeu s'actualise bien, que les
joueurs sont bien supprimé et que le tour commence bien
avec le bon nombre
   /*
   unsigned int i = l.enjeu[0]->numero;
   next(l);
   assert(i+1 == l.enjeu[0]->numero);
   l.enjeu[0]->NB_points+=100;
   unsigned int tmp = 2;
```

```
   unsigned int* tmp1 = &tmp;
   elimination(l,l.Taille_enjeu,tmp1);
   assert(*l.Taille_enjeu == 3);
   assert(l.enjeu[0]->numero == 4);
   cout << "bon" << endl;*/
   //Veriffie si les lettres sont bien comptées
   /*for(unsigned int i = 0; i <
l.enjeu[0]->nb_carte;++i)
   {
       l.enjeu[0]->Main[i].Nom = 'A';
       l.enjeu[0]->Main[i].points = 10;
   }
   compte_points(l,l.Taille_enjeu);
   assert(l.enjeu[0]->NB_points == 100);*/

   //Veriffie si la fonction Talon fonctione bien
   /*l.enjeu[0]->Main[8].Nom = 'Z';
   l.enjeu[0]->Main[8].points = 2;
   for(unsigned int i = 0;i < 8;++i)
   {
       l.enjeu[0]->Main[i].Nom = 'A';
       l.enjeu[0]->Main[i].points = 10;
   }
   l.enjeu[0]->Main[9].Nom = 'A';
   l.enjeu[0]->Main[9].points = 10;
   char lettre = 'B';
   assert(!Talon(l,pioche,expose,lettre));
   lettre = 'Z';
   char lettretalon = pioche.carte[0].Nom;
   assert(Talon(l,pioche,expose,lettre));
   assert(l.enjeu[0]->Main[9].Nom == lettretalon);
   assert(expose.carte[0].Nom == 'Z');*/

   //Verifie si la fonction expose fonctione bien
   /*
   l.enjeu[0]->Main[8].Nom = 'Z';
   l.enjeu[0]->Main[8].points = 2;
   for(unsigned int i = 0;i < 8;++i)
   {
       l.enjeu[0]->Main[i].Nom = 'A';
       l.enjeu[0]->Main[i].points = 10;
```

```
    }
    l.enjeu[0]->Main[9].Nom = 'A';
    l.enjeu[0]->Main[9].points = 10;
    char lettre = 'B';
    assert(!exposee(l,expose,lettre));
    lettre = 'Z';
    char lettreexpose = expose.carte[0].Nom;
    assert(exposee(l,expose,lettre));
    assert(l.enjeu[0]->Main[9].Nom == lettreexpose);
    assert(expose.carte[0].Nom == 'Z');
    */
    //Verifie si poser fonctionne bien
    /*
    char mot[4] = "IYL";

assert(!poser(l,plateau,mot,3,dictionnaire,indice_lettre_
dico));
    assert(l.enjeu[0]->nb_carte == 10);
    assert(plateau.nb_mots == 0);
    l.enjeu[0]->Main[0].Nom = 'I';
    l.enjeu[0]->Main[5].Nom = 'L';
    l.enjeu[0]->Main[9].Nom = 'Y';
    l.enjeu[0]->Main[0].points = 10;
    l.enjeu[0]->Main[5].points = 8;
    l.enjeu[0]->Main[9].points = 4;

assert(poser(l,plateau,mot,3,dictionnaire,indice_lettre_d
ico));
    assert(l.enjeu[0]->NB_points == 3);
    assert(l.enjeu[0]->nb_carte == 10);
    assert(plateau.nb_mots == 0);
    char mot1[3] = "IL";

assert(poser(l,plateau,mot1,2,dictionnaire,indice_lettre_
dico));
    assert(l.enjeu[0]->nb_carte == 8);
    assert(plateau.nb_mots == 1);*/

    //Verifie si la fonction Remplacer
    /*
    l.enjeu[0]->Main[9].Nom = 'T';
```

```
   l.enjeu[0]->Main[8].Nom = 'E';
   l.enjeu[0]->Main[9].points = 8;
   l.enjeu[0]->Main[8].points = 10;
   l.enjeu[0]->Main[0].Nom = 'E';
   l.enjeu[0]->Main[1].Nom = 'S';
   l.enjeu[0]->Main[0].points = 10;
   l.enjeu[0]->Main[1].points = 8;
   for(unsigned int i = 2; i < 8; ++i)
   {
       l.enjeu[0]->Main[i].Nom = 'A';
       l.enjeu[0]->Main[i].points = 10;
   }
   char base[3] = "ES";

poser(l,plateau,base,2,dictionnaire,indice_lettre_dico);
   char mot[3] = "EN";

assert(!remplacer(l,plateau,dictionnaire,2,mot,indice_let
tre_dico));

assert(!remplacer(l,plateau,dictionnaire,1,mot,indice_let
tre_dico));
   char mot1[3] = "EA";

assert(remplacer(l,plateau,dictionnaire,1,mot1,indice_let
tre_dico));
   assert(l.enjeu[0]->NB_points == 3);
   char mot2[3] = "ET";

assert(remplacer(l,plateau,dictionnaire,1,mot2,indice_let
tre_dico));
   assert(l.enjeu[0]->Main[7].Nom == 'S');
   */

   //Verifie si completer fonctionne
   /*
   l.enjeu[0]->Main[9].Nom = 'T';
   l.enjeu[0]->Main[8].Nom = 'E';
   l.enjeu[0]->Main[9].points = 8;
   l.enjeu[0]->Main[8].points = 10;
   l.enjeu[0]->Main[0].Nom = 'E';
```

```
    l.enjeu[0]->Main[1].Nom = 'S';
    l.enjeu[0]->Main[0].points = 10;
    l.enjeu[0]->Main[1].points = 8;
    for(unsigned int i = 2; i < 8; ++i)
    {
        l.enjeu[0]->Main[i].Nom = 'A';
        l.enjeu[0]->Main[i].points = 10;
    }
    char base[3] = "ES";

poser(l,plateau,base,2,dictionnaire,indice_lettre_dico);
    char mot[4] = "Kes";

assert(!completer(l,plateau,dictionnaire,1,mot,3,indice_l
ettre_dico));
    char mot1[4] = "AES";

assert(completer(l,plateau,dictionnaire,1,mot1,3,indice_l
ettre_dico));
    assert(l.enjeu[0]->NB_points == 3);
    char mot2[4] = "EST";

assert(completer(l,plateau,dictionnaire,1,mot2,3,indice_l
ettre_dico));
    assert(l.enjeu[0]->nb_carte == 7);*/


}

int main(int argc, const char* argv[]) {

    int nb_joueur = atoi(argv[1]);
    if(nb_joueur <  2 || nb_joueur > 4)//Vérifie que le
nombre de joueur est d'au moins 2 et au max 4
    {
        return 1;
    }
    Dictionnaire dictionnaire;
    int nb_mot_dictionnaire = 0;
    dictionnaire.nbmots = &nb_mot_dictionnaire;
    int indice_lettre_dico[MAXTAILLEMOT+1];
```

```
   indice_lettre_dico[0] = 0;

if(chargerDictionnaire("ods4.txt",dictionnaire,indice_let
tre_dico) < 0)
        return 1;
   Liste l;
   l.Taille= nb_joueur;
   l.Taille_enjeu = &nb_joueur;
   initialiser_liste_joueur(l,l.Taille);
   cout << "(Commandes valides : TEPRC)" << endl;
   unsigned int com = 0;
   unsigned int* commence = &com;
   cin.putback('\0');
   while (std::cin.get() != '\0') {
        // vide le buffer
   }
   /*bool unit_test = true;*/
   while(partie(l)/*unit_test*/)
   {
        /*unit_test = false;*/
        Tas pioche;
        pioche.nb_carte = NB;
        pioche.carte = new Carte [NB];
        Tas expose;
        expose.nb_carte = UN;
        expose.carte = new Carte [UN];
        Plateau plateau;
        initialiser_carte(pioche);
        melanger(pioche);
        distribuer(pioche,l,expose);
        initialiser_plateau(plateau);
        unsigned int i;
        do
        {
            cout << endl;
            i = l.enjeu[0]->numero;

jouer(l,pioche,expose,plateau,dictionnaire,indice_lettre_
dico);
        }
```

```
        while(l.liste_joueur[i-1]->nb_carte != 0);//Tant
qu'au tous les joueurs ont des cartes

Test_unitaires(l,pioche,expose,plateau,dictionnaire,indic
e_lettre_dico);
        compte_points(l,l.Taille_enjeu);
        affiche_tour(l);
        elimination(l,l.Taille_enjeu,commence);
        del_tas(pioche.carte);
        del_tas(expose.carte);
        if(plateau.nb_mots > 0)
            delet_plateau(plateau);
    }
    cout << endl << "La partie est finie";
    delete [] dictionnaire.dictionnaire;
    dictionnaire.dictionnaire = nullptr;
    del_joueur(l,nb_joueur);
    return 0;
}
```