DOSSIER DE DÉVELOPPEMENT SAÉ 1.02 Comparaison d'approches algorithmiques

Mouhamed BSIKRI Groupe 105

Nassim Maatougui Groupe 103

Table des matières

I – Présentation du Projet

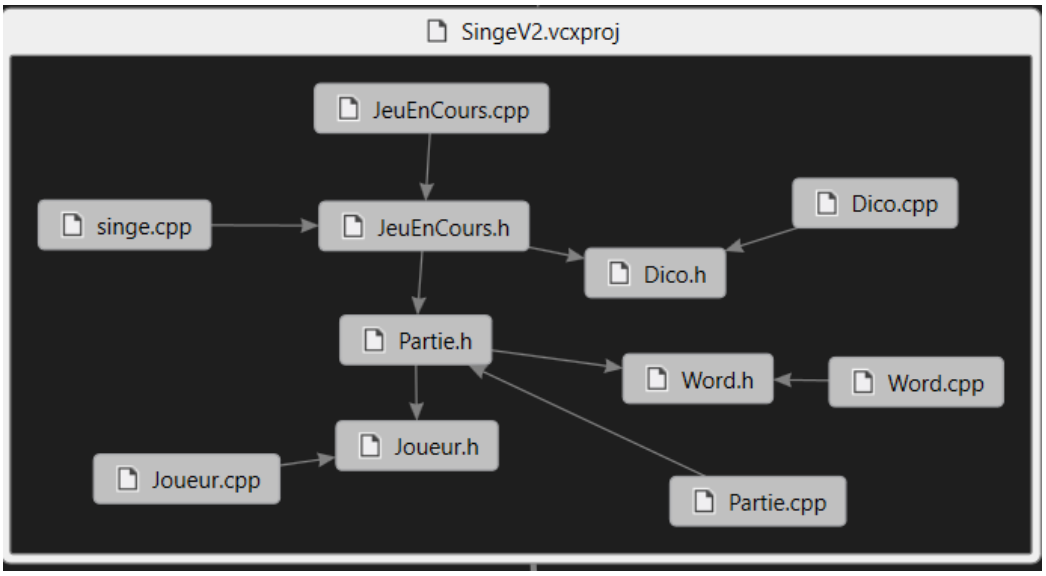
L'application que nous avons développée permet à plusieurs joueurs de jouer une partie de quart de singe. En effet, elle permet à l'utilisateur d'entrer des joueurs, humains ou robots, via la ligne de commande, en entrant NomExecutable.exe HHHH pour 4 humains par exemple, afin qu'ils puissent jouer au jeu du quart de singe. Le jeu consiste à permettre aux joueurs d'entrer chacun leur tour une lettre afin de compléter une suite de lettres pouvant former un mot. Le joueur veille à compléter le mot avec une lettre cohérente mais à ne pas compléter un mot existant dans un fichier texte représentant un dictionnaire. S'il complète le mot, il perd la manche et écope d'un quart de singe. Par conséquent, au bout de quatre quarts de singe accumulés par un joueur, la partie se termine.

D'une part, l'humain peut ajouter des lettres pour compléter un mot, abandonner la manche (et la perdre), et interroger le joueur précédent afin d'inviter ce dernier à saisir le mot auquel il pense, selon les règles du jeu du quart de singe. Lorsqu'il est interrogé, le joueur doit saisir un mot existant de tel sorte à gagner la manche, la perdant au cas inverse. D'autres part, le joueur robot est un joueur surpuissant programmé de tel sorte à tout le temps compléter un mot existant dans le dictionnaire. Si le mot n'existe pas, le robot interroge le joueur précédent, ce dernier perdant forcément la manche. Par conséquent, le robot perd seulement lorsqu'il complète un mot existant dans le dictionnaire.

Afin d'assurer un travail et une production efficace ainsi qu'organiser, nous avons développé l'application sur Visual Studio 2019. Nous avons opté, dès le début du développement de notre application, par la séparation de notre code en une multitude de composants (fichiers .h et .cpp) selon l'intérêt de chaque fonction, structures de données et autres. En effet nous avons utilisé le principe de la compilation séparée consistant à séparer notre application en plusieurs fichiers sources afin de rendre la compilation plus rapide ainsi que de permettre le bon déroulement du travail en équipe. Cela a permis de bien structurer notre code de manière à le rendre lisible et agréable mais aussi de manière à repérer et corriger les erreurs facilement. Par ailleurs, nous communiquions via Discord ou durant les créneaux dédiés à la SAE afin de mettre en commun notre avancement mais aussi nos difficultés.

II – Graphes de dépendances

Nous avons généré le graphe de dépendances grâce à l'outil architecture de Visual Studio Entreprise 2019, qui permet de générer un graphe de dépendances.



III – Organisation des tests et bilan de validation

Dans un premier temps, nous avons développé les cours programmes ci-dessous dans lesquels nous avons testé nos différents composants, en l'occurrence Joueur, Dico et Word. Cela nous a permis de s'assurer qu'ils fonctionnaient bien. En effet, du fait que ce sont nos composants de bases, s'ils ne fonctionnent pas correctement, alors le jeu ne pourra pas être construit. Nous testions régulièrement ces cours programmes, de tel sorte à s'assurer qu'ils fonctionnent tout au long du développement après adaptation de ces premiers.

```
/**
 * @file TestDico.cpp
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 1 28/12/2022
 * Test of dictionary component
 */


#include <iostream>
#include <iomanip> // pour setw
#include <cctype>
#include <cassert>
#include "Dico.h"
using namespace std;


int main() {
	Dictionnaire dico;
	uint capa = 350000;
	double pas = 1.25;
	Initialiser_Dico(dico,capa,pas);
	Ecrire_Mots_Dico(dico);

	const char* test = "TEST";
	assert(Recherche_Mot_Dico_Dichoto(dico.contenu, dico.nombreMots, test) == true);

	const char* testfail = "TESTFAIL";
	assert(Recherche_Mot_Dico_Dichoto(dico.contenu, dico.nombreMots, testfail) ==
false);

	Detruire_Dico(dico);
}
```

```
/**
 * @file TestWord.cpp
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 1 2/12/2022
 * Test du composant Word
 */


#include <iostream>
#include <iomanip> // pour setw
#include <cctype>
#include <cassert>
#include "Word.h"
using namespace std;

int main() {
	Word mot;
	Nouveau_Mot(mot);
	assert(mot.word[0] == '\0');
	assert(mot.lenword == 0);

	const int constante = 4;
	char lettre;
	cout << "entrez cinq lettres" << endl;
	for (int i = 0; i < constante; ++i) {
		cin >> lettre;
		Add_Letter(mot, lettre);
		cout << mot.word[i] << endl;
		cout << mot.lenword << endl << i;
		assert(mot.word[i] == toupper(lettre));
		assert(mot.lenword == i+1);
	}

	Word mottest;
	Nouveau_Mot(mottest);
	Add_Letter(mottest, 'P');
	Add_Letter(mottest, 'R');
	Add_Letter(mottest, 'I');
	Add_Letter(mottest, 'S');
	Add_Letter(mottest, 'O');
	Add_Letter(mottest, 'N');

	assert(strcmp(mottest.word, "PRISON") == 0);
	assert(mottest.lenword == 6);
	assert(strcmp(mottest.word, "MAMAN") != 0);

	char mottestcompare[26] = "PRISON";
	assert(Compare_Mot(mottest, mottestcompare) == true);

	char mottestcompare2[26] = "PRISONFLEURY";
	assert(Compare_Mot(mottest, mottestcompare2) == true);

	char mottestcompare3[26] = "PRIS";
	assert(Compare_Mot(mottest, mottestcompare3) == false);

	char mottestcompare4[26] = "PRAGMATIQUE";
```

```
	assert(Compare_Mot(mottest, mottestcompare4) == false);

	Detruire_Mot_Manche(mot);
	Detruire_Mot_Manche(mottest);
}
```

```
/**
 * @file TestJoueur.cpp
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 1 31/12/2022
 * test du composant Joueur
 */

#include <iostream>
#include <iomanip> // pour setw
#include <fstream>
#include <cctype>
#include <cassert>
#include "Joueur.h"
using namespace std;

int main() {
	Joueur joueur;
	joueur.nature = 'H';
	joueur.numerojoueur = 1;
	joueur.quartDeSinge = 0;

	assert(A_Joueur_Singe(joueur) == false);

	Ajout_Quart_de_Singe(joueur);
	assert(A_Joueur_Singe(joueur) == false);
	Ajout_Quart_de_Singe(joueur);
	assert(A_Joueur_Singe(joueur) == false);
	Ajout_Quart_de_Singe(joueur);
	assert(A_Joueur_Singe(joueur) == false);
	Ajout_Quart_de_Singe(joueur);
	assert(A_Joueur_Singe(joueur) == true);

}
```

Dans un second temps, nous avons testé le jeu du quart de singe uniquement avec des joueurs humains. En effet, nous avons proposé à des camarades de jouer avec nous pour tester le jeu dans des conditions réelles. Cela à permis de vérifier que les différents cas possibles durant le déroulement de la partie fonctionnaient correctement. Cependant, pour nous assurer du bon affichage des messages prédéfinis pour le jeu, nous avons redirigé le fichier in.txt de l'enseignant vers un fichier res.txt que nous avons comparé avec le fichier out.txt.

Soit la commande : Singe.txt HHHHHH <in.txt> res.txt
fc res.txt out.txt

A
D

V
E
R
S
I
T
E

1H, () > 2H, (A) > 3H, (AD) > 4H, (ADV) > 5H, (ADVE) > 6H, (ADVER) > 1H, (ADVERS) > 2H, (ADVERSI) > 3H, (ADVERSIT) > le mot ADVERSITE existe, le joueur 3H prend un quart de singe
1H : 0; 2H : 0; 3H : 0.25; 4H : 0; 5H : 0; 6H : 0

A
G
H
Q
?
AGI

3H, () > 4H, (A) > 5H, (AG) > 6H, (AGH) > 1H, (AGHQ) > 6H, saisir le mot > le mot AGI ne commence pas par les lettres attendues, le joueur 6H prend un quart de singe
1H : 0; 2H : 0; 3H : 0.25; 4H : 0; 5H : 0; 6H : 0.25

A
T
C
H
O
U
?
ATCHOUM

() > 1H, (A) > 2H, (AT) > 3H, (ATC) > 4H, (ATCH) > 5H, (ATCHO) > 6H, (ATCHOU) > 5H, saisir le mot > le mot ATCHOUM existe, le joueur 6H prend un quart de singe
1H : 0; 2H : 0; 3H : 0.25; 4H : 0; 5H : 0; 6H : 0.5

A
T
O
C
!

() > 1H, (A) > 2H, (AT) > 3H, (ATO) > 4H, (ATOC) > le joueur 4H abandonne la manche et prend un quart de singe
1H : 0; 2H : 0; 3H : 0.25; 4H : 0.25; 5H : 0; 6H : 0.5

A
D
S
O
R
B
I
O
N
?
ADSORBIONS

() > 5H, (A) > 6H, (AD) > 1H, (ADS) > 2H, (ADSO) > 3H, (ADSOR) > 4H, (ADSORB) > 5H, (ADSORBI) > 6H, (ADSORBIO) > 1H, (ADSORBION) > 6H, saisir le mot > le mot ADSORBIONS existe, le joueur 1H prend un quart de singe
1H : 0.25; 2H : 0; 3H : 0.25; 4H : 0.25; 5H : 0; 6H : 0.5

A
L
V
E
O
L
I
T
D
?
ALVEOLITE

() > 2H, (A) > 3H, (AL) > 4H, (ALV) > 5H, (ALVE) > 6H, (ALVEO) > 1H, (ALVEOL) > 2H, (ALVEOLI) > 3H, (ALVEOLIT) > 4H, (ALVEOLITD) > 3H, saisir le mot > le mot ALVEOLITE ne commence pas par les lettres attendues, le joueur 3H prend un quart de singe
1H : 0.25; 2H : 0; 3H : 0.5; 4H : 0.25; 5H : 0; 6H : 0.5

A
T
Y
P
I
Q
U
!

() > 4H, (A) > 5H, (AT) > 6H, (ATY) > 1H, (ATYP) > 2H, (ATYPI) > 3H, (ATYPIQ) > 4H, (ATYPIQU) > le joueur 4H abandonne la manche et prend un quart de singe
1H : 0.25; 2H : 0; 3H : 0.5; 4H : 0.5; 5H : 0; 6H : 0.5

A
F
A
P
?
AFAT

() > 5H, (A) > 6H, (AF) > 1H, (AFA) > 2H, (AFAP) > 1H, saisir le mot > le mot AFAT ne commence pas par les lettres attendues, le joueur 1H prend un quart de singe
1H : 0.5; 2H : 0; 3H : 0.5; 4H : 0.5; 5H : 0; 6H : 0.5

A
C
C
L
I
M
A
T
O
N
!

() > 2H, (A) > 3H, (AC) > 4H, (ACC) > 5H, (ACCL) > 6H, (ACCLI) > 1H, (ACCLIM) > 2H, (ACCLIMA) > 3H, (ACCLIMAT) > 4H, (ACCLIMATO) > 5H, (ACCLIMATON) > le joueur 5H abandonne la manche et prend un quart de singe
1H : 0.5; 2H : 0; 3H : 0.5; 4H : 0.5; 5H : 0.25; 6H : 0.5

A
V
O

C
A
S
S
I
E
X
?
AVOCASSIEZ

() > 6H, (A) > 1H, (AV) > 2H, (AVO) > 3H, (AVOC) > 4H, (AVOCA) > 5H, (AVOCAS) > 6H, (AVOCASS) > 1H, (AVOCASSI) > 2H, (AVOCASSIE) > 3H, (AVOCASSIEX) > 2H, saisir le mot > le mot AVOCASSIEZ ne commence pas par les lettres attendues, le joueur 2H prend un quart de singe
1H : 0.5; 2H : 0.25; 3H : 0.5; 4H : 0.5; 5H : 0.25; 6H : 0.5

A
T
A
M
A
?
ATAMAN

() > 3H, (A) > 4H, (AT) > 5H, (ATA) > 6H, (ATAM) > 1H, (ATAMA) > 6H, saisir le mot > le mot ATAMAN existe, le joueur 1H prend un quart de singe
1H : 0.75; 2H : 0.25; 3H : 0.5; 4H : 0.5; 5H : 0.25; 6H : 0.5

A
C
U
T
A
N
G
L
!

() > 2H, (A) > 3H, (AC) > 4H, (ACU) > 5H, (ACUT) > 6H, (ACUTA) > 1H, (ACUTAN) > 2H, (ACUTANG) > 3H, (ACUTANGL) > le joueur 3H abandonne la manche et prend un quart de singe
1H : 0.75; 2H : 0.25; 3H : 0.75; 4H : 0.5; 5H : 0.25; 6H : 0.5

A
P
N
E
I
Q
U
E

() > 4H, (A) > 5H, (AP) > 6H, (APN) > 1H, (APNE) > 2H, (APNEI) > 3H, (APNEIQ) > 4H, (APNEIQU) > le mot APNEIQUE existe, le joueur 4H prend un quart de singe
1H : 0.75; 2H : 0.25; 3H : 0.75; 4H : 0.75; 5H : 0.25; 6H : 0.5

A
I
O
L
?
AIOLI

() > 5H, (A) > 6H, (AI) > 1H, (AIO) > 2H, (AIOL) > 1H, saisir le mot > le mot AIOLI existe, le joueur 2H prend un quart de singe
1H : 0.75; 2H : 0.5; 3H : 0.75; 4H : 0.75; 5H : 0.25; 6H : 0.5

A
B
E
L
I
E
C
?
ABELIEN

() > 3H, (A) > 4H, (AB) > 5H, (ABE) > 6H, (ABEL) > 1H, (ABELI) > 2H, (ABELIE) > 3H, (ABELIEC) > 2H, saisir le mot > le mot ABELIEN ne commence pas par les lettres attendues, le joueur 2H prend un quart de singe
1H : 0.75; 2H : 0.75; 3H : 0.75; 4H : 0.75; 5H : 0.25; 6H : 0.5

A
A
K
?
AAS

() > 3H, (A) > 4H, (AA) > 5H, (AAK) > 4H, saisir le mot > le mot AAS ne commence pas par les lettres attendues, le joueur 4H prend un quart de singe
1H : 0.75; 2H : 0.75; 3H : 0.75; 4H : 1; 5H : 0.25; 6H : 0.5
La partie est finie

Enfin, nous avons ajouté des robots petits à petits dans nos parties de quarts de singe. Le robot étant assez imprévisible, il fut assez compliqué de le tester. Toutefois, en regardant directement ses interactions, on pouvait savoir s'il fonctionnait comme souhaité. Par ailleurs, nous avons testé des parties entièrement avec des robots. En conclusion, nos humains et nos robots fonctionnent parfaitement et peuvent dérouler une partie de quart de singe de manière à finir la partie dans les règles.

IV – Bilan du projet

1- Difficultés rencontrées

Dans un premier temps, la mise en place de la séparation de nos fonctions, structures, constantes et autres dans des fichiers (.cpp/.h) ne fut pas chose moindre. En effet, puisque nous étions en début du développement, fraichement instruits de la compilation séparée, il a été assez compliqué d'y procéder. Nous n'avions pas forcément compris comment disposer les fonctions dans les fichiers .h et .cpp, de tel sorte qu'on puisse les retrouver facilement, en raison du fait qu'elles étaient parfois assez compliquées et faisaient appel à d'autres fonctions dans d'autres fichiers. Pour remédier à cela, nous avons questionné les chargés de TD sur la manière la plus efficace de former nos .h et .cpp. Nous avons aussi trié les fonctions simples associées à chaque structure de données. Nous avons alors remarqué que les fonctions appelant d'autres fonctions étaient celles permettant le déroulement du jeu. Nous leur avons alors dédier un .h à part. Aussi, nous avons entrepris la minimisation de l'utilisation d'inclusions inutiles, pour une meilleure rapidité de compilation.

Ensuite, l'intégration de notre fichier.txt, représentant le dictionnaire, dans un tableau de chaînes de caractères n'a pas été simple lors des allocations mémoire. En effet, nous nous sommes inspirés du conteneurTDE vus en cours pour stocker les mots du dictionnaire dans un tableau de chaînes de caractères. Or, d'une part, le fait de devoir lire dans le fichier, allouer une nouvelle chaîne de caractère de manière dynamique, copier le mot du fichier dans cette nouvelle chaîne et faire pointer la dernière chaîne de caractères du tableau (représentant le dictionnaire) vers la chaîne de caractère dynamique ne fut pas chose simple à comprendre. Au-delà de cela, lors de l'augmentation de la capacité du tableau de chaînes de caractères, il fallait allouer un nouveau tableau, chose encore une fois assez compliquée. C'est la partie du développement qui nous a pris le plus de temps. Toutefois, nous sommes parvenus à notre objectif et maîtrisons dorénavant bien les allocations mémoire.

Enfin, l'organisation afin de collaborer en binôme pour travailler sur le développement de l'application fut assez compliqué étant donné que nous étions en distanciel puis en vacances. Toutefois, en consacrant un temps dédié pour le projet, nous sommes parvenus à le réaliser dans son entièreté. Aussi le fait que nous ne nous connaissions pas très bien nous a permis de découvrir l'aspect professionnelle de l'informatique : nous changeons constamment d'équipe de travail.

2- Réussites

Nous avons beaucoup progressé durant le développement du jeu du quart de singe. Premièrement, nous avons réussi à mettre le contenu du fichier .txt dictionnaire directement dans un conteneurTDE appelé Dictionnaire (avec un char** pour le contenu). En effet, la recherche à même le fichier prenait beaucoup de temps et faisait ramer le pc. Nous avions bien conscience du fait que la capacité serait bien plus grande que le nombre réel de mots, mais cela ne posait pas de soucis.

De ce fait, il est devenu plus rapide et plus simple de parcourir l'ensemble du dictionnaire pour comparer la suite de lettres de la manche à chacun des mots du dictionnaire (en vue de trouver une similitude). Dans un second temps, plutôt que d'effectuer une recherche du mot identique de manière linéaire (du début à la fin du tableau de mots), nous avons opté pour le développement d'un algorithme de recherche dichotomique dans un tableau de chaînes de caractères. Ainsi, nous avons facilité et rendu rapide la recherche une première fois avec le fait de mettre le contenu du fichier texte dans un tableau, puis une deuxième fois grâce à l'algorithme de recherche dichotomique.

Enfin, le fait de développer une application complètement permet d'une part de prouver que nous pouvons réaliser des missions en suivant un cahier des charges mais aussi que nous savons optimiser, organiser et simplifier le code toujours

3- Améliorations

Les améliorations que nous aurions pu effectuer concernent principalement le joueur robot qui, à titre de rappel, complète la suite de lettres par une lettre du mot qui commence par la suite de lettres. De ce fait, le robot est sûr d'avoir une réponse à l'interrogation. En effet, s'il ne trouve pas de lettres, alors il renvoie ' ?'. De plus, il cherche le mot grâce à un algorithme de recherche dichotomique, donc toujours le mot le plus proche de la suite de lettres et le moins long. Nous aurions pu améliorer le robot en faisant en sorte qu'il varie les mots à compléter selon s'il finit un mot, s'il peut mettre en difficulté un joueur, s'il est sûr d'être celui qui complète le mot… Or, cela ressemble presque à de l'intelligence artificielle, ce que nous ne maitrisons pas vraiment. Mais pour rendre le jeu plus amusant et avec plus d'enjeux, cela aurait été intéressant. Nous aurions aussi pu prévoir un in-out pour le robot.

V - ANNEXES

```
/**
 * @file Word.h
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 1 2/12/2022
 * Composant Word
 */

#ifndef _WORD_
#define _WORD_

typedef unsigned int uint;// raccourci de "unsigned int"

enum { zerochar = 1 };//taille initiale de la chaîne de caractère contenue dans Word

struct Word {
    char* word; //mot complété tout au long d'une manche de quart de singe
    int lenword; // longueur de ce mot
};

/**@brief alloue un nouveau mot pour la manche dans la partie en cours
*@see detruire_mot_manche, fonction dans laquelle le mot est désalloué
*@param[in,out] NouveauMotManche : le mot pour la manche dans la partie en cours,
alloué
*/
void Nouveau_Mot(Word& NouveauMotManche);

/**@brief ajoute une lettre au Word de la partie
*@param[in] motDePartie : la suite de lettres actuel contenu dans Word
*@param[in] lettre : lettre à ajouter à la suite de lettres
*@param[out]: suite de lettres avec la lettre ajoutée
*/
void Add_Letter(Word& motDePartie, const char lettre);

/**brief compare deux mots, celui contenue dans la structure de données Word et celui
saisi par un joueur lors d'une interrogation
* @param[in] motDeLaManche : représentation du mot à comparer
* @param[in] motSaisi : mot saisi par le joueur
* @return si les mots sont identiques ou non
*/
bool Compare_Mot(const Word& motDeLaManche, const char motSaisi[]);

/**@brief désalloue le mot alloué pour la manche
*@see Nouveau_Mot, fonction dans laquelle le mot est alloué
*@param[in,out] MotDeLaManche : le mot a desalloué
*/
void Detruire_Mot_Manche(Word& MotDeLaManche);
```

```
#endif //! _WORD_
```

```
/**
 * @file Word.cpp
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 1 2/12/2022
 * fonctions associées au Composant Word
 */

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

void Nouveau_Mot(Word& NouveauMotManche) {
    NouveauMotManche.word = new char[zerochar];
    NouveauMotManche.word[0] = '\0';
    NouveauMotManche.lenword = 0;
}

void Add_Letter(Word& motDePartie, const char lettre) {
    uint lenMot = motDePartie.lenword;
    uint newTaille = lenMot + 2;

    char* newword = new char[newTaille];
    strcpy(newword, motDePartie.word);
    newword[lenMot] = toupper(lettre);

    delete[] motDePartie.word;
    motDePartie.word = newword;
    motDePartie.word[lenMot + 1] = '\0';
    motDePartie.lenword = lenMot + 1;
}

bool Compare_Mot(const Word& motDeLaManche, const char motSaisi[]) {
    uint i = 0;
    while (i < motDeLaManche.lenword) {
        if (motSaisi[i] != motDeLaManche.word[i]) {
            return false;
        }
        ++i;
    }
    return true;
}

void Detruire_Mot_Manche(Word& MotDeLaManche) {
    delete[] MotDeLaManche.word;
    MotDeLaManche.word = NULL;
}
```

```
/**
 * @file Joueur.h
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 2 3/12/2022
 * composant Joueur
 */

#ifndef _JOUEUR_
#define _JOUEUR_
typedef unsigned int uint;

struct Joueur {
    char nature; //nature du joueur, robot ou humain
    uint numerojoueur; //numero du joueur
    float quartDeSinge; // nombre de quart de singe affecté au joueur (0.25, 0.5,
0.75, 1)
};



/**@brief ajoute un quart de singe à un joueur
* @param[in,out] Joueur : joueur écopant d'un quart de singe
* @pre singe du joueur < 1
*/
void Ajout_Quart_de_Singe(Joueur& Joueur);

/**@brief verifie si un joueur a obtenu quatre quarts de singe
* @param[in] joueur : joueur pour lequel on veut verifier le nombre de quart de singe
* @return un booléen selon si le joueur a quatre quarts de singe ou non
*/
bool A_Joueur_Singe(const Joueur& joueur);

/**@brief donne l'indice du joueur suivant le joueur courant pendant le déroulement
d'une partie de quart de singe
* @param[in] indice_joueur_courant : indice du joueur courant
* @param[in] NombreJoueurs : le nombre de joueurs jouant au jeu du quart de singe
* @return indice du joueur suivant
*/
uint Joueur_Suivant(uint indice_joueur_courant, const uint NombreJoueurs);

/**@brief donne l'indice du joueur precedent le joueur courant pendant le déroulement
d'une partie de quart de singe
* @param[in] indice_joueur_courant : indice du joueur courant
* @param[in] NombreJoueurs : le nombre de joueurs jouant au jeu du quart de singe
* @return indice du joueur precedent
*/
uint Joueur_Precedent(uint& indice_joueur, const uint NombreJoueurs);

/**@brief quitte le programme si autres que des humains('H') et des robots('R') sont
entrés dans l'invite de commande pour jouer une partie de quart de singe
* @param[in] argv : chaîne de caractères récupérer depuis l'invite de commande
* @param[in] nombreJoueurs : nombre de joueurs entrés par l'utilisateur sur l'invite
de commande
*/
void Joueur_Inconnu(const char* argv, const uint nombreJoueurs);

/**@brief quitte le programme si le jeu accueil moins de 2 joueurs
```

```
* @param[in] nombreJoueurs: nombre de joueurs
*/
void Nombre_Incorrect_Joueur(const uint nombreJoueurs);

#endif // !_JOUEUR_
```

```
/**
 * @file Joueur.cpp
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 2 3/12/2022
 * composant Joueur
 */

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

void Ajout_Quart_de_Singe(Joueur& joueur) {
    assert(joueur.quartDeSinge < 1);
    joueur.quartDeSinge += 0.25;
}

bool A_Joueur_Singe(const Joueur& joueur) {
    return (joueur.quartDeSinge == 1) ? true : false;
}

uint Joueur_Suivant(uint indice_joueur_courant, const uint NombreJoueurs) {
    if (indice_joueur_courant == NombreJoueurs - 1) {
        return 0;
    }
    else {
        return indice_joueur_courant += 1;
    }
}

uint Joueur_Precedent(uint& indice_joueur, const uint NombreJoueurs) {
    if (indice_joueur == 0) {
        return NombreJoueurs-1;
    }
    else {
        return indice_joueur - 1;
    }
}

void Joueur_Inconnu(const char* argv, const uint nombreJoueurs) {
    for (int i = 0; i < nombreJoueurs; ++i) {
        if (argv[i] != 'H' && argv[i] != 'R') {
            cout << "seuls les humains et les robots peuvent jouer";
            exit(2);
        }
    }
}

void Nombre_Incorrect_Joueur(const uint nombreJoueurs) {
```

```
    if (nombreJoueurs < 2) {
        cout << "La partie se joue doit se jouer a plusieurs";
        exit(2);
    }
}
```

```
/**
 * @file Partie.h
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 1 31/11/2022
 * Composant de la partie
 */

#ifndef _PARTIE_
#define _PARTIE_

#include "Joueur.h"
#include "Word.h"



struct Partie {
    uint nbjoueurs; //nombre de joueurs participant à la partie de quart de Singe
    Joueur* joueurs; //tableau de joueurs participants à la partie de quart de Singe
(voir structure de données Joueur dans Joueur.h)
    Word wordInGame; //mot à compléter pendant la partie de quart de singe
};

/**
 * @brief CreateGame crée une partie de quart de singe
 * Allocation en mémoire dynamique de la partie
 * de capacité (NombreJoueurs)
 * @see EndGame, pour sa désallocation en fin d'utilisation
 * @param[out] game : la partie de quart de singe
 * @param [in] NombreJoueurs : le nombre de joueurs
 * @pre NombreJoueurs>1
 */
void Create_Game(Partie& game, const int NombreJoueurs);

/**
 * @brief Ajouter des joueurs dans une Partie
 * @param[out] game : la partie de quart de singe
 * @param[in] NombreJoueurs : le nombre de joueurs dans la partie
 * @param[in] argv : une chaine de caractère contenant la liste des joueurs et leur
nature respective
 * @pre NombreJoueurs > 1
 */
void Add_Players(Partie& game, const int NombreJoueurs, const char* argv);

/**
 * @brief Désalloue une partie de quart de singe
 * @see CreateGame, la partie à déjà été alloué
 * @param[out] game : la partie de quart de singe
 */
void End_Game(Partie& game);
```

```
#endif // !_PARTIE_
```

```
/**
 * @file Partie.cpp
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 1 31/11/2022
 * Composant de la partie
 */

#include <iostream>
#include <iomanip> // pour setw
#include <fstream>
#include <cctype>
#include <cassert>
#include <cassert>
#include "Partie.h"
using namespace std;

void Create_Game(Partie& game, const int NombreJoueurs) {
    assert(NombreJoueurs > 1);
    game.joueurs = new Joueur[NombreJoueurs];
    game.nbjoueurs = NombreJoueurs;
}

void Add_Players(Partie& game, const int NombreJoueurs, const char* argv) {
    assert(NombreJoueurs > 1);
    for (int i = 0; i < NombreJoueurs; ++i) {
        game.joueurs[i].nature = toupper(argv[i]);
        game.joueurs[i].numerojoueur = i + 1;
        game.joueurs[i].quartDeSinge = 0;
    }
}

void End_Game(Partie& game) {
    delete[] game.joueurs;
    game.joueurs = NULL;
}
```

```
/**
 * @file Dico.h
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 1 24/12/2022
 * Composant du dictionnaire
 */

#ifndef _DICO_
#define _DICO_
#include <cstring>
#pragma warning(disable: 4996)

typedef unsigned int uint;
```

```
struct Dictionnaire {
    char** contenu; //un tableau de mots appelé dictionnaire contenant l'ensemble des
mots d'un dictionnaire
    uint capacite; // capacité du dictionnaire (souvent différente du nombre de mots
réel)
    double pasExtension; //pas d'extension de la capacité du dictionnaire
    uint nombreMots; // nombre réel de mots dans le dictionnaire
};


/**@brief initialise un tableau de chaine de caractères qui contiendra tous les mots
du dictionnaire
*@see Detruire_Dico pour la desallocation du Dictionnaire
*@param[in,out] dico : le Dictionnaire contenant tous les mots d'un fichier
*@param[in] capa : la capacité initiale du Dictionnaire
*@param[in] p : le pas d'extension choisi lorsqu'il faudra augmenter la taille du
Dictionnaire
* @pre capa > 0 , p > 0
*/
void Initialiser_Dico(Dictionnaire& dico, const uint capa, const double p);

/**@brief inclus (en lisant) tous les mots d'un fichier texte représentant le
dictionnaire dans le Dictionnaire dynamique
*@param[in,out] dico : le Dictionnaire qui va contenir tous les mots du dictionnaire
*/
void Ecrire_Mots_Dico(Dictionnaire& conteneurTdeDico);

/**@brief recherche dichotomique d'une chaîne de caractères dans un tableau de chaine
de caractères
*@param[in] dico : le tableau de chaines de caractères dans lequel on cherche le mot
*@param[in] nombreMots : nombre de chaînes de caractères dans le tableau
*@param[in] word : mot pour lequel on cherche la présence ou non dans le tableau de
chaînes de caractères
*@return la présence ou non du mot dans le tableau de chaînes de caractères sous la
forme d'un booléen
*/
bool Recherche_Mot_Dico_Dichoto(char** dico, const uint nombreMots, const char* word);

/**@brief recherche dichotomique, par le robot, dans un tableau de chaine de
caractères, d'un mot commençant par les mêmes lettres que le chaine de caractère
complétée durant la partie pour le soumettre en réponse
*@param[in] dico : le tableau de chaines de caractères dans lequel on cherche le mot
*@param[in] dicoNombreMots : nombre de chaînes de caractères dans le tableau de chaîne
de caracètre
*@param[in] motDeLaManche : mot complétée durant la partie, pour lequel on cherche un
mot identique
*@param[in] lenword : longueur du mot complété durant la manche
*@return le mot trouvé, ayant les premières lettres identiques à celui complétée
durant la partie
*/
char* Recherche_Robot_Mot_Dichoto(char** dico, const uint dicoNombreMots, const char*
motDeLaManche, const uint lenword);

/**@brief recherche dichotomique, par le robot, dans un tableau de chaine de
caractères, d'un mot commençant par les mêmes lettres que le chaine de caractère
complétée durant la partie pour jouer la lettre suivante
*@param[in] dico : le tableau de chaines de caractères dans lequel on cherche le mot
```

```
*@param[in] dicoNombreMots : nombre de chaînes de caractères dans le tableau de chaîne
de caracètre
*@param[in] motDeLaManche : mot complété durant la partie, pour lequel on cherche la
lettre suivante
* @param[in] lenword : longueur du mot complété durant la manche
*@return une lettre, pour un mot ayant les premières lettres identiques à celui
complétée durant la partie, ou d'un ? si aucune lettre n'est cohérente
*/
char Recherche_Début_Mot_Dichoto(char** dico, const uint dicoNombreMots, const char*
motDeLaManche, const uint lenword);

/**@brief desalloue le dictionnaire à la fin de la partie
*@see Initialiser_Dico pour l'allocation du Dictionnaire
*@param[in,out] dico : le Dictionnaire utilisé pour stocké tous les mots du
dictionnaire
*/
void Detruire_Dico(Dictionnaire& dico);


#endif // !_DICO_
```

```
/**
 * @file Dico.cpp
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 1 24/12/2022
 * Composant du dictionnaire
 */

#include <iostream>
#include <iomanip> // pour setw
#include <fstream>
#include <cctype>
#include <cassert>
#include "Dico.h"
using namespace std;



void Initialiser_Dico(Dictionnaire& dico, const uint capa, const double p) {
    assert((capa > 0) && (p>0));
    dico.capacite = capa;
    dico.pasExtension = p;
    dico.contenu = new char*[capa];
}

void Ecrire_Mots_Dico(Dictionnaire& conteneurTdeDico) {
    ifstream in("ods4.txt");
    if (!in) {
        cout << "le dictionnaire n'a pu etre ouvert" << endl;
        exit(69);
    }
    const int MAX = 26;
    char motFichier[MAX];
    conteneurTdeDico.nombreMots = 0;
    in >> setw(MAX) >> motFichier; // on essaye de lire le premier mot
```

```
    while (in) {
        char* mot = new char[strlen(motFichier)+1];
        strcpy(mot, motFichier);
        if (conteneurTdeDico.nombreMots >= conteneurTdeDico.capacite) {
            uint newTaille = (conteneurTdeDico.capacite *
conteneurTdeDico.pasExtension);
            char** newDico = new char*[newTaille];
            for (uint j = 0; j < conteneurTdeDico.capacite; ++j) {
                newDico[j] = conteneurTdeDico.contenu[j];
            }
            delete[] conteneurTdeDico.contenu;
            conteneurTdeDico.contenu = newDico;
            conteneurTdeDico.capacite = newTaille;
        }
        conteneurTdeDico.contenu[conteneurTdeDico.nombreMots] = mot;
        conteneurTdeDico.nombreMots ++;
        in >> setw(MAX) >> motFichier; // on essaye de lire le mot suivant

    }
    in.close(); // on ferme le fichier
}

bool Recherche_Mot_Dico_Dichoto(char** dico, const uint nombreMots, const char* word)
{
    bool estMotDico = false;
    int debutDico = 0, finDico = nombreMots - 1;

    while (debutDico <= finDico) {
        int milieuDico = (debutDico + finDico) / 2;
        if (strcmp(word, dico[milieuDico]) == 0) {
            estMotDico = true;
            break;
        }
        else {
            if (strcmp(word, dico[milieuDico]) < 0)
                finDico = milieuDico - 1;
            else
                debutDico = milieuDico + 1;
        }

    }
    return estMotDico;
}

char* Recherche_Robot_Mot_Dichoto(char** dico, const uint dicoNombreMots, const char*
motDeLaManche,const uint lenword) {
    int debutDico = 0, finDico = dicoNombreMots - 1;
    while (debutDico <= finDico) {
        int milieuDico = (debutDico + finDico) / 2;

        if (strncmp(motDeLaManche, dico[milieuDico], lenword) == 0) {
            cout << dico[milieuDico] << endl;
            return dico[milieuDico];

        }
        else {
```

```
            if (strncmp(motDeLaManche, dico[milieuDico], strlen(dico[milieuDico])) <
0)
                finDico = milieuDico - 1;

            else
                debutDico = milieuDico + 1;
        }

    }
}

char Recherche_Début_Mot_Dichoto(char** dico, const uint dicoNombreMots, const char*
motDeLaManche, const uint lenword) {

    int debutDico = 0, finDico = dicoNombreMots - 1;

    while (debutDico <= finDico) {
        int milieuDico = (debutDico + finDico) / 2;

        if (strncmp(motDeLaManche, dico[milieuDico], lenword) == 0) {
            return dico[milieuDico][lenword];
        }
        else {
            if (strncmp(motDeLaManche, dico[milieuDico], strlen(dico[milieuDico])) <
0)
                finDico = milieuDico - 1;

            else
                debutDico = milieuDico + 1;
        }

    }
    return '?';
}

void Detruire_Dico(Dictionnaire& dico) {
    delete[] dico.contenu;
    dico.contenu = NULL;
}
```

```
/**
 * @file JeuEnCours.h
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 2 28/12/2022
 * Fonctions essentiels au bon déroulement du jeu du quart de singe
 */

#ifndef _JeuEnCours_
#define _JeuEnCours_
       #include "Partie.h"
       #include "Dico.h"

enum {
       MAX = 26,
```

```
};

/**@brief annonce la fin de la manche et désalloue le mot constitué lors de cette
manche
* @see Affiche_Check_Point pour l'affichage de la fin de la manche
* @param[in,out] game : la partie de quart de singe
* @param[in,out] motdeLaManche : mot désalloué à la fin de la manche
*/
void Fin_Manche(Partie& game,Word& motdeLaManche);

/**@brief annonce l'abandon d'un joueur, lui ajoute un quart de singe et arrête la
manche
* @param[in,out] game : la partie de quart de singe qui se déroule
* @param[in,out] motDeLaManche : le mot de la manche qui va être désalloué
* @param[in,out] joueur : le joueur ayant abandonné et recevant un quart de singe
suite à son abandon
*/
void Abandon_Joueur(Partie& game, Word& motDeLaManche, Joueur& joueur);

/**@brief determine le joueur qui doit jouer sous forme d'indice selon si c'est la fin
d'une manche ou non
* @see Joueur_Suivant pour comprendre qui est le joueur suivant si ce n'est pas la fin
d'une manche
* @param[in] finManche : booleen determinant si c'est la fin d'une manche ou non
* @param[in] joueur : joueur courant
* @param[in] nombreJoueurs : Nombre de joueur jouant au jeu du quart de singe pour
cette partie
* @return le joueur suivant
*/
uint Determine_Joueur_Suivant(const bool finManche, const uint joueur, const uint
nombreJoueurs);

/**@brief Annonce que le joueur interrogé n'a pas saisi un mot commençant par les
mêmes lettres que la chaine de caractères complétée pendant la manche de quart de
singe et lui ajoute un quart de singe
* @param[in] motEnCoursDeSaisie : mot saisie par le joueur interrogé, ne commençant
pas par les bonnes lettres
* @param[in] joueurInterroge : le joueur interrogé
*/
void Mauvais_Debut_Mot(const char motEnCoursDeSaisie[], Joueur& joueurInterroge);

/**@brief met tous les caractères de la chaîne de caractères en majuscule
* @param[in,out] motEnCoursDeSaisie : chaîne de caractères à mettre en majuscule
*/
void Majuscule(char* motEnCoursDeSaisie);

/**brief Confrontation entre le joueur interrogé et le joueur concurrent. Le joueur
interrogé saisi un mot, s'il n'existe pas ou s'il ne commence pas par les bonnes
lettres, il perd la manche et prend un quart de singe, sinon c'est le joueur
concurrent qui perd.
* @param[in] game : la partie de quart de singe
* @param[in,out] joueurInterroge: Joueur invité à saisir un mot
* @param[in,out] joueurConcurrent: Joueur ayant invité le joueur précedent à saisir un
mot
* @param[in,out] motDeLaManche: mot de la manche détruit à l'issu de cet confrontation
* @param[in] dico: Dictionnaire dans lequel le mot saisi figure ou non
* @param[in,out] plDebManche: joueur qui va entamé la manche à l'issu de cet
confrontation entre les deux joueurs.
```

```
*/
void Saisie_Mot_Humain(Partie& game, Joueur& joueurInterroge, Joueur&
joueurConcurrent, Word& motDeLaManche, Dictionnaire& dico, uint& plDebManche);

/**@brief renvoie une lettre aléatoire dans l'alphabet
* @return une lettre au hasard dans l'alphabet
*/
char Lettre_Random_Robot();


/**@brief affiche l'identifiant d'un joueur et le mot actuel afin d'annoncer son tour.
*@param[in] joueur : joueur courant
*@param[in] motDelaPartie : la suite de lettres actuel
*/
void Affiche_Tour(const Joueur& joueur, Word& motDelaPartie);

/**@brief annonce l'abandon d'une manche par un joueur
*@param[in] joueur : joueur ayant abandonné la manche
*/
void Affiche_Abandon(const Joueur& joueur);

/**@brief annonce que le mot existe ainsi que le joueur prend un quart de singe
*@param[in] MotDeLaManche : mot existant à afficher
*@param[in] joueur : le joueur ayant entré un mot existant et perdu la manche
*/
void Affiche_Mot_Existe(const Word& MotDeLaManche, const Joueur& joueur);

/**@brief affiche pour chaque joueur le nombre de quart de singe à la fin de chaque
manche
*@param[in] game : la partie se déroulant
*/
void Affiche_Check_Point(const Partie& game);


/**@brief affiche une demande au joueur interrogé afin qu'il saisisse un mot
*@param[in] joueurInterroge: Le joueur interroge
*/
void Affiche_Saisir_Mot(const Joueur& joueurInterroge);

/**@brief affiche que le mot saise par le joueur interrogé ne commence pas par les
bonnes lettres
*@param[in] motEnCoursDeSaisie : le mot saisie par le joueur interrogé
*@param[in] joueurInterroge : le joueur interroge
*/
void Affiche_Mauvais_Debut_Mot(const char motEnCoursDeSaisie[], Joueur&
joueurInterroge);

/**@brief affiche que le mot saisi existe et que, par consequent, le joueur ayant
interrogé le joueur precedent perd la manche
*@param[in] motSaisie : le mot saisi lors de l'interrogation
*@param[in] joueurConcurrent : le joueur ayant interroge le joueur precedent, perdant
de la manche
*/
void Affiche_Joueur_Concurrent_Manche_Perdu(const char motSaisie[], Joueur&
joueurConcurrent);

/**@brief affiche que le mot saisi n'existe pas et que, par consequent, le joueur
interrogé perd la manche
```

```
*@param[in] motSaisie : le mot saisi lors de l'interrogation
*@param[in] joueurConcurrent : le joueur ayant interrogé, perdant de la manche
*/
void Affiche_Joueur_Interroge_Manche_Perdu(const char motSaisie[], Joueur&
joueurInterroge);

/**@brief annonce que la réponse du robot existe et qu'il prend un quart de singe
*@param[in] reponse : mot existant à afficher
*@param[in] joueur : le robot ayant entré un mot existant et perdu la manche
*/

void Affiche_Mot_Existe(char* reponse, const Joueur& joueur);

/**@brief affiche une lettre
*@param[in] lettre : lettre à afficher
*/
void Afficher_Lettre(const char lettre);

#endif // !_JeuEnCours_
```

```
/**
 * @file JeuEnCours.cpp
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version 2 28/12/2022
 * Fonctions essentiels au bon déroulement du jeu du quart de singe
 */

#include <iostream>
#include <cassert>
#include <iomanip>
#include "JeuEnCours.h"
using namespace std;

void Fin_Manche(Partie& game,Word& motdeLaManche) {
	Affiche_Check_Point(game);
	Detruire_Mot_Manche(motdeLaManche);
}


void Abandon_Joueur(Partie& game,Word& motDeLaManche, Joueur& joueur) {
	Affiche_Abandon(joueur);
	Ajout_Quart_de_Singe(joueur);
	Fin_Manche(game,motDeLaManche);
}


uint Determine_Joueur_Suivant(const bool finManche, const uint joueur, const uint
nombreJoueurs) {
	if (finManche == true) {
		return joueur;
	}
	else {
		return Joueur_Suivant(joueur, nombreJoueurs);
```

```
        }
}

void Mauvais_Debut_Mot(const char motEnCoursDeSaisie[], Joueur& joueurInterroge) {
    Affiche_Mauvais_Debut_Mot(motEnCoursDeSaisie, joueurInterroge);
    Ajout_Quart_de_Singe(joueurInterroge);
}

void Majuscule(char* motEnCoursDeSaisie) {
    for (int i = 0; i < strlen(motEnCoursDeSaisie); ++i) {
        motEnCoursDeSaisie[i] = toupper(motEnCoursDeSaisie[i]);
    }
}



void Saisie_Mot_Humain(Partie& game, Joueur& joueurInterroge, Joueur&
joueurConcurrent, Word& motDeLaManche, Dictionnaire& dico, uint& plDebManche) {
    char motEnCoursDeSaisie[MAX];
    cin >> setw(MAX) >> motEnCoursDeSaisie;
    cin.ignore(INT_MAX, '\n');

    Majuscule(motEnCoursDeSaisie);
    if (motDeLaManche.lenword != 0) {
        if (Compare_Mot(motDeLaManche, motEnCoursDeSaisie) == false) {
            Mauvais_Debut_Mot(motEnCoursDeSaisie, joueurInterroge);
            Fin_Manche(game, motDeLaManche);
            plDebManche = joueurInterroge.numerojoueur - 1; // ce joueur entamera la
manche suivante
            return;
        }
    }


    if (Recherche_Mot_Dico_Dichoto(dico.contenu, dico.nombreMots, motEnCoursDeSaisie))
{
        Affiche_Joueur_Concurrent_Manche_Perdu(motEnCoursDeSaisie, joueurConcurrent);
        Ajout_Quart_de_Singe(joueurConcurrent);
        Fin_Manche(game, motDeLaManche);
        plDebManche = joueurConcurrent.numerojoueur - 1; // ce joueur entamera la
manche suivante
        return;
    }

    Affiche_Joueur_Interroge_Manche_Perdu(motEnCoursDeSaisie, joueurInterroge);
    Ajout_Quart_de_Singe(joueurInterroge);
    Fin_Manche(game, motDeLaManche);
    plDebManche = joueurInterroge.numerojoueur - 1; // ce joueur entamera la manche
suivante
    return;
}

char Lettre_Random_Robot() {
    char lettre;
    const char alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    size_t indice;

    indice = rand() % (25 + 1);
```

```
    lettre = alphabet[indice];
    return lettre;
}

void Affiche_Tour(const Joueur& joueur, Word& motDelaPartie) {
    cout << joueur.numerojoueur << joueur.nature << "," << " ";
    cout << "(" << motDelaPartie.word << ") > ";
}

void Affiche_Abandon(const Joueur& joueur) {
    cout << "le joueur " << joueur.numerojoueur << joueur.nature << " abandonne la
manche et prend un quart de singe" << endl;

}

void Affiche_Mot_Existe(const Word& MotDeLaManche, const Joueur& joueur) {
    cout << "le mot " << MotDeLaManche.word << " existe, le joueur " <<
joueur.numerojoueur << joueur.nature << " prend un quart de singe" << endl;
}

void Affiche_Check_Point(const Partie& game) {
    for (uint i = 0; i < game.nbjoueurs; ++i) {
        if (i == game.nbjoueurs - 1)
            cout << game.joueurs[i].numerojoueur << game.joueurs[i].nature << " : " <<
game.joueurs[i].quartDeSinge;
        else
            cout << game.joueurs[i].numerojoueur << game.joueurs[i].nature << " : " <<
game.joueurs[i].quartDeSinge << "; ";
    }
    cout << endl;
}

void Affiche_Saisir_Mot(const Joueur& joueurInterroge) {
    cout << joueurInterroge.numerojoueur << joueurInterroge.nature << ", saisir le mot
> ";
}

void Affiche_Mauvais_Debut_Mot(const char motEnCoursDeSaisie[], Joueur&
joueurInterroge) {
    cout << "le mot " << motEnCoursDeSaisie << " ne commence pas par les lettres
attendues, ";
    cout << "le joueur " << joueurInterroge.numerojoueur << joueurInterroge.nature <<
" prend un quart de singe" << endl;
}

void Affiche_Joueur_Concurrent_Manche_Perdu(const char motSaisie[], Joueur&
joueurConcurrent) {
    cout << "le mot " << motSaisie << " existe, le joueur " <<
joueurConcurrent.numerojoueur << joueurConcurrent.nature << " prend un quart de singe"
<< endl;
}

void Affiche_Joueur_Interroge_Manche_Perdu(const char motSaisie[], Joueur&
joueurInterroge) {
    cout << "le mot " << motSaisie << " n'existe pas, le joueur " <<
joueurInterroge.numerojoueur << joueurInterroge.nature << " prend un quart de singe"
<< endl;
}
```

```
void Affiche_Mot_Existe(char* reponse, const Joueur& joueur) {
    cout << "le mot " << reponse << " existe, le joueur " << joueur.numerojoueur <<
joueur.nature << " prend un quart de singe" << endl;

}

void Afficher_Lettre(const char lettre) {
    cout << lettre << endl;
}
```

```
/**
 * @file singe.cpp
 * @brief SAE 1.02
 * @author BSIKRI Mouhamed Nadir / MAATOUGUI Nassim
 * @version finale 3/1/2023
 * Deroulement d'une partie de quart de singe
 */

#include <iostream>
#include <iomanip> // pour setw
#include "JeuEnCours.h"
using namespace std;



int main(int argc, const char* argv[]) {
    srand(time(NULL));
    Partie game;
    uint NombreJoueurs = strlen(argv[1]);

    Nombre_Incorrect_Joueur(NombreJoueurs);
    Joueur_Inconnu(argv[1], NombreJoueurs);

    Create_Game(game, NombreJoueurs);
    Add_Players(game, NombreJoueurs, argv[1]);
    Nouveau_Mot(game.wordInGame);
    Word& motDePartie = game.wordInGame;


    Dictionnaire dico;
    Initialiser_Dico(dico, 350000, 1.25);
    Ecrire_Mots_Dico(dico);


    bool finmanche = false;
    uint i = 0;

    while (1) {
        char lettre;
        finmanche = false;
        Affiche_Tour(game.joueurs[i], motDePartie);
        if (game.joueurs[i].nature == 'H') {
            cin >> lettre;
            cin.ignore(INT_MAX, '\n');
```

```
        }
        else if (game.joueurs[i].nature == 'R') {
            if (motDePartie.lenword == 0) {

                lettre = Lettre_Random_Robot();
            }
            else {
                lettre = Recherche_Début_Mot_Dichoto(dico.contenu, dico.nombreMots,
motDePartie.word,motDePartie.lenword);
            }
            Afficher_Lettre(lettre);
        }

        if (lettre == '!') {

            Abandon_Joueur(game, motDePartie, game.joueurs[i]);
            if (A_Joueur_Singe(game.joueurs[i])) {
                break;
            }
            Nouveau_Mot(motDePartie);
        }
        else if (lettre == '?') {
            i = Joueur_Precedent(i, NombreJoueurs);
            Affiche_Saisir_Mot(game.joueurs[i]);
            if (game.joueurs[i].nature == 'H') {
                Saisie_Mot_Humain(game, game.joueurs[i],
game.joueurs[Joueur_Suivant(i, NombreJoueurs)], motDePartie, dico, i);
            }
            else if (game.joueurs[i].nature == 'R') {
                char* reponse;
                reponse = Recherche_Robot_Mot_Dichoto(dico.contenu, dico.nombreMots,
motDePartie.word,motDePartie.lenword);
                Affiche_Mot_Existe(reponse,
game.joueurs[Joueur_Suivant(i,NombreJoueurs)]);
                Ajout_Quart_de_Singe(game.joueurs[Joueur_Suivant(i, NombreJoueurs)]);
                Fin_Manche(game, motDePartie);
                i = Joueur_Suivant(i, NombreJoueurs);
            }
            if (A_Joueur_Singe(game.joueurs[i]) ||
A_Joueur_Singe(game.joueurs[Joueur_Suivant(i, NombreJoueurs)])) {
                break;
            }
            Nouveau_Mot(motDePartie);
        }
        else {
            Add_Letter(motDePartie, lettre);
            if (motDePartie.lenword > 2) {
                if (Recherche_Mot_Dico_Dichoto(dico.contenu, dico.nombreMots,
motDePartie.word)) {
                    Affiche_Mot_Existe(motDePartie, game.joueurs[i]);
                    Ajout_Quart_de_Singe(game.joueurs[i]);
                    Fin_Manche(game, motDePartie);
                    if (A_Joueur_Singe(game.joueurs[i])) {
                        break;
                    }
                    Nouveau_Mot(motDePartie);
                    finmanche = true;
                }
```

```
            }
            i = Determine_Joueur_Suivant(finmanche, i, NombreJoueurs);
        }
    }

    cout << "La partie est finie";
    Detruire_Dico(dico);
    End_Game(game);
}
```