BUT- SAE S.1 02

Comparaison d’approches algorithmiques
-LE QUART DE SINGE-

Réalisé par Zakary Melouk (108) et Leïna Ben Bella (106)

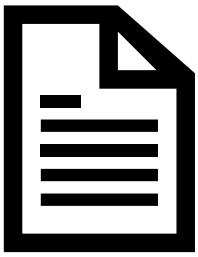


Table des matières

Présentation du projet

Résumé du sujet

Le projet à réaliser avait pour but de créer un jeu nommé « Le quart de singe ».
Ce jeu est simple, il se joue à au moins 2 joueurs et consiste à former un mot du dictionnaire français en énonçant les lettres chacun son tour, et sans savoir à quel mot pense son ou ses adversaires.
A chaque manche qu'il perd le joueur prend un quart de singe, arrivé à quatre quarts de singe le joueur perd la partie.
Notre travail a été de réaliser le jeu de manière à pouvoir avoir des parties : humain contre humain, humain contre robot, et robot contre robot.

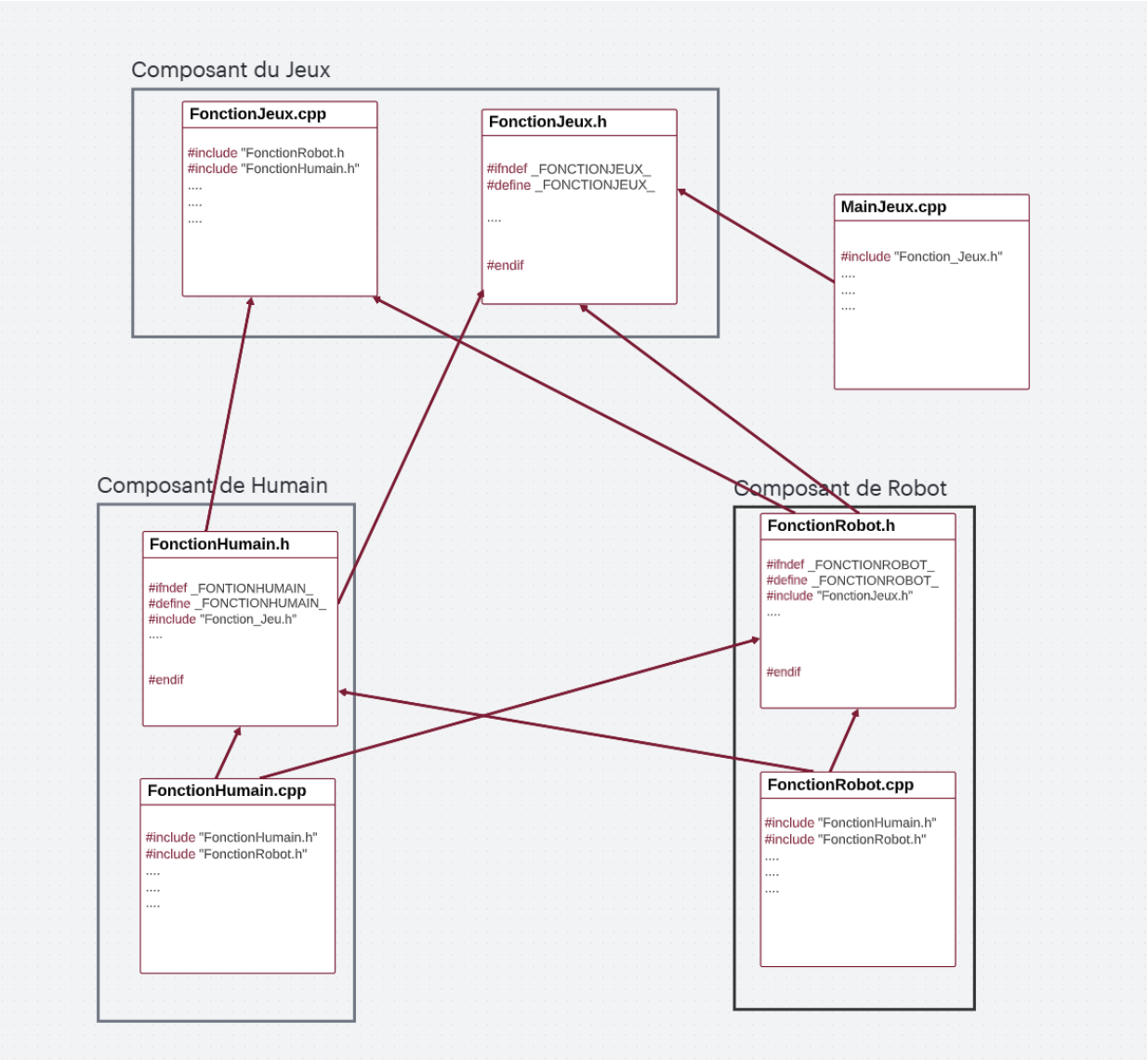
Spécificités du code

La partie humaine de ce jeu est assez classique, cependant pour la partie robot nous avons apporté plusieurs spécifications afin de le rendre le plus performant possible.
Quand le robot est le premier à jouer, il donne une lettre au hasard, ensuite à chacun de ses tours il vérifie que le mot en cours est un bien un mot du dictionnaire inclus. Si ce n'est pas le cas le robot envoie un « ? » pour piéger le joueur précédent, sinon il enregistre toutes les lettres qui peuvent finir le mot en cours et ainsi le faire perdre.
Avec la liste de lettre enregistrées le robot va chercher une lettre ne faisant pas partie de cette liste et étant avant dernière d'un mot pour piéger le jouer suivant. Si le robot ne parvient à trouver aucune lettre, il donnera la prochaine lettre (qui ne doit pas faire partie des lettres enregistrées précédemment) du mot le plus long du dictionnaire commençant pas les lettres du mot en cours.
Dans le cas où le robot n'a aucune possibilité de jouer une lettre qui ne le fait pas perdre, notre robot a plusieurs actions possibles :

- Si les deux joueurs avant et après sont des robots, il abandonne en envoyant un « ! ».
- Si le joueur précédent est un robot et le suivant un humain, le robot va « bluffer » en envoyant une lettre au hasard.
- Si le joueur précédant est un humain et le suivant un robot, le robot va envoyer un « ? » car la seule possibilité pour lui de s'en sortir c'est que l'humain ne connaisse pas de mot qui commence par les bonnes lettres ou encore qu'il se trompe dans l'orthographe.
- Et enfin si les deux joueurs avant et après lui sont des humains, le robot a une chance sur deux de décider de peut-être piéger le joueur précédent en envoyant un « ? » ou alors le joueur suivant en envoyant une lettre au hasard.

Graphe de dépendance des fichiers

Voici notre graphe de dépendance des fichiers :



Code source des tests unitaires

Voici la série de test pour le fichier FonctionJeux.cpp :

```
/**
 * @file testFonctionJeux.cpp
 * Test du composant FonctionJeux.cpp
 */

#include <iostream>
#include <cassert>
#include "FonctionJeux.H"


using namespace std;


int main() {
    Jeux j;
    char c = 'a';
    assert(transforme(c) == 'A');
    char joueur2[2] = "";
    joueur2[0] = 'H';
    assert(verif(joueur2) == 1);
    char joueur3[4] = "GGG";
    assert(verif(joueur3) == 1);
    char joueur[4] = "HRH";
    assert(verif(joueur) == 0);
    intialiser_struct(j, joueur);
    assert(j.nb_joueurs == 3);
    assert(j.nb_robots == 1);
    assert(j.nb_humains == 2);
    ajout_lettre(j, 0, 'A', joueur);
    assert(j.mot_en_cours[0] == 'A');
    assert(j.nbtour == 1);
    assert(j.indice_mot == 1);
    assert(j.joueur_precedent == 0);

}
```

Voici celle pour le fichier FonctionHumain.cpp :

```
/**
 * @file testFonctionHumain.cpp
 * Test du composant FonctionHumain.cpp
 */

#include <iostream>
#include <cassert>
```

```
#include "FonctionHumain.H"

using namespace std;


int main() {
    Jeux j;
    char test[MAX_CHAR] = "Bonjour";
    unsigned int taille = strlen(test);
    assert(verif_existence_mot(test) == false);
    for (unsigned int i = 0; i < taille; ++i) {
        test[i] = transforme(test[i]);
    }
    assert(verif_existence_mot(test) == true);
    assert(verif_existence_mot("hello") == false);

    char joueur[] = "HHH";

    intialiser_struct(j, joueur);

    ajout_lettre(j, 0, 'B', joueur);
    ajout_lettre(j, 0, 'A', joueur);
    char test1[7] = "AZERTY";
    char test2[8] = "BAZERTY";
    assert(verifier_validite_mot(test1, j) == false);
    assert(verifier_validite_mot(test2, j) == true);
}
```

Et enfin, la série de test pour le fichier FonctionRobot.cpp :

```
/**
 * @file testFonctionRobot.cpp
 * Test du composant FonctionRobot.cpp
 */

#include <iostream>
#include <cassert>
#include "FonctionRobot.h"


using namespace std;



int main() {
    Jeux j;

    char joueur[] = "HHH";
    intialiser_struct(j, joueur);
    ajout_lettre(j, 0, 'B', joueur);
```

```
    lettre_perdante(j);
    assert(trouver_lettre_piege(j) == 'A');
    ajout_lettre(j, 0, 'A', joueur);
    char mot[MAX_CHAR];
    assert(trouver_mot(j, mot) == 0);
    assert(verif_existence_mot_en_cours(j) == true);
    ajout_lettre(j, 0, 'X', joueur);
    ajout_lettre(j, 0, 'Z', joueur);
    ajout_lettre(j, 0, 'W', joueur);
    assert(trouver_mot(j, mot) == 1);
    assert(verif_existence_mot_en_cours(j) == false);
}
```

Nous avons réussi à passer tous les tests que nous avons élaborer pour nos fonctions. Nous avons également testé les fichier in/out envoyé par le professeur et notre programme les a également réussit

Bilan du projet

Pour conclure ce nouveau projet a été une expérience plus particulière et différente que le précédente, du fait de sa plus grande liberté.
Sur la première SAE nous étions plus guidés, sur l'ordre des commandes à réaliser, ce qui est bien plus flou dans cette nouvelle. C'est pourquoi nous avons pu rencontrer dès lors du démarrage de la réalisation du projet ou encore de la répartition des tâches.
Bien heureusement ses problèmes n'ont pas persisté.
Mais cela nous a permis de plus se sentir dans un cadre professionnel, dans lequel les attentes ne sont pas toujours explicites.

Pour ce qui est des problèmes « algorithmiques », nous avons eu quelques soucis pour bien stocker les quarts de singes aux bons indices ou encore les lettres formant le mot en cours avec les méthodes dynamiques, des difficultés de manipulations de ficher en mode lecture avec setw.
Heureusement, notre collaboration à deux nous a permis de nous entraider et de nous tirer vers le haut, à chaque problème rencontré, nous étions deux cerveaux à réfléchir sur la question et la résolution de ces problèmes allait donc beaucoup plus vite.

En outre, la réalisation de ce projet s'est passée sans accros et a été une réussite.

Annexes

```
#ifndef _FONCTIONJEUX_
#define _FONCTIONJEUX_

#include <iostream>
using namespace std;

enum { MAX_CHAR = 25 };
enum { NB_MOT = 369085 };

struct Joueurs {
    unsigned int indice_joueur_humain;
    float nb_quart_singe;
};

struct Robots {
    unsigned int indice_joueur_robots;
    float nb_quart_de_singe;
};


struct Jeux {
    Joueurs* joueurs;
    Robots* robots;
    unsigned int nb_joueurs;
    char* mot_en_cours;
    char* lettre_perdante;
    unsigned int indice_mot;
    unsigned int agrandissement_mot;
    unsigned int joueur_precedent;
    unsigned int joueur_precedant_indice;
    unsigned int nbtour;
    unsigned int joueur_perdant;
    unsigned int passage_h;
    unsigned int passage_r;
    unsigned int nb_robots;
    unsigned int nb_humains;
    unsigned int indice_lettre;
    bool partie_fini;
    bool manche_fini;
};
```

```
/**
 * @brief transforme un caractere en majuscule si c'est une majuscule
 * @param[in] c : un caractere
 * @return le caractere majuscule
 */
char transforme(char c);

/**
 * @brief initialise la partie
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 */
void intialiser_struct(Jeux& j, const char joueurs[]);

/**
 * @brief verifie que le nombre et la nature des joueurs sont bons
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 * @return 1 si les conditions ne sont pas respectées, sinon 0
 */
int  verif(char joueurs[]);

/**
 * @brief Lance une partie
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 */
void partie(Jeux& j, const char joueurs[]);

/**
 * @brief ajoute une lettre au tableau contenant le mot en cours
 * @param[in,out] j : structure de la partie
 * @param[in] i: indice du joueur actuel
 * @param[in] lettre: la lettre à ajouter
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 */
void ajout_lettre(Jeux& j, const unsigned int i, const char lettre, const char
joueurs[]);

/**
 * @brief affiche le nombre de quart de singe de chaque joueur et si la partie
est finie
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
```

```
 * @return true si la partie est finie, sinon false
 */
bool affichage(Jeux& j, const char joueurs[]);

/**
 * @brief Désallocation d'une partie
 * @param[in,out] j : structure de la partie
 */
void desallouer(Jeux& j);


#endif
```

```
#include <iostream>
#include <fstream>
#include <iomanip> // pour setw
#pragma warning(disable:4996)
using namespace std;

#include "FonctionRobot.h"
#include "FonctionHumain.h"

/**
 * @brief transforme un caractere en majuscule si c'est une majuscule
 * @param[in] c : un caractere
 * @return le caractere majuscule
 */
char transforme(char c) {
    c = toupper(c);
    return c;
}
/**
 * @brief initialise la partie
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 */
void intialiser_struct(Jeux& j, const char joueurs[]) {

    j.nb_humains = 0;
    j.nb_robots = 0;

    for (int i = 0; i < strlen(joueurs); ++i) {

        if (joueurs[i] == 'R') {
            j.nb_robots++;
        }
        else {
            j.nb_humains++;
```

```
        }
    }
    j.indice_mot = 0;
    j.nb_joueurs = j.nb_humains + j.nb_robots;
    j.joueurs = new Joueurs[j.nb_humains];
    j.robots = new Robots[j.nb_robots];
    j.nbtour = 0;
    j.joueur_perdant = 0;
    j.passage_h = 0;
    j.passage_r = 0;
    j.indice_lettre = 0;
    j.partie_fini = false;
    j.manche_fini = false;

    for (unsigned int i = 0; i < j.nb_humains; ++i) {
        j.joueurs[i].indice_joueur_humain = i;
        j.joueurs[i].nb_quart_singe = 0;
    }

    for (unsigned int i = 0; i < j.nb_robots; ++i) {
        j.robots[i].indice_joueur_robots = i;
        j.robots[i].nb_quart_de_singe = 0;
    }

    j.mot_en_cours = new char[MAX_CHAR];
    j.agrandissement_mot = MAX_CHAR;
    for (unsigned int i = 0; i < MAX_CHAR; ++i) {
        j.mot_en_cours[i] = NULL;
    }
}
/**
 * @brief verifie que le nombre et la nature des joueurs sont bons
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 * @return 1 si les conditions ne sont pas respectées, sinon 0
 */
int verif(char joueurs[]) {

    unsigned int taille = 0;
    taille = (int) strlen(joueurs);
    for (unsigned int i = 0; i < taille; ++i) {
        joueurs[i] = transforme(joueurs[i]);
    }
    for (unsigned int i = 0; i < taille; ++i) {

        if (joueurs[i] != 'R' && joueurs[i] != 'H') {
            cout << "Les lettres que vous avez entrer ne sont pas valides" <<
endl;
            return 1;
```

```
        }
    }

    if (taille < 2){
        cout << "Le nombre de joueurs doit-etre superieur ou egal a 2" <<
endl;
        return 1;
    }
    return 0;
}
/**
 * @brief Lance une partie
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 */
void partie(Jeux& j, const char joueurs[]) {

    do {
        if (j.manche_fini == false) {
            j.passage_h = 0;
            j.passage_r = 0;
        }
        for (unsigned int i = 0; i < j.nb_joueurs; ++i) {
            if (j.manche_fini) {
                i = j.joueur_perdant;
            }
            j.manche_fini = false;

            if (joueurs[i] == 'R') {
                j.manche_fini = jeux_robot(j, joueurs, i);
            }
            else {
                jeux_humain(j, joueurs, i);

                if (j.partie_fini == true || j.manche_fini == true)
                    break;
            }
        }
    } while (j.partie_fini == false);
}

/**
 * @brief affiche le nombre de quart de singe de chaque joueur et si la partie
est finie
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 * @return true si la partie est finie, sinon false
```

```
 */
bool affichage(Jeux& j, const char joueurs[]) {
    unsigned int humain = 0;
    unsigned int robot = 0;
    for (unsigned int u = 0; u < j.nb_joueurs;) {
        if (joueurs[u] == 'H') {
            cout << u + 1 << 'H' << " : " << j.joueurs[humain].nb_quart_singe;
            ++humain;
            ++u;
        }
        else {
            cout << u + 1 << "R" << " : " <<
j.robots[robot].nb_quart_de_singe;
            ++robot;
            ++u;
        }

        if (u == j.nb_joueurs) {
            cout << endl;

        }
        else
            cout << "; ";
    }
    for (unsigned int i = 0; i < j.nb_humains; ++i) {
        if (j.joueurs[i].nb_quart_singe == 1) {
            cout << "La partie est finie" << endl;
            delete[]j.mot_en_cours;
            return true;
        }
    }
    for (unsigned int i = 0; i < j.nb_robots; ++i) {
        if (j.robots[i].nb_quart_de_singe == 1) {
            cout << "La partie est finie" << endl;
            delete[]j.mot_en_cours;
            return true;
        }
    }
    delete[]j.mot_en_cours;
    j.mot_en_cours = new char[MAX_CHAR];
    j.agrandissement_mot = MAX_CHAR;
    for (unsigned int i = 0; i < MAX_CHAR; ++i) {
        j.mot_en_cours[i] = NULL;
    }
    j.indice_mot = 0;
    j.nbtour = 0;
    return false;
}
```

```
/**
 * @brief ajoute une lettre au tableau contenant le mot en cours
 * @param[in,out] j : structure de la partie
 * @param[in] i: indice du joueur actuel
 * @param[in] lettre: la lettre à ajouter
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 */
void ajout_lettre(Jeux& j, const unsigned int i, const char lettre, const char
joueurs[]) {

     if (j.indice_mot == j.agrandissement_mot - 1) {
        j.agrandissement_mot += 1;
        char* mot_en_cours = new char[j.agrandissement_mot];
        strcpy(mot_en_cours, j.mot_en_cours);
        delete[] j.mot_en_cours;
        j.mot_en_cours = mot_en_cours;
    }
     char lettre_donnee[2] = "";
    lettre_donnee[0] = lettre;
    strcat(j.mot_en_cours, lettre_donnee);
    j.indice_mot++;
    j.nbtour++;
    j.joueur_precedent = i;
}
/**
 * @brief Désallocation d'une partie
 * @param[in,out] j : structure de la partie
 */
void desallouer(Jeux& j) {
    delete[]j.joueurs;
    delete[]j.robots;
}
```

```
#ifndef _FONCTIONHUMAIN_
#define _FONCTIONHUMAIN_
#include <iostream>
#include "FonctionJeux.H"
/**
 * @brief Verifie l'existence d'un mot dans le dictionnaire
 * @param[in] mot_donne : tableau contenant les lettres composant le mot
 * @return true si le mot est present dans le dictionnaire, sinon false.
 */
```

```
bool verif_existence_mot(const char mot_donne[MAX_CHAR]);

/**
 * @brief verifie si le mot donné apres un "?" commence bien par les même
lettres que le mot en cours ou si l'humain n'a pas tapé un '!' pour abandonner
la manche
 * @param[in] mot : tableau contenant les lettres du mot donné
 * @param[in,out] j :structure de la partie
 * @return true si le mot est valide, sinon false.
 */
bool verifier_validite_mot(char mot[MAX_CHAR], Jeux& j);
/**
 * @brief Se lance quand c'est le tour d'un humain
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 * @param[in] i: indice du joueur actuel
 */
void jeux_humain(Jeux& j, const char joueurs[], const unsigned int i);

#endif
```

```
#include "FonctionHumain.h"
#include "FonctionRobot.h"
#include <iostream>
#include <fstream>
#include <iomanip>
#pragma warning(disable:4996)

using namespace std;

/**
 * @brief Verifie l'existence d'un mot dans le dictionnaire
 * @param[in] mot_donne : tableau contenant les lettres composant le mot
 * @return true si le mot est present dans le dictionnaire, sinon false.
 */
bool verif_existence_mot(const char mot_donne[MAX_CHAR]) {

    ifstream dico("dico.txt");

    if (!dico) {
        cout << "le dictionnaire n'a pu etre ouvert" << endl;
        return 0;
    }

    char mot_dico[MAX_CHAR];
    dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le premier mot
    while (dico) {
        if (strcmp(mot_dico, mot_donne) == 0) {
```

```
            if (strlen(mot_dico) > 2) {
                return true;
            }

        }
        dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le mot
suivant
    }

    dico.close();
    return false;
}

/**
 * @brief verifie si le mot donné apres un "?" commence bien par les même
lettres que le mot en cours ou si l'humain n'a pas tapé un '!' pour abandonner
la manche
 * @param[in] mot : tableau contenant les lettres du mot donné
 * @param[in,out] j :structure de la partie
 * @return true si le mot est valide, sinon false.
 */
bool verifier_validite_mot(char mot[MAX_CHAR], Jeux& j) {
    for (unsigned int i = 0; i < strlen(mot); ++i) {
        mot[i] = transforme(mot[i]);
    }
    for (unsigned int g = 0; g < strlen(j.mot_en_cours); ++g) {
        if (mot[g] == '!') {
            cout << "le joueur " << j.joueur_precedent + 1 << "H abandonne la
manche et prend un quart de singe" << endl;
            return false;
        }
        else if (j.mot_en_cours[g] != mot[g]) {
            cout << "le mot " << mot << " ne commence pas par les lettres
attendues, le joueur " << j.joueur_precedent + 1 << "H prend un quart de
singe" << endl;
            return false;
        }
    }
    return true;

}
/**
 * @brief Se lance quand c'est le tour d'un humain
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 * @param[in] i: indice du joueur actuel
 */
void jeux_humain(Jeux& j, const char joueurs[], const unsigned int i) {
```

```
    bool trouve_mot = false;
    bool mot_valide = true;
    char lettre;

    cout << i + 1 << "H, (" << j.mot_en_cours << ") > ";

    cin >> lettre;
    cin.ignore(INT_MAX, '\n');

    lettre = transforme(lettre);

    if (lettre == '?') {

        if (j.nbtour == 0) {

            cout << "Vous etes le premier joueur, lettre incorrect, un quart
de singe vous ai attribue" << endl;
            j.joueurs[j.passage_h].nb_quart_singe += 0.25;
            j.joueur_perdant = i;
            j.partie_fini = affichage(j, joueurs);
            j.manche_fini = true;
            return;

        }

        else if (joueurs[j.joueur_precedent] == 'R') {

            cout << j.joueur_precedent + 1 << "R, saisir le mot > ";
            char mot[MAX_CHAR] = "";
            if (trouver_mot(j, mot) == 0) {
                j.joueurs[j.passage_h].nb_quart_singe += 0.25;
                j.joueur_perdant = i;
                cout << mot << endl;
                cout << "le mot " << mot << " existe, le joueur " << i + 1 <<
"H prend un quart de singe" << endl;
                j.manche_fini = true;
                j.partie_fini = affichage(j, joueurs);
                return;
            }
            else {
                cout << "!" << endl;
                cout << "le joueur " << j.joueur_precedent + 1 << "R abandonne
la manche et prend un quart de singe" << endl;
                j.robots[j.joueur_precedant_indice].nb_quart_de_singe += 0.25;
                j.passage_r = j.joueur_precedant_indice;
                j.joueur_perdant = j.joueur_precedent;
                j.manche_fini = true;
                j.partie_fini = affichage(j, joueurs);
                return;
```

```
            }
        }

        cout << j.joueur_precedent + 1 << "H, saisir le mot > ";
        char mot[MAX_CHAR];
        cin >> setw(MAX_CHAR) >> mot;
        cin.ignore(INT_MAX, '\n');

        for (unsigned int f = 0; f < strlen(mot); ++f) {
            mot[f] = transforme(mot[f]);
        }

        mot_valide = verifier_validite_mot(mot, j);

        if (mot_valide == false) {
            j.joueurs[j.joueur_precedant_indice].nb_quart_singe += 0.25;
            j.joueur_perdant = j.joueur_precedent;
            j.partie_fini = affichage(j, joueurs);
            j.passage_h =j.joueur_precedant_indice;
            j.manche_fini = true;
            return;
        }

        trouve_mot = verif_existence_mot(mot);
        if (trouve_mot == true) {
            j.joueurs[j.passage_h].nb_quart_singe += 0.25;
            cout << "le mot " << mot << " existe, le joueur " << i + 1 << "H
prend un quart de singe" << endl;
            j.joueur_perdant = i;
            j.partie_fini = affichage(j, joueurs);
            j.manche_fini = true;
            return;
        }
        else {
            j.joueurs[j.joueur_precedant_indice].nb_quart_singe += 0.25;
            j.joueur_perdant = j.joueur_precedent;
            cout << "le mot " << mot << " n'existe pas, le joueur " <<
j.joueur_precedent + 1 << "H prend un quart de singe" << endl;
            j.passage_h = j.joueur_precedant_indice;
            j.manche_fini = true;
            j.partie_fini = affichage(j, joueurs);
            return;


        }
    }
    else if (lettre == '!') {
        j.joueurs[j.passage_h].nb_quart_singe += 0.25;
```

```
        cout << "le joueur " << i + 1 << "H abandonne la manche et prend un
quart de singe" << endl;
        j.joueur_perdant = i;
        j.partie_fini = affichage(j, joueurs);
        j.manche_fini = true;
        return;

    }
    else {
        ajout_lettre(j, i, lettre, joueurs);
        trouve_mot = verif_existence_mot(j.mot_en_cours);
        j.joueur_precedant_indice = j.passage_h;
        if (trouve_mot == true) {
            j.joueur_perdant = i;
            j.joueurs[j.passage_h].nb_quart_singe += 0.25;
            cout << "le mot " << j.mot_en_cours << " existe, le joueur " << i
+ 1 << "H prend un quart de singe" << endl;
            j.partie_fini = affichage(j, joueurs);
            j.manche_fini = true;
            return;
        }
        else {
            j.passage_h++;
        }

    }
}
```

```
#ifndef _FONCTIONROBOT_
#define _FONCTIONROBOT_
#include <iostream>
using namespace std;
#include "FonctionJeux.h"

/**
 * @brief Génère une lettre au hasard pour le tour du robot
 * @return une lettre au hasard
 */
char Lettre_Robot_Hasard();

/**
 * @brief Verifie si les lettres actuelle qui compose le mot en cours peuvent
potentiellement former le debut d'un mot existant
```

```
 * @param[in] j : structure de la partie
 * @return true si le mot est present dans le dictionnaire, sinon false.
 */
bool verif_existence_mot_en_cours(const Jeux& j);

/**
 * @brief trouve un mot dans le dictionnaire commençant par les lettres du mot en cours
 * @param[in] j : la partie
 * @param[in,out] a : le mot trouvé
 */
int trouver_mot(const Jeux& j, char a[MAX_CHAR]);

/**
 * @brief  Trouve toutes les lettres qui peuvent former un mot en finissant le mot en cours
 * @param[in,out] j : la partie
 */
void lettre_perdante(Jeux& j);

/**
 * @brief  Trouve toutes les lettres qui peuvent former un mot en finissant le mot en cours
 * @param[in,out] j : la partie
 */
char trouver_lettre_mot_long(const Jeux& j);

/**
 * @brief trouve l'avant derniere lettre d'un mot qui commence par les meme que le mot en cours mot ou un "?" si aucune lettre n'est trouvee
 * @param[in] j : la partie
 * @return "?" si aucune lettre n'est trouvee, sinon la lettre piege.
 */
char trouver_lettre_piege(const Jeux& j);

/**
 * @brief Se lance quand c'est le tour d'un robot
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au nombre de joueurs et leur nature
 * @param[in] indice: indice du joueur en cours
 * @return true si la manche est finie, sinon false
 */
bool jeux_robot(Jeux& j, const char joueurs[], const unsigned int indice);


#endif
```

```cpp
#include <iostream>
#include <fstream>
#include <iomanip>
#include "FonctionRobot.h"
#include "FonctionHumain.h"
#pragma warning(disable:4996)


using namespace std;

/**
 * @brief Génère une lettre au hasard pour le tour du robot
 * @return une lettre au hasard
 */
char Lettre_Robot_Hasard() {

    srand(time(NULL));

    char c = (rand() % ((90 - 65) + 1)) + 65;

    c = transforme(c);

    return c;
}
/**
 * @brief Verifie si les lettres actuelle qui compose le mot en cours peuvent
potentiellement former le debut d'un mot existant
 * @param[in] j : structure de la partie
 * @return true si le mot est present dans le dictionnaire, sinon false.
 */
bool verif_existence_mot_en_cours(const Jeux& j) {

    ifstream dico("dico.txt");

    if (!dico) {
        cout << "le dictionnaire n'a pu etre ouvert" << endl;
        return 0;
    }

    if (strlen(j.mot_en_cours) < 3) { //On prends pas en compte les mots de 2
lettres
        return true;
    }

    char mot_dico[MAX_CHAR];
    dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le premier mot
    while (dico) {
        unsigned int validite = 0;
        for (unsigned int i = 0; i < strlen(j.mot_en_cours); ++i) {
```

```
            if (mot_dico[i] == j.mot_en_cours[i]) {
                validite += 1;
            }
            if (validite == strlen(j.mot_en_cours))
                return true;
        }
        dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le mot
suivant
    }

    dico.close();
    return false;
}

/**
 * @brief trouve un mot dans le dictionnaire commençant par les lettres du mot
en cours
 * @param[in] j : la partie
 * @param[in,out] a : le mot trouvé
 */
int trouver_mot(const Jeux& j, char a[MAX_CHAR]) {

    ifstream dico("dico.txt");

    if (!dico) {
        cout << "le dictionnaire n'a pu etre ouvert" << endl;
        return 1;
    }
    char mot_dico[MAX_CHAR];
    dico >> setw(MAX_CHAR) >> mot_dico;// on essaye de lire le premier mot

    while (dico) {
        unsigned int validite = 0;
        for (unsigned int i = 0; i < j.indice_mot; ++i) {
            if (mot_dico[i] == j.mot_en_cours[i]) {
                validite += 1;
            }
            if ((validite == j.indice_mot) && strlen(mot_dico) > 2) {
                strcpy(a, mot_dico);
                return 0;
            }
        }
        dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le mot
suivant
    }
    dico.close();

    return 1;
}
```

```
/**
 * @brief  Trouve toutes les lettres qui peuvent former un mot en finissant le
mot en cours
 * @param[in,out] j : la partie
 */
void lettre_perdante(Jeux& j) {

    ifstream dico("dico.txt");
    if (!dico) {
        cout << "le dictionnaire n'a pu etre ouvert" << endl;
        return;
    }
    char mot_dico[MAX_CHAR] = "";
    dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le premier mot

    j.indice_lettre = 0;
    unsigned int nb_lettre = 2;
    j.lettre_perdante = new char[nb_lettre];
    for (unsigned int i = 0; i < nb_lettre; ++i) {
        j.lettre_perdante[i] = NULL;
    }

    while (dico) {
        unsigned int validite = 0;
        for (unsigned int i = 0; i <j.indice_mot; ++i) {
            if (mot_dico[i] == j.mot_en_cours[i]) {
                validite++;
                if (validite == j.indice_mot) {
                    if (strlen(mot_dico) > 2) {
                        if (strlen(mot_dico) == j.indice_mot + 1) {
                            if (j.indice_lettre == nb_lettre - 1) {
                                ++nb_lettre;
                                char* lettre_perdante = new char[nb_lettre];
                                for (unsigned int i = 0; i < nb_lettre; ++i) {
                                    lettre_perdante[i] = NULL;
                                }
                                strcpy(lettre_perdante, j.lettre_perdante);
                                delete[]j.lettre_perdante;
                                j.lettre_perdante = lettre_perdante;
                            }
                            j.lettre_perdante[j.indice_lettre] =
mot_dico[j.indice_mot];
                            ++j.indice_lettre;
                        }
                    }
                }
            }
        }
```

```
        dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le mot
suivant
    }
    dico.close();
    return;
}

/**
 * @brief  Trouve toutes les lettres qui peuvent former un mot en finissant le
mot en cours
 * @param[in,out] j : la partie
 */
char trouver_lettre_mot_long(const Jeux& j) {
    char mot_dico[MAX_CHAR] = "";
    ifstream dico("dico.txt");

    if (!dico) {
        cout << "le dictionnaire n'a pu etre ouvert" << endl;
        return 0;
    }
    dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le premier mot
    unsigned int taille = 0;
    taille = (int) strlen(mot_dico);
    char plus_grand[MAX_CHAR] = "";
    strcpy(plus_grand, mot_dico);
    while (dico) {
        unsigned int validite = 0;
        for (unsigned int i = 0; i < j.indice_mot; ++i) {
            if (mot_dico[i] == j.mot_en_cours[i]) {
                validite++;
                if ((validite == j.indice_mot) && strlen(mot_dico) > 2) {
                    unsigned int taille_intermediaire = 0;
                    taille_intermediaire = (int) strlen(mot_dico);
                    if (taille_intermediaire > taille) {
                        unsigned int validite2 = 0;
                        for (unsigned int u = 0; u < j.indice_lettre; ++u) {
                            if (j.lettre_perdante[u] !=
mot_dico[j.indice_mot]) {
                                validite2++;
                            }
                        }
                        if (validite2 == j.indice_lettre) {
                            taille = taille_intermediaire;
                            strcpy(plus_grand, mot_dico);
                        }
                    }
                }
            }
        }
```

```
        dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le mot
suivant
    }
    if (strcmp("AA", plus_grand) == 0) {
        return '?';
    }
    return plus_grand[j.indice_mot];
}
/**
 * @brief trouve l'avant derniere lettre d'un mot qui commence par les meme
que le mot en cours mot ou un "?" si aucune lettre n'est trouvee
 * @param[in] j : la partie
 * @return "?" si aucune lettre n'est trouvee, sinon la lettre piege.
 */
char trouver_lettre_piege(const Jeux& j) {
    char mot_dico[MAX_CHAR] = "";
    ifstream dico("dico.txt");

    if (!dico) {
        cout << "le dictionnaire n'a pu etre ouvert" << endl;
        return 0;
    }
    dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le premier mot
    char piege[MAX_CHAR] = "";
    strcpy(piege, mot_dico);
    while (dico) {
        unsigned int validite = 0;
        for (unsigned int i = 0; i < j.indice_mot; ++i) {
            if (mot_dico[i] == j.mot_en_cours[i]) {
                validite++;
                if ((validite == j.indice_mot) && (strlen(mot_dico) ==
(j.indice_mot + 2))) {// Pour pieger
                    unsigned int validite2 = 0;
                    for (unsigned int u = 0; u < j.indice_lettre; ++u) {
                        if (j.lettre_perdante[u] != mot_dico[j.indice_mot]) {
                            validite2++;
                        }
                    }
                    if (validite2 == j.indice_lettre) {
                        strcpy(piege, mot_dico);
                        return piege[j.indice_mot];
                    }
                }
            }
        }
        dico >> setw(MAX_CHAR) >> mot_dico; // on essaye de lire le mot
suivant
    }
    return '?';
```

```
}

/**
 * @brief Se lance quand c'est le tour d'un robot
 * @param[in,out] j : structure de la partie
 * @param[in] joueurs : tableau contenant les lettres qui correspondent au
nombre de joueurs et leur nature
 * @param[in] indice: indice du joueur en cours
 * @return true si la manche est finie, sinon false
 */
bool jeux_robot(Jeux& j, const char joueurs[], const unsigned int indice) {
    bool manche_fini = false;
    cout << indice + 1 << "R, (" << j.mot_en_cours << ") > ";
    if (j.indice_mot == 0) {
        char c;
        c = Lettre_Robot_Hasard();
        cout << c << endl;
        ajout_lettre(j, indice, c, joueurs);
        j.joueur_precedant_indice = j.passage_r;
        j.passage_r++;
    }

    else if (verif_existence_mot_en_cours(j) == false) {
        cout << "?" << endl;
        if (joueurs[j.joueur_precedent] == 'H') {
            char mot_donne[MAX_CHAR];
            cout << j.joueur_precedent + 1 << "H, saisir le mot > ";
            cin >> setw(MAX_CHAR) >> mot_donne; //inutile de tester le mot vu
que la fonction "verifier_existence_mot_en_cours" verifie qu'aucun mot ne
commence par les lettres donn?es.
            cin.ignore(INT_MAX, '\n');
            if (verifier_validite_mot(mot_donne, j) == true) {
                cout << "le mot " << mot_donne << " n'existe pas, " <<
j.joueur_precedent + 1 << "H prend un quart de singe" << endl;
                j.joueurs[j.joueur_precedant_indice].nb_quart_singe += 0.25;
                j.joueur_perdant = j.joueur_precedent;
            }
            else {
                j.joueurs[j.joueur_precedant_indice].nb_quart_singe += 0.25;
                j.joueur_perdant = j.joueur_precedent;
            }
            j.passage_h = j.joueur_precedant_indice;
            j.partie_fini = affichage(j, joueurs);
            manche_fini = true;
        }
    }

    else {
        lettre_perdante(j);
```

```
        char c = trouver_lettre_piege(j);
        if (c == '?') {
            char c = trouver_lettre_mot_long(j);
            if (c == '?') {
                if (joueurs[((indice + 1) % j.nb_joueurs)] == 'H') {
                    if (joueurs[j.joueur_precedent] == 'H') {
                        srand(time(NULL));
                        unsigned int i = (rand() % 2); // Une chance sur 2 de
bluffer
                        if (i == 0) {
                            char t;
                            bool lettre_valide = false;
                            do {
                                t =  Lettre_Robot_Hasard();
                                unsigned int validite = 0;
                                for (unsigned int i = 0; i < j.indice_lettre;
++i) {
                                    if (j.lettre_perdante[i] != t)
                                        validite++;
                                    if (validite == j.indice_lettre)
                                        lettre_valide = true;
                                }
                            } while (lettre_valide == false);
                            cout << t << endl;
                            ajout_lettre(j, indice, t, joueurs);
                            j.joueur_precedant_indice = j.passage_r;
                            j.passage_r++;
                        }
                        else {
                            cout << c << endl;
                            char mot_donne[MAX_CHAR];
                            cout << j.joueur_precedent + 1 << "H, saisir le
mot > ";
                            cin >> setw(MAX_CHAR) >> mot_donne;
                            cin.ignore(INT_MAX, '\n');
                            unsigned int taille = (int)strlen(mot_donne);
                            for (unsigned int f = 0; f < taille; ++f) {
                                mot_donne[f] = transforme(mot_donne[f]);
                            }

                            if (verifier_validite_mot(mot_donne, j) == true)
{

                                if (verif_existence_mot(mot_donne) == true) {
                                    j.robots[j.passage_r].nb_quart_de_singe +=
0.25;
                                    cout << "le mot " << mot_donne << "
existe, le joueur " << indice + 1 << "R prend un quart de singe" << endl;
                                    j.joueur_perdant = indice;
```

```
                                }
                                else {
                                    j.joueurs[j.joueur_precedant_indice].nb_qu
art_singe += 0.25;
                                    cout << "le mot " << mot_donne << "
n'existe pas, le joueur " << j.joueur_precedent + 1 << "H prend un quart de
singe" << endl;
                                    j.joueur_perdant = j.joueur_precedent;
                                    j.passage_h = j.joueur_precedant_indice;
                                }
                            }
                            else {
                                j.joueurs[j.joueur_precedant_indice].nb_quart_
singe += 0.25;
                                j.joueur_perdant = j.joueur_precedent;
                            }
                            j.partie_fini = affichage(j, joueurs);
                            manche_fini = true;
                        }
                    }
                    else {
                        char t;
                        bool lettre_valide = false;
                        do {
                            t = Lettre_Robot_Hasard();
                            unsigned int validite = 0;
                            for (unsigned int i = 0; i < j.indice_lettre; ++i)
{
                                if (j.lettre_perdante[i] != t)
                                    validite++;
                                if (validite == j.indice_lettre)
                                    lettre_valide = true;
                            }
                        } while (lettre_valide == false);
                        t = Lettre_Robot_Hasard();
                        cout << t << endl;
                        ajout_lettre(j, indice, t, joueurs);// On va bluffer
l'humain qui joue apr�s car avant c'est un robot qui joue
                        j.joueur_precedant_indice = j.passage_r;
                        j.passage_r++;
                    }
                }
                else {
                    if (joueurs[j.joueur_precedent] == 'H') {
                        cout << c << endl;
                        char mot_donne[MAX_CHAR];
                        cout << j.joueur_precedent + 1 << "H, saisir le mot >
";
                        cin >> setw(MAX_CHAR) >> mot_donne;
```

```
                        cin.ignore(INT_MAX, '\n');
                        for (unsigned int f = 0; f < strlen(mot_donne); ++f) {
                            mot_donne[f] = transforme(mot_donne[f]);
                        }

                        if (verifier_validite_mot(mot_donne, j) == true) {
                            if (verif_existence_mot(mot_donne) == true) {
                                j.robots[j.passage_r].nb_quart_de_singe +=
0.25;
                                cout << "le mot " << mot_donne << " existe, le
joueur " << indice + 1 << "R prend un quart de singe" << endl;
                                j.joueur_perdant = indice;
                            }
                            else {
                                j.joueurs[j.joueur_precedant_indice].nb_quart_
singe += 0.25;
                                cout << "le mot " << mot_donne << " n'existe
pas, le joueur " << j.joueur_precedent + 1 << "H prend un quart de singe" <<
endl;
                                j.joueur_perdant = j.joueur_precedent;
                                j.passage_h = j.joueur_precedant_indice;
                            }
                        }
                        else
                        {
                            j.joueurs[j.joueur_precedant_indice].nb_quart_sing
e += 0.25;
                            j.joueur_perdant = j.joueur_precedent;
                        }
                    }
                    else {
                        cout << "!" << endl;
                        j.robots[j.passage_r].nb_quart_de_singe += 0.25;
                        cout << "le joueur " << indice + 1 << "R abandonne la
manche et prend un quart de singe" << endl;
                        j.joueur_perdant = indice;
                    }
                    j.partie_fini = affichage(j, joueurs);
                    manche_fini = true;
                }


            }
            else {
                cout << c << endl;
                ajout_lettre(j, indice, c, joueurs);
                j.joueur_precedant_indice = j.passage_r;
                ++j.passage_r;
            }
```

```
        }
        else {
            cout << c << endl;
            ajout_lettre(j, indice, c, joueurs);
            j.joueur_precedant_indice = j.passage_r;
            ++j.passage_r;
        }
        delete[]j.lettre_perdante;
    }
    return manche_fini;
}
```

```
#include <iostream>
#include "FonctionJeux.h"
using namespace std;
/**
 * @brief Programme d'execution du jeu
 */

int main(int argc, char* argv[]) {
    Jeux j;

    if (verif(argv[1]) == 1)
        return 1;

    intialiser_struct(j, argv[1]);

    partie(j, argv[1]);

    desallouer(j);

    return 0;

}
```