KALIAMOORTHY Namodacane
CAROUNANITHI Alexandre

Groupe 15
112

Dossier de développement logiciel

SAE S1.02 – Initiation au développement Comparaison d'approches algorithmiques

2022/2023

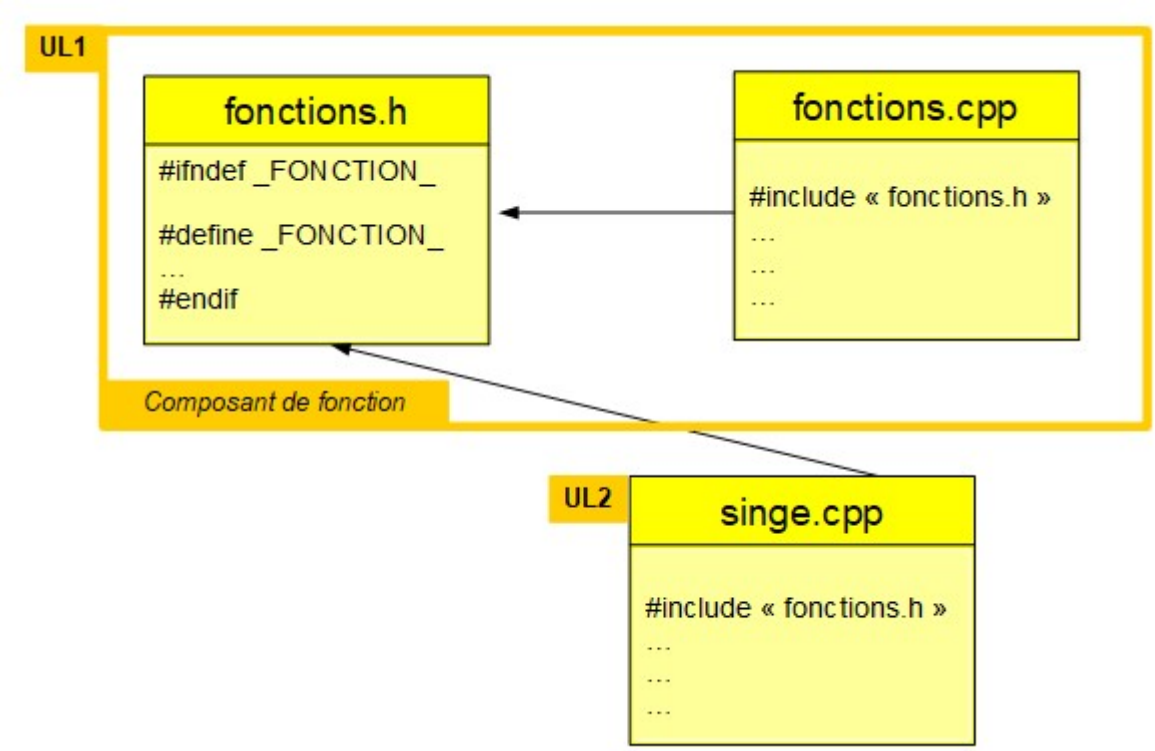
Denis Poitrenaud

Table des matières

1. Présentation du projet

Ce projet est un jeu console en C++, le Quart de Singe. Il se joue avec des humains et/ou des robots et prend en compte un dictionnaire des mots français avec plus de 300000 mots. Au cours d'une manche, chaque joueur saisie une lettre en ayant pour but de ne pas former un mot existant qui est présent dans le dictionnaire français. D'autres possibilités tels que le point d'interrogation et le point d'exclamation y figurent, ainsi que différentes règles à respecter. Ainsi, le but de ce projet consiste à la recherche d'algorithmes permettant de répondre au cahier des charges exprimé.

2. Graphe de dépendance des fichiers sources



3. Code source des tests unitaires

Pour notre premier jeu de test, nous avons créé une partie normale de quart de singe, sans cas spéciaux. Les joueurs présents sont tous des humains, toutes les lettres tapées sont en majuscule et aucunes erreurs d'inattention ne figurent dans ce jeu de test.

Paramètre : HHHH	
IN1	OUT1
R O U G E R I P ?	1H, () > 2H, (R) > 3H, (RO) > 4H, (ROU) > 1H, (ROUG) > le mot ROUGE existe, le joueur 1H prend un quart de singe 1H : 0.25; 2H : 0; 3H : 0; 4H : 0 1H, () > 2H, (R) > 3H, (RI) > 4H, (RIP) > 3H, saisir le mot > le mot AGRIPPER ne commence pas par les lettres attendues, le joueur 3H prend un quart de singe 1H : 0.25; 2H : 0; 3H : 0.25; 4H : 0
AGRIPPER O U R A G A !	3H, () > 4H, (O) > 1H, (OU) > 2H, (OUR) > 3H, (OURA) > 4H, (OURAG) > 1H, (OURAGA) > le joueur 1H abandonne la manche et prend un quart de singe 1H : 0.5; 2H : 0; 3H : 0.25; 4H : 0 1H, () > 2H, (S) > 3H, (SC) > 4H, (SCE) > 3H, saisir le mot > le mot SCEAU existe, le joueur 4H prend un quart de singe
S C E ?	1H : 0.5; 2H : 0; 3H : 0.25; 4H : 0.25 4H, () > 1H, (H) > 2H, (HA) > 3H, (HAR) > 4H, (HARP) > 3H, saisir le mot > le mot HARPENT existe, le joueur 4H prend un quart de singe
SCEAU H A R P ?	1H : 0.5; 2H : 0; 3H : 0.25; 4H : 0.5 4H, () > 1H, (F) > 2H, (FU) > le mot FUS existe, le joueur 2H prend un quart de singe 1H : 0.5; 2H : 0.25; 3H : 0.25; 4H : 0.5 2H, () > 3H, (M) > 4H, (ME) > 1H, (MED) > 2H, (MEDI) > 3H, (MEDIO) > 2H, saisir le mot > le mot MEDIORE n'existe pas, 2H prend un quart de singe
HARPENT F U S M E D I O ?	1H : 0.5; 2H : 0.5; 3H : 0.25; 4H : 0.5 2H, () > 3H, (Y) > 4H, (YU) > 1H, (YUC) > 2H, (YUCK) > 1H, saisir le mot > le mot YUCKA n'existe pas, 1H prend un quart de singe 1H : 0.75; 2H : 0.5; 3H : 0.25; 4H : 0.5 1H, () > 2H, (P) > 3H, (PA) > le mot PAR existe, le joueur 3H prend un quart de singe 1H : 0.75; 2H : 0.5; 3H : 0.5; 4H : 0.5
MEDIORE Y U	3H, () > 4H, (H) > 1H, (HI) > 2H, (HIS) > 3H, (HIST) > le joueur 3H abandonne la manche et prend un quart de singe

C K ? YUCKA P A R H I S T ! G U E	1H : 0.75; 2H : 0.5; 3H : 0.75; 4H : 0.5 3H, () > 4H, (G) > 1H, (GU) > le mot GUE existe, le joueur 1H prend un quart de singe 1H : 1; 2H : 0.5; 3H : 0.75; 4H : 0.5 La partie est finie

Pour notre second jeu de test, nous avons ajouté certaines erreurs. Par exemple, les joueurs ont tapé plusieurs caractères au lieu d'un, certains de ces caractères sont en minuscule, d'autres en majuscule. Également, nous avons testé le cas de la saisie d'un point d'interrogation lors du premier tour, ainsi que le cas du dernier joueur / premier joueur pour vérifier si la saisie et les points ont bien été attribué.

Paramètre : HHHHH	
IN2	OUT2
B I A ? BIAFINE O R A CL LE E l o NS! U ? LONGUE Q U A T ? QUATERBACK T U	1H, () > 2H, (B) > 3H, (BI) > 4H, (BIA) > 3H, saisir le mot > le mot BIAFINE n'existe pas, 3H prend un quart de singe 1H : 0; 2H : 0; 3H : 0.25; 4H : 0; 5H : 0 3H, () > 4H, (O) > 5H, (OR) > 1H, (ORA) > 2H, (ORAC) > 3H, (ORACL) > le mot ORACLE existe, le joueur 3H prend un quart de singe 1H : 0; 2H : 0; 3H : 0.5; 4H : 0; 5H : 0 3H, () > 4H, (L) > 5H, (LO) > 1H, (LON) > 2H, (LONU) > 1H, saisir le mot > le mot LONGUE ne commence pas par les lettres attendues, le joueur 1H prend un quart de singe 1H : 0.25; 2H : 0; 3H : 0.5; 4H : 0; 5H : 0 1H, () > 2H, (Q) > 3H, (QU) > 4H, (QUA) > 5H, (QUAT) > 4H, saisir le mot > le mot QUATERBACK n'existe pas, 4H prend un quart de singe 1H : 0.25; 2H : 0; 3H : 0.5; 4H : 0.25; 5H : 0 4H, () > 5H, (T) > 1H, (TU) > 5H, saisir le mot > le mot TUE existe, le joueur 1H prend un quart de singe 1H : 0.5; 2H : 0; 3H : 0.5; 4H : 0.25; 5H : 0 1H, () > 2H, (T) > 3H, (TR) > 4H, (TRA) > 5H, (TRAP) > 4H, saisir le mot > le mot TAPER ne

? TUE T R A p ? TAPeR F I ? FIRPE! ? SUPER A S ? ASCENCEUR O! p iE ?? OPIACE ASI I G O ? AIGRE D ! ? HIPER pA! a a m ? pameau p Y R A m I D E J i	commence pas par les lettres attendues, le joueur 4H prend un quart de singe 1H : 0.5; 2H : 0; 3H : 0.5; 4H : 0.5; 5H : 0 4H, () > 5H, (F) > 1H, (FI) > 5H, saisir le mot > le mot FIRPE! n'existe pas, 5H prend un quart de singe 1H : 0.5; 2H : 0; 3H : 0.5; 4H : 0.5; 5H : 0.25 5H, () > 4H, saisir le mot > le mot SUPER existe, le joueur 5H prend un quart de singe 1H : 0.5; 2H : 0; 3H : 0.5; 4H : 0.5; 5H : 0.5 5H, () > 1H, (A) > 2H, (AS) > 1H, saisir le mot > le mot ASCENCEUR n'existe pas, 1H prend un quart de singe 1H : 0.75; 2H : 0; 3H : 0.5; 4H : 0.5; 5H : 0.5 1H, () > 2H, (O) > 3H, (OP) > 4H, (OPI) > 3H, saisir le mot > le mot OPIACE existe, le joueur 4H prend un quart de singe 1H : 0.75; 2H : 0; 3H : 0.5; 4H : 0.75; 5H : 0.5 4H, () > 5H, (A) > 1H, (AI) > 2H, (AIG) > 3H, (AIGO) > 2H, saisir le mot > le mot AIGRE ne commence pas par les lettres attendues, le joueur 2H prend un quart de singe 1H : 0.75; 2H : 0.25; 3H : 0.5; 4H : 0.75; 5H : 0.5 2H, () > 3H, (D) > le joueur 3H abandonne la manche et prend un quart de singe 1H : 0.75; 2H : 0.25; 3H : 0.75; 4H : 0.75; 5H : 0.5 3H, () > 2H, saisir le mot > le mot HIPER n'existe pas, 2H prend un quart de singe 1H : 0.75; 2H : 0.5; 3H : 0.75; 4H : 0.75; 5H : 0.5 2H, () > 3H, (P) > 4H, (PA) > 5H, (PAA) > 1H, (PAAM) > 5H, saisir le mot > le mot PAMEAU ne commence pas par les lettres attendues, le joueur 5H prend un quart de singe 1H : 0.75; 2H : 0.5; 3H : 0.75; 4H : 0.75; 5H : 0.75 5H, () > 1H, (P) > 2H, (PY) > 3H, (PYR) > 4H, (PYRA) > 5H, (PYRAM) > 1H, (PYRAMI) > 2H, (PYRAMID) > le mot PYRAMIDE existe, le joueur 2H prend un quart de singe 1H : 0.75; 2H : 0.75; 3H : 0.75; 4H : 0.75; 5H : 0.75 2H, () > 3H, (J) > 4H, (JI) > 3H, saisir le mot > le mot JIPITER n'existe pas, 3H prend un quart de singe 1H : 0.75; 2H : 0.75; 3H : 1; 4H : 0.75; 5H : 0.75 La partie est finie

? JIPITER	

Pour notre dernier test, nous nous sommes intéressés au comportement des robots lors d'une partie composée uniquement de robots. Voici le résultat d'un des tests réalisés :

Paramètre : RRRR
OUT
1R, () > L 2R, (L) > O 3R, (LO) > U 4R, (LOU) > V 1R, (LOUV) > O 2R, (LOUVO) > Y 3R, (LOUVOY) > O 4R, (LOUVOYO) > ? 3R, saisir le mot > LOUVOYONS le mot LOUVOYONS existe, le joueur 4R prend un quart de singe 1R : 0; 2R : 0; 3R : 0; 4R : 0.25 4R, () > M 1R, (M) > Y 2R, (MY) > X 3R, (MYX) > O 4R, (MYXO) > E 1R, (MYXOE) > D 2R, (MYXOED) > E 3R, (MYXOEDE) > M 4R, (MYXOEDEM) > A 1R, (MYXOEDEMA) > T 2R, (MYXOEDEMAT) > E 3R, (MYXOEDEMATE) > U 4R, (MYXOEDEMATEU) > ? 3R, saisir le mot > MYXOEDEMATEUX le mot MYXOEDEMATEUX existe, le joueur 4R prend un quart de singe 1R : 0; 2R : 0; 3R : 0; 4R : 0.5 4R, () > C 1R, (C) > H 2R, (CH) > R 3R, (CHR) > O 4R, (CHRO) > N 1R, (CHRON) > I 2R, (CHRONI) > Q 3R, (CHRONIQ) > U 4R, (CHRONIQU) > I 1R, (CHRONIQUI) > E 2R, (CHRONIQUIE) > ? 1R, saisir le mot > CHRONIQUIEZ le mot CHRONIQUIEZ existe, le joueur 2R prend un quart de singe 1R : 0; 2R : 0.25; 3R : 0; 4R : 0.5

2R, () > Y
3R, (Y) > O
4R, (YO) > Y
1R, (YOY) > O
2R, (YOYO) > T
3R, (YOYOT) > T
4R, (YOYOTT) > I
1R, (YOYOTTI) > O
2R, (YOYOTTIO) > ?
1R, saisir le mot > YOYOTTIONS
le mot YOYOTTIONS existe, le joueur 2R prend un quart de singe
1R : 0; 2R : 0.5; 3R : 0; 4R : 0.5
2R, () > C
3R, (C) > A
4R, (CA) > M
1R, (CAM) > I
2R, (CAMI) > O
3R, (CAMIO) > ?
2R, saisir le mot > CAMIONNANT
le mot CAMIONNANT existe, le joueur 3R prend un quart de singe
1R : 0; 2R : 0.5; 3R : 0.25; 4R : 0.5
3R, () > P
4R, (P) > E
1R, (PE) > L
2R, (PEL) > L
3R, (PELL) > I
4R, (PELLI) > C
1R, (PELLIC) > U
2R, (PELLICU) > L
3R, (PELLICUL) > I
4R, (PELLICULI) > E
1R, (PELLICULIE) > ?
4R, saisir le mot > PELLICULIEZ
le mot PELLICULIEZ existe, le joueur 1R prend un quart de singe
1R : 0.25; 2R : 0.5; 3R : 0.25; 4R : 0.5
1R, () > O
2R, (O) > Z
3R, (OZ) > E
4R, (OZE) > N
1R, (OZEN) > ?
4R, saisir le mot > OZENES
le mot OZENES existe, le joueur 1R prend un quart de singe
1R : 0.5; 2R : 0.5; 3R : 0.25; 4R : 0.5
1R, () > O
2R, (O) > N
3R, (ON) > G
4R, (ONG) > U
1R, (ONGU) > I
2R, (ONGUI) > C

3R, (ONGUIC) > U
4R, (ONGUICU) > L
1R, (ONGUICUL) > ?
4R, saisir le mot > ONGUICULEE
le mot ONGUICULEE existe, le joueur 1R prend un quart de singe
1R : 0.75; 2R : 0.5; 3R : 0.25; 4R : 0.5
1R, () > Y
2R, (Y) > O
3R, (YO) > Y
4R, (YOY) > O
1R, (YOYO) > T
2R, (YOYOT) > T
3R, (YOYOTT) > O
4R, (YOYOTTO) > ?
3R, saisir le mot > YOYOTTONS
le mot YOYOTTONS existe, le joueur 4R prend un quart de singe
1R : 0.75; 2R : 0.5; 3R : 0.25; 4R : 0.75
4R, () > Y
1R, (Y) > T
2R, (YT) > T
3R, (YTT) > R
4R, (YTTR) > I
1R, (YTTRI) > U
2R, (YTTRIU) > ?
1R, saisir le mot > YTTRIUM
le mot YTTRIUM existe, le joueur 2R prend un quart de singe
1R : 0.75; 2R : 0.75; 3R : 0.25; 4R : 0.75
2R, () > B
3R, (B) > L
4R, (BL) > I
1R, (BLI) > N
2R, (BLIN) > Q
3R, (BLINQ) > U
4R, (BLINQU) > I
1R, (BLINQUI) > E
2R, (BLINQUIE) > ?
1R, saisir le mot > BLINQUIEZ
le mot BLINQUIEZ existe, le joueur 2R prend un quart de singe
1R : 0.75; 2R : 1; 3R : 0.25; 4R : 0.75
La partie est finie

4. Bilan du projet

a. Difficultés rencontrées.

Durant ce projet, nous avons rencontré plusieurs difficultés que nous avons réussi à surmonter au fur et à mesure. Ci-dessous, vous retrouverez dans l'ordre toutes les difficultés rencontrées :

- Difficulté à trouver la bonne structure, nous sommes partis d'une structure très basique faisant intervenir uniquement le type du joueur ainsi que son score. Toutefois, nous avons vu au fur et à mesure que nous avions besoin de nouvelles caractéristiques pour chaque joueur, tels que « tourActuel » qui permet de savoir à quel tour le joueur doit jouer, « posLettre » pour savoir à quel indice la lettre saisie doit être placée...
- Difficulté à trouver l'algorithme du robot puisque le robot joue de manière totalement indépendante contrairement aux humains. Le codage de fonctions pour les humains et pour les robots sont différentes l'un de l'autre. C'est pourquoi nous devions penser à tout. Par exemple, il fallait rendre le robot plus fort mais pas imbattable, savoir quand il devait saisir une lettre, un mot, un point d'interrogation ou un point d'exclamation.
- Difficulté à rechercher plus rapidement un mot dans le dictionnaire. Pour cela, nous avons commencé par mettre le dictionnaire en mémoire, rendant la recherche assez rapide. Ensuite, nous avons réussi à rendre cette recherche encore plus rapide avec la méthode dichotomique.
- Concernant la fonction point d'interrogation, nous avons eu certaines difficultés pour la saisie du dernier joueur lorsque le tout premier joueur place un point d'interrogation. L'attribution des points n'était pas la même, c'est pourquoi nous avons rajouté certaines modifications pour ce cas spécial.
- Pour finir, nous avons également eu quelques difficultés concernant la suppression des allocations mémoire à la toute fin d'une manche aussi il fallait penser à toutes les allocations faites depuis le début (p.Joueurs, p.motTap...).

b. Ce qui est réussi.

Malgré les difficultés évoquées, notre quart de singe fonctionne correctement et est conforme au cahier de charges exprimé. Globalement, les saisies humaines et robots sont réussies. L'attribution des points de chaque joueur ainsi que les fonctions tels que le point d'interrogation ou le point d'exclamation fonctionnent correctement. Tous les cas de figures

ont été testé et vérifié grâce à nos différents tests unitaires ci-dessus. Également, le robot a été optimisé de manière à ce qu'il soit imbattable mais pas trop et la recherche du mot dans le dictionnaire est assez rapide.

c. Ce qui peut être amélioré.

Outre le fait que notre programme fonctionne correctement, il peut être amélioré sur certains points notamment sur le code en lui-même puisque nous avons pas mal de fonctions dans notre programme. Globalement, toutes les améliorations que nous pensions faire ont été réalisé durant ce projet (le tableau dynamique pour le mot tapé, l'optimisation du robot, la recherche par dichotomie).

Annexe du dossier

```
/**
 * @file singe.cpp
 * Projet SAE102 - Comparaison d'approches algorithmiques
 * @author Namodacane KALIAMOORTHY et Alexandre CAROUNANITHI
 * @version 15 - 02/01/2023
 * @brief Programme principal du jeu : Le Quart du Singe
 */

#include <iostream>
#include <cstdlib>
#include <locale>

#include "fonctions.h"

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

    setlocale(LC_CTYPE, "fra");
    srand(time(NULL));

    Partie p;

    if (!verifNbJoueur(argv)) {
        std::cout << "Nombre insuffisant de joueurs" << std::endl;
        return 2;
    }
    else {
        if (verifJoueur(argv)) {
            std::cout << "Seul les joueurs humains et robots sont acceptés"
<< std::endl;
            return 2;
        }
        else {
            initialiserPartie(p, argv);
            jouerPartie(p);
            detruirePartie(p);
        }
    }

    return 0;
}
```

```
#pragma once

#ifndef _FONCTIONS_
#define _FONCTIONS_

/**
 * @file fonctions.h
 * @brief Entête du composant de fonctions
 */

 /**
 * @brief Les constantes
 */
enum {
    MIN_JOUEURS = 2,
    MAX = 28,
};

/**
* @brief Structure de données de type Dico
*/
struct Dico {
    char** mots;
    unsigned int nbMot;
};
/**
* @brief Structure de données de type Joueur
*/
struct Joueur {
    char type; // H ou R
    int score;
};
/**
* @brief Structure de données de type Partie
*/
struct Partie {
    Joueur* joueurs;
    unsigned int nbJoueurs;
    char* motTap; // Lettres annoncées pendant la partie
    unsigned int tailleMot;
    unsigned int posLettre; // Position où on doit insérer la lettre dans le
mot
    unsigned int tourActuel; // Indice du joueur dans le tableau joueurs
    char* motTapVerif;
    Dico d;
};

/**
* @brief Verification du nombre de joueur
* @param[in] argv : L'argument utilisé pour compter le nombre de joueur
* @return retourne un booléan
* @pre argv valide
*/
bool verifNbJoueur(const char* argv[]);
/**
* @brief Vérifie le type de joueur
* @param[in] argv : L'argument utilisé pour vérifier le type de joueur
* @return Retourne un booléen
* @pre argv doit être valide
*/
bool verifJoueur(const char* argv[]);
/**
* @brief Charge le dictionnaire de mots
* @param[in,out] p: La partie à laquelle le dictionnaire est associé
```

```
* @pre p est valide
*/
void initialiserDico(Partie& p);
/**
* @brief Initialise une partie
* @param[in,out] p: La partie à initialiser
* @param[in] argv : Les arguments de la commande utilisés pour initialiser la
partie
* @pre argv doit être valide
* @see initialiserDico
*/
void initialiserPartie(Partie& p, const char* argv[]);
/**
* @brief Vérifie si le mot tapé par le joueur est vide
* @param[in] p: La partie en cours
* @return Vrai si le mot tapé est vide, faux sinon
*/
bool estZeroLettre(Partie& p);
/**
* @brief Vérifie si le mot tapé par le joueur n'a qu'une seule lettre
* @param[in] p: La partie en cours
* @return Vrai si le mot tapé n'a qu'une seule lettre, faux sinon
*/
bool estPremiereLettre(Partie& p);
/**
* @brief Vérifie si le mot tapé par le joueur n'a que deux lettres ou qu'une
lettre suivie d'un '?'
* @param[in] p: La partie en cours
* @return Vrai si le mot tapé n'a que deux lettres ou qu'une lettre suivie
d'un '?', faux sinon
*/
bool estDeuxLettre(Partie& p);
/**
* @brief Vérifie si un mot est valide (si il est présent dans le dictionnaire
de la partie)
* @param[in] p: La partie en cours
* @param[in] mot: Le mot à vérifier
* @return Vrai si le mot est valide, faux sinon
*/
bool estMotValide(Partie& p, char* mot);
/**
* @brief Agrandi la taille de la chaîne de caractère du mot tapé
* @param[in,out] p: La partie en cours
*/
void agrandir(Partie& p);
/**
* @brief Ajoute une lettre au mot tapé par le joueur
* @param[in,out] p: La partie en cours
* @param[in] c: La lettre à ajouter
*/
void ajoutLettre(Partie& p, char c);
/**
* @brief Demande à l'utilisateur de saisir une lettre et l'ajoute au mot tapé
par le joueur
* @param[in,out] p: La partie en cours
*/
void saisiHumain(Partie& p);
/**
* @brief Vérifie si un caractère est une voyelle
* @param[in] c: Le caractère à vérifier
* @return Vrai si c est une voyelle, faux sinon
*/
bool estVoyelle(const char c);
/**
```

```
* @brief Vérifie si un caractère est une consonne
* @param[in] c: Le caractère à vérifier
* @return Vrai si c est une consonne, faux sinon
* @see estVoyelle
*/
bool estConsonne(const char c);
/**
* @brief Vérifie si le dernier caractère tapé par le joueur est un point
d'interrogation
* @param[in,out] p: La partie en cours
* @return Vrai si le dernier caractère est un point d'interrogation, faux
sinon
*/
bool ptInterrogation(Partie& p);
/**
* @brief Le robot tape un mot dans le cas où l'humain a tapé un point
d'interrogation
* @param[in,out] p: La partie en cours
*/
void casPtInterroR(Partie& p);
/**
* @brief Cas du robot où il tape une lettre
* @param[in,out] p: La partie en cours
*/
void casNormalSaisiR(Partie& p);
/**
* @brief Génère une lettre aléatoire pour le robot
* @param[in,out] p: La partie en cours
*/
void saisiRobot(Partie& p);
/**
* @brief Fait jouer le joueur en cours
* @param[in,out] p: La partie en cours
*/
void jouer(Partie& p);
/**
* @brief Affiche le mot tapé par le joueur courant dans la partie en cours
* @param[in] p: La partie en cours
*/
void motSaisi(Partie& p);
/**
* @brief Affiche l'état de la partie en cours
* @param[in,out] p: La partie en cours
*/
void afficher(Partie& p);
/**
* @brief Fonction permettant de mettre à jour le score d'un joueur
* @param[in] p: La partie en cours
* @param[in] i: L'indice du joueur dont on veut mettre à jour le score
*/
void score(Partie& p, unsigned int i);
/**
* @brief Affiche les scores des joueurs
* @param[in] p: La partie en cours
*/
void afficheScore(Partie& p);
/**
* @brief Ajoute un point au score du joueur en cours
* @param[in,out] p: La partie en cours
*/
void ajouteScore(Partie& p);
/**
* @brief Réinitialiser la manche en cours
* @param[in,out] p : La partie en cours
```

```
*/
void resetManche(Partie& p);
/**
* @brief Message si le mot tapé par le joueur existe dans le dictionnaire
* @param[in,out] p : La partie en cours
*/
void motExiste(Partie& p);
/**
* @brief Message si le mot tapé par le joueur n'est pas le même que le mot
tapé
* @param[in,out] p : La partie en cours
*/
void lettresDifferentes(Partie& p);
/**
* @brief Message si le mot tapé par le joueur existe
* @param[in,out] p : La partie en cours
*/
void motExisteVerif(Partie& p);
/**
* @brief Message si le mot tapé par le joueur n'existe pas
* @param[in,out] p : La partie en cours
*/
void motExistePas(Partie& p);
/**
* @brief Saisir un mot pour verifier s'il existe
* @param[in,out] p : La partie en cours
*/
void saisirMotTapVerif(Partie& p);
/**
* @brief Message si le joueur tape un point d'exclamation
* @param[in,out] p : La partie en cours
*/
void exclamation(Partie& p);
/**
* @brief Verifie si le caractère tapé est un point d'exclamation
* @param[in] p : La partie en cours
* @return Vrai si le dernier caractère est un point d'exclamation, faux sinon
*/
bool ptExclamation(Partie& p);
/**
* @brief Verifie si les caractère tapé sont les mêmes que le mot tapé
* @param[in] p : La partie en cours
* @return retourne un booléen
*/
bool verifLettres(Partie& p);
/**
* @brief Cas où c'est un point d'interrogation, on fais les vérifications
* @param[in,out] p : La partie en cours
*/
void verifInterrogation(Partie& p);
/**
* @brief Cas où c'est un point d'exclamation, on fait les vérifications
* @param[in,out] p : La partie en cours
*/
void verifExclamation(Partie& p);
/**
* @brief Cas normal, on regarde si le mot existe
* @param[in,out] p : La partie en cours
*/
void verifNormal(Partie& p);
/**
* @brief Fonction reunissant toutes les verifications
* @param[in,out] p : La partie en cours
*/
```

```
void verification(Partie& p);
/**
* @brief Le jeu
* @param[in, out] p : La partie en cours
*/
void jouerPartie(Partie& p);
/**
* @brief Détruit une partie et libère les ressources associées
* @param[in,out] p: La partie à détruire
*/
void detruirePartie(Partie& p);


#endif // !_FONCTIONS_
```

```
/**
 * @file fonctions.cpp
 * @brief Entête du composant de fonctions
 */

#include <iostream>
#include <fstream>
#include <iomanip>
#include <locale>
#include <cstring>

#include "fonctions.h"

#pragma warning(disable:4996,6385)

using namespace std;

/**
* @brief Vérification du nombre de joueur
* @param[in] argv : L'argument utilisé pour compter le nombre de joueur
* @return retourne un booléan
* @pre argv valide
*/
bool verifNbJoueur(const char* argv[]) {
    // Vérifier que le tableau d'arguments contient au moins un élément
    if (argv == nullptr || argv[0] == nullptr) {
        return false;
    }

    const char* param = argv[1];

    // Vérifier que la chaîne de caractères représentant le nombre de joueurs
est de longueur minimale MIN_JOUEURS
    if (param == nullptr || strlen(param) < MIN_JOUEURS) {
        return false;
    }

    return true;
}
/**
* @brief Vérifie le type de joueur
* @param[in] argv : L'argument utilisé pour vérifier le type de joueur
* @return Retourne un booléen
* @pre argv doit être valide
*/
bool verifJoueur(const char* argv[]) {
    // Vérifier que le tableau d'arguments contient au moins un élément
```

```
    if (argv == nullptr || argv[0] == nullptr) {
        return false;
    }

    const char* param = argv[1];
    char type;

    // Vérifier que la chaîne de caractères représentant le type de joueurs
n'est pas vide
    if (param == nullptr || strlen(param) == 0) {
        return false;
    }

    // Vérifier que chaque caractère de la chaîne est soit "H" soit "R"
    for (unsigned int i = 0; i < strlen(param); ++i) {
        type = toupper(param[i]);
        if (type != 'H' && type != 'R') {
            return true;
        }
    }

    return false;
}
/**
* @brief Charge le dictionnaire de mots
* @param[in,out] p: La partie à laquelle le dico est associé
* @pre p est valide
*/
void initialiserDico(Partie& p) {
    p.d.nbMot = 0;

    //Ouverture du dico
    ifstream dico("./ods4.txt");
    if (!dico.good()) cout << "Dico pas ouvert";

    //Se placer au début du fichier
    dico.clear();
    dico.seekg(0, ios::beg);

    //Compte le nombre de mots dans le dico
    char* mot = new char[MAX];
    mot[0] = '\0';
    p.d.nbMot = 0;
    while (dico >> setw(MAX) >> mot) {
        p.d.nbMot++;
    }
    delete[] mot;

    //Se placer au début du fichier
    dico.clear();
    dico.seekg(0, ios::beg);

    // Allouer un tableau de mots
    char* motTemp;
    motTemp = new char[MAX];
    p.d.mots = new char* [p.d.nbMot];
    for (unsigned int i = 0; i < p.d.nbMot ; ++i) {
        dico >> setw(MAX) >> motTemp;
        p.d.mots[i] = new char[strlen(motTemp)+1];
        strcpy(p.d.mots[i],motTemp);
    }
    delete[] motTemp;

    // Fermer le fichier
```

```
    dico.close();
}
/**
* @brief Initialise une partie
* @param[in,out] p: La partie à initialiser
* @param[in] argv : Les arguments de la commande utilisés pour initialiser la
partie
* @pre argv doit être valide
* @see initialiserDico
*/
void initialiserPartie(Partie& p, const char* argv[]) {

    const char* param = argv[1];
    p.nbJoueurs = (unsigned int)strlen(param);

    p.joueurs = new Joueur[p.nbJoueurs];

    //Initialisation des types des joueurs
    for (unsigned int i = 0; i < p.nbJoueurs; ++i) {
        p.joueurs[i].type = toupper(param[i]);
        p.joueurs[i].score = 0;
    }

    p.tourActuel = 0;
    p.tailleMot = 2;


    p.motTap = new char[p.tailleMot + 1];
    p.motTap[0] = '\0';
    p.posLettre = 0;
    p.motTapVerif = new char[MAX];
    p.motTapVerif[0] = '\0';

    //Allocation du dico en mémoire
    initialiserDico(p);
}
/**
* @brief Vérifie si le mot tapé par le joueur est vide
* @param[in] p: La partie en cours
* @return Vrai si le mot tapé est vide, faux sinon
*/
bool estZeroLettre(Partie& p) {
    return (p.motTap[0] == '\0');
}
/**
* @brief Vérifie si le mot tapé par le joueur n'a qu'une seule lettre
* @param[in] p: La partie en cours
* @return Vrai si le mot tapé n'a qu'une seule lettre, faux sinon
*/
bool estPremiereLettre(Partie& p) {
    return (p.motTap[0] == '\0' || p.motTap[1] == '\0');
}
/**
* @brief Vérifie si le mot tapé par le joueur n'a que deux lettres ou qu'une
lettre suivie d'un '?'
* @param[in] p: La partie en cours
* @return Vrai si le mot tapé n'a que deux lettres ou qu'une lettre suivie
d'un '?', faux sinon
*/
bool estDeuxLettre(Partie& p) {
    return (p.motTap[0] == '\0' || p.motTap[1] == '\0' || p.motTap[2] == '\0'
|| p.motTap[2] == '?');
}
/**
```

```
* @brief Vérifie si un mot est valide (si il est présent dans le dictionnaire
de la partie)
* @param[in] p: La partie en cours
* @param[in] mot: Le mot à vérifier
* @return Vrai si le mot est valide, faux sinon
*/
bool estMotValide(Partie& p, char* mot) {

    int min = 0;
    int max = p.d.nbMot - 1;
    int milieu;
    while (min <= max){
        milieu = (min + max) / 2;
        if (strcmp(p.d.mots[milieu], mot) == 0) return true;
        else if (strcmp(p.d.mots[milieu], mot) < 0) min = milieu + 1;
        else max = milieu - 1;
    }
    return false;

}
/**
* @brief Agrandi la taille de la chaîne de caractère du mot tapé
* @param[in,out] p: La partie en cours
*/
void agrandir(Partie& p) {

    unsigned int newMax = p.tailleMot + 1;

    char* newMot = new char[newMax+1];
    strcpy(newMot, p.motTap);

    delete[] p.motTap;
    p.motTap = new char[newMax + 1];
    strcpy(p.motTap, newMot);

    p.tailleMot = newMax;

    delete[] newMot;

}

/**
* @brief Ajoute une lettre au mot tapé par le joueur
* @param[in,out] p: La partie en cours
* @param[in] c: La lettre à ajouter
*/
void ajoutLettre(Partie& p, char c) {

    if (p.posLettre >= p.tailleMot) {
        agrandir(p);
    }

    p.motTap[p.posLettre] = toupper(c); // Majuscule
    p.motTap[p.posLettre + 1] = '\0';
    ++p.posLettre;
}
/**
* @brief Demande à l'utilisateur de saisir une lettre et l'ajoute au mot tapé
par le joueur
* @param[in,out] p: La partie en cours
*/
void saisiHumain(Partie& p) {
    char c;
    cin >> c;
```

```
    ajoutLettre(p, c);
    cin.ignore(INT_MAX, '\n');
}
/**
* @brief Vérifie si un caractère est une voyelle
* @param[in] c: Le caractère à vérifier
* @return Vrai si c est une voyelle, faux sinon
*/
bool estVoyelle(const char c) {
    const char voyelle[6] = { 'a', 'e', 'i', 'o', 'u', 'y' };
    for (unsigned int i = 0; i < 6; ++i) {
        if (c == toupper(voyelle[i])) return true;
    }
    return false;
}
/**
* @brief Vérifie si un caractère est une consonne
* @param[in] c: Le caractère à vérifier
* @return Vrai si c est une consonne, faux sinon
* @see estVoyelle
*/
bool estConsonne(const char c) {
    return !estVoyelle(c);
}
/**
* @brief Vérifie si le dernier caractère tapé par le joueur est un point
d'interrogation
* @param[in,out] p: La partie en cours
* @return Vrai si le dernier caractère est un point d'interrogation, faux
sinon
*/
bool ptInterrogation(Partie& p) {
    if (estZeroLettre(p)) {
        if (p.motTap[p.posLettre] == '?') return true;
        else return false;
    }
    else {
        if (p.motTap[p.posLettre - 1] == '?') return true;
        else return false;
    }
}
/**
* @brief Le robot tape un mot dans le cas où le joueur a tapé un point
d'interrogation
* @param[in,out] p: La partie en cours
*/
void casPtInterroR(Partie& p) {
    char c = '\0';
    if (estPremiereLettre(p)) {
        char motRobot[] = { "ABRUTI" };
        for (unsigned int i = 0; i < strlen(motRobot); ++i) {
            p.motTapVerif[i] = toupper(motRobot[i]);
        }
        p.motTapVerif[strlen(motRobot)] = '\0';
        cout << p.motTapVerif << endl;
    }
    else {

        // On regarde les premières lettres du mot tapé
        char* prefixe = new char[MAX];
        for (unsigned int i = 0; i < p.posLettre - 1; i++) {
            prefixe[i] = p.motTap[i];
        }
        prefixe[p.posLettre - 1] = '\0';
```

```
        unsigned int lenghPref = strlen(prefixe);

        // On compte le nombre de mots qui commencent par ce préfixe
        unsigned int nbMots = 0;
        for (unsigned int i = 0; i < p.d.nbMot; i++) {
            if (strncmp(p.d.mots[i], prefixe, lenghPref) == 0) {
                nbMots++;
            }
        }

        if (nbMots > 0) {
            unsigned int motChoisi = rand() % nbMots;
            unsigned int motCompte = 0;

            do {
                for (unsigned int i = 0; i < p.d.nbMot; i++) {
                    if (strncmp(p.d.mots[i], prefixe, lenghPref) == 0) { //
Trouve d'abord la partie du dico où commence ce prefixe
                        if (motCompte == motChoisi) { // Vérifie si c'est le
mot choisi
                            prefixe = p.d.mots[i];
                            break;
                        }
                        motCompte++;
                    }
                }

            } while (!estMotValide(p, prefixe));

            unsigned int longueur = strlen(prefixe);
            for (unsigned int i = 0; i < longueur; ++i) {
                p.motTapVerif[i] = prefixe[i];
            }
            p.motTapVerif[strlen(prefixe)] = '\0';
            cout << p.motTapVerif << endl;
        }
        else {
            c = '!';
            cout << c << endl;
            p.motTapVerif[0] = c;
            p.motTapVerif[1] = '\0';
        }

    }
}
/**
* @brief Cas du robot où il tape une lettre
* @param[in,out] p: La partie en cours
*/
void casNormalSaisiR(Partie& p) {
    char c = '\0';
    if (estZeroLettre(p)) {
        char lettres[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        unsigned int index = rand() % 26;
        c = lettres[index];
        cout << c << endl;
        ajoutLettre(p, c);
    }
    else {

        unsigned int nbTest = 0;

        // On regarde les premières lettres du mot tapé
        char* prefixe = new char[MAX];
```

```
        for (unsigned int i = 0; i < p.posLettre; i++) {
            prefixe[i] = p.motTap[i];
        }
        prefixe[p.posLettre] = '\0';
        unsigned int lenghPref = strlen(prefixe);

        // On compte le nombre de mots qui commencent par ce préfixe
        unsigned int nbMots = 0;
        for (unsigned int i = 0; i < p.d.nbMot; i++) {
            if (strncmp(p.d.mots[i], prefixe, lenghPref) == 0) {
                nbMots++;
            }
        }

        // Si il y a au moins un mot qui commence par ce préfixe, on choisit
une lettre aléatoirement parmi les mots qui commencent par ce préfixe
        if (nbMots > 0) {
            unsigned int motChoisi = rand() % nbMots;
            unsigned int motCompte = 0;

            do {
                for (unsigned int i = 0; i < p.d.nbMot; i++) {
                    if (strncmp(p.d.mots[i], prefixe, lenghPref) == 0) { //
Trouve d'abord la partie du dico où commence ce prefixe
                        if (motCompte == motChoisi) { // Vérifie si c'est le
mot choisi
                            c = p.d.mots[i][p.posLettre]; // Récupère la
lettre suivante du mot choisi
                            break;
                        }
                        motCompte++;
                    }
                }


                prefixe[p.posLettre] = toupper(c); // Majuscule
                prefixe[p.posLettre + 1] = '\0';
                if (nbMots <= 3 && estMotValide(p, prefixe)) {
                    c = '?';
                    break;
                }
                else {
                    motChoisi = rand() % nbMots;
                    motCompte = 0;
                }

                if (estPremiereLettre(p)) break;

                ++nbTest;

                if (nbMots > 2000) {
                    if (nbTest > nbMots / 4) {
                        if (estConsonne(prefixe[p.posLettre - 1])) {
                            char voyelle[6] = { 'a', 'e', 'i', 'o', 'u', 'y'
};
                            unsigned int index = rand() % 6;
                            c = voyelle[index];
                            break;
                        }
                        else {
                            char consonnes[21] = { 'b', 'c', 'd', 'f', 'g',
'h', 'j', 'k', 'l', 'm', 'n', 'p', 'q', 'r', 's', 't', 'v', 'w', 'x', 'y',
'z' };
                            unsigned int index = rand() % 21;
```

```
                            c = consonnes[index];
                            break;
                        }
                        break;
                    }

                }
                else {
                    if (nbTest > nbMots / 2) {
                        c = '?';
                        break;
                    }
                }

            } while (estMotValide(p, prefixe));

            cout << c << endl;
            ajoutLettre(p, c);
        }
        else {
            c = '?';
            cout << c << endl;
            ajoutLettre(p, c);
        }
        delete[] prefixe;
    }
}
/**
* @brief Génère une lettre aléatoire pour le robot
* @param[in,out] p: La partie en cours
*/
void saisiRobot(Partie& p) {
    char c = '\0';

    if (ptInterrogation(p)) {
        casPtInterroR(p);
    }
    else {
        casNormalSaisiR(p);
    }

}
/**
* @brief Fait jouer le joueur en cours
* @param[in,out] p: La partie en cours
*/
void jouer(Partie& p) {

    if (p.joueurs[p.tourActuel].type == 'H') {
        saisiHumain(p);
    }
    else {
        saisiRobot(p);
    }

}
/**
* @brief Affiche le mot tapé par le joueur courant dans la partie en cours
* @param[in] p: La partie en cours
*/
void motSaisi(Partie& p) {
    unsigned int longueur = strlen(p.motTap);
    for (unsigned int i = 0; i < longueur; ++i) {
        if (p.motTap[i] == 0) {
```

```
            break;
        }
        else {
            cout << p.motTap[i];
        }
    }
}
/**
* @brief Affiche l'état de la partie en cours
* @param[in,out] p: La partie en cours
*/
void afficher(Partie& p) {
    cout << p.tourActuel + 1 << p.joueurs[p.tourActuel].type << ", " << "(";
    motSaisi(p);
    cout << ")" << " > ";
}
/**
* @brief Fonction permettant de mettre à jour le score d'un joueur
* @param[in] p: La partie en cours
* @param[in] i: L'indice du joueur dont on veut mettre à jour le score
*/
void score(Partie& p, unsigned int i) {
    switch (p.joueurs[i].score) {
    case 1:
        cout << "0.25";
        break;
    case 2:
        cout << "0.5";
        break;
    case 3:
        cout << "0.75";
        break;
    case 4:
        cout << "1";
        break;
    default:
        cout << "0";
        break;
    }
}
/**
* @brief Affiche les scores des joueurs
* @param[in] p: La partie en cours
*/
void afficheScore(Partie& p) {
    for (unsigned int i = 0; i < p.nbJoueurs; ++i) {
        if (i == p.nbJoueurs - 1) {
            cout << i + 1 << p.joueurs[i].type << " : ";
            score(p, i);
            cout << endl;
        }
        else {
            cout << i + 1 << p.joueurs[i].type << " : ";
            score(p, i);
            cout << "; ";
        }
    }
}
/**
* @brief Ajoute un point au score du joueur en cours
* @param[in,out] p: La partie en cours
*/
void ajouteScore(Partie& p) {
    ++p.joueurs[p.tourActuel].score;
```

```
}
/**
* @brief Réinitialiser la manche en cours
* @param[in,out] p : La partie en cours
*/
void resetManche(Partie& p) {
    if (p.joueurs[p.tourActuel].score == 4);
    else {
        p.motTap[0] = '\0';
        p.motTapVerif[0] = '\0';
    }
    --p.tourActuel;
    p.posLettre = 0;
    return;
}
/**
* @brief Message si le mot tapé par le joueur existe dans le dictionnaire
* @param[in,out] p : La partie en cours
*/
void motExiste(Partie& p) {
    cout << "le mot "; motSaisi(p); cout << " existe, le joueur ";
    cout << p.tourActuel + 1 << p.joueurs[p.tourActuel].type << " prend un
quart de singe" << endl;
    ajouteScore(p);
    afficheScore(p);
    resetManche(p);

}
/**
* @brief Message si le mot tapé par le joueur n'est pas le même que le mot
tapé
* @param[in,out] p : La partie en cours
*/
void lettresDifferentes(Partie& p) {
    cout << "le mot " << p.motTapVerif << " ne commence pas par les lettres
attendues, le joueur " << p.tourActuel << p.joueurs[p.tourActuel - 1].type <<
" prend un quart de singe" << endl;
    --p.tourActuel; // Recule d'un tour donc d'un joueur
    ajouteScore(p);
    afficheScore(p);
    resetManche(p);
}
/**
* @brief Message si le mot tapé par le joueur existe
* @param[in,out] p : La partie en cours
*/
void motExisteVerif(Partie& p) {
    cout << "le mot " << p.motTapVerif << " existe, le joueur ";
    if (p.tourActuel == p.nbJoueurs) {
        p.tourActuel = 0;
        cout << p.tourActuel + 1 << p.joueurs[p.tourActuel].type << " prend
un quart de singe" << endl; // Cas particulier du dernier/premier joueur
    }
    else
        cout << p.tourActuel + 1 << p.joueurs[p.tourActuel].type << " prend
un quart de singe" << endl;
    ajouteScore(p);
    afficheScore(p);
    resetManche(p);
}
/**
* @brief Message si le mot tapé par le joueur n'existe pas
* @param[in,out] p : La partie en cours
*/
```

```
void motExistePas(Partie& p) {
    cout << "le mot " << p.motTapVerif << " n'existe pas, " << p.tourActuel
<< p.joueurs[p.tourActuel - 1].type << " prend un quart de singe" << endl;
    if (p.tourActuel == 0) {
        p.tourActuel = p.nbJoueurs;
    }
    else --p.tourActuel; // Recule d'un tour donc d'un joueur
    ajouteScore(p);
    afficheScore(p);
    resetManche(p);
}
/**
* @brief Saisir un mot pour verifier s'il existe
* @param[in,out] p : La partie en cours
*/
void saisirMotTapVerif(Partie& p) {

    if (p.joueurs[p.tourActuel - 1].type == 'H') {
        cin >> setw(MAX) >> p.motTapVerif;
        cin.ignore(INT_MAX, '\n');
        unsigned int longueur = strlen(p.motTapVerif);
        for (unsigned int i = 0; i < longueur; ++i) {
            p.motTapVerif[i] = toupper(p.motTapVerif[i]); // Majuscule
        }
    }
    else {
        saisiRobot(p);
    }

}
/**
* @brief Message si le joueur tape un point d'exclamation
* @param[in,out] p : La partie en cours
*/
void exclamation(Partie& p) {
    cout << "le joueur " << p.tourActuel + 1 << p.joueurs[p.tourActuel].type
<< " abandonne la manche et prend un quart de singe" << endl;
    ajouteScore(p);
    afficheScore(p);
    resetManche(p);
}
/**
* @brief Verifie si le caractère tapé est un point d'exclamation
* @param[in] p : La partie en cours
* @return Vrai si le dernier caractère est un point d'exclamation, faux sinon
*/
bool ptExclamation(Partie& p) {
    if (p.posLettre == 0) return false;
    if (p.motTap[p.posLettre - 1] == '!') return true;
    else return false;
}
/**
* @brief Verifie si les caractères tapés sont les mêmes que le mot tapé
* @param[in] p : La partie en cours
* @return retourne un booléen
*/
bool verifLettres(Partie& p) {
    unsigned int longueur = strlen(p.motTap);
    for (unsigned int i = 0; i < longueur -1; ++i) {
        if (p.motTapVerif[i] != p.motTap[i])
            return false;
    }
    return true;
}
```

```
/**
* @brief Cas où c'est un point d'interrogation, on fait les vérifications
* @param[in,out] p : La partie en cours
*/
void verifInterrogation(Partie& p) {
    if (p.tourActuel == 0) { // Cas particulier du dernier/premier joueur
        p.tourActuel = p.nbJoueurs;
    }
    cout << p.tourActuel << p.joueurs[p.tourActuel - 1].type << ", saisir le
mot > ";
    saisirMotTapVerif(p);

    if (!verifLettres(p)) {
        lettresDifferentes(p);
    }
    else if (!estDeuxLettre(p)) {
        if (estMotValide(p, p.motTapVerif)) {
            motExisteVerif(p);
        }
        else {
            motExistePas(p);
        }
    }
    else if (estPremiereLettre(p)) {
        if (estMotValide(p, p.motTapVerif)) {
            motExisteVerif(p);
        }
        else {
            motExistePas(p);
        }
    }
    else {
        motExistePas(p);
    }
}
/**
* @brief Cas où c'est un point d'exclamation, on fait les vérifications
* @param[in,out] p : La partie en cours
*/
void verifExclamation(Partie& p) {
    exclamation(p);
}
/**
* @brief Cas normal, on regarde si le mot existe
* @param[in,out] p : La partie en cours
*/
void verifNormal(Partie& p) {
    if (!estMotValide(p, p.motTap)) { // Vérification de la validité du mot
        return;
    }
    else { // Si le mot existe dans le dictionnaire
        motExiste(p);
        return;
    }
}
/**
* @brief Fonction reunissant toutes les verifications
* @param[in,out] p : La partie en cours
*/
void verification(Partie& p) {
    if (ptInterrogation(p)) { // Si le joueur a tapé ?
        if (estPremiereLettre(p)) {
            cout << "Aucun mot n'a été saisi, ";
```

```
            cout << p.tourActuel + 1 << p.joueurs[p.tourActuel].type << "
prend un quart de singe" << endl;
            ajouteScore(p);
            afficheScore(p);
            resetManche(p);
        }
        else {
            verifInterrogation(p); // Verification des cas possibles
        }

    }
    else if (ptExclamation(p)) { // Si le joueur a tapé !
        verifExclamation(p); // Verification des cas possibles
    }
    else {
        if (!estDeuxLettre(p)) {
            verifNormal(p); //Sinon verifications normales
        }
        else; //Si c'est la première lettre, on ne fait rien
    }
}
/**
* @brief Le jeu
* @param[in, out] p : La partie en cours
*/
void jouerPartie(Partie& p) {
    while (p.joueurs[p.tourActuel].score != 4) {

        afficher(p);
        jouer(p);
        verification(p);


        if (p.tourActuel == p.nbJoueurs - 1) p.tourActuel = 0;
        else ++p.tourActuel;
    }
    cout << "La partie est finie" << endl;
}
/**
* @brief Détruit une partie et libère les ressources associées
* @param[in,out] p: La partie à détruire
*/
void detruirePartie(Partie& p) {
    delete[] p.joueurs;
    p.joueurs = nullptr;
    delete[] p.motTap;
    p.motTap = nullptr;

    for (unsigned int i = 0; i < p.d.nbMot; ++i) {
        delete[] p.d.mots[i];
    }
    delete[] p.d.mots;

    p.d.nbMot = NULL;
    p.d.mots = nullptr;

    delete[] p.motTapVerif;
    p.motTapVerif = nullptr;
}
```