Leveque Julia et Martins Paulo Classe 108, groupe 12 sur Moodle

Dossier – Comparaison d'approches algorithmiques – Projet (S1.02)

Ce dossier présente une application du jeu Quart de Singe, codée en C++.

IUT de Paris - Rives de Seine
Université Paris Cité

2022-2023

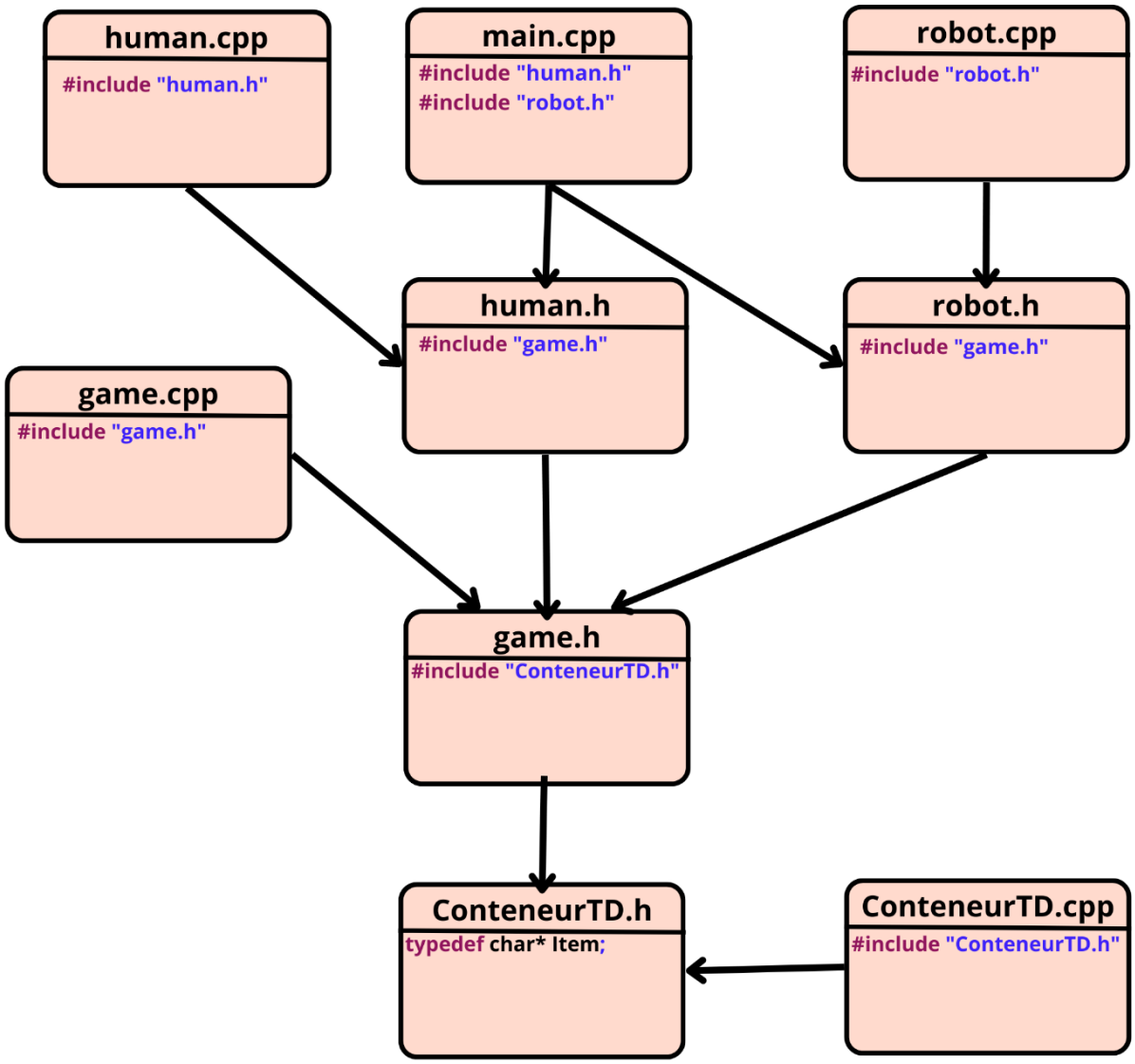
Tables des matières:

Introduction

Cette application codée en C++ est un logiciel permettant à un ensemble de joueurs, autant humains que robots, de disputer une partie de quart de singe.

Elle permet de gérer la totalité du déroulement d'une partie selon les règles du jeu jusqu'à l'annonce du perdant.

Graphe de dépendance



Code source des tests unitaires

Plusieurs tests ont été effectués tout le long du développement de l'application.

La plupart sont directement dans le code. En effet, nous avons utilisé des asserts dans nos fonctions pour pouvoir les sécuriser et permettre une meilleure utilisation de celle-ci.

Enfin, nous avons créé un fichier C++ pour pouvoir tester notre ConteneurTD et notamment l'allocation et libération des données en mémoire:

```
#include <iostream>
#include <assert.h>
#include "ConteneurTD.h"
#pragma warning (disable:4996)

using namespace std;

/* Test d'un conteneur d'éléments de type Item */
int main() {
    ConteneurTD c; // Déclaration du conteneur d'Items
    char letter[26];

    // Initialisation de la pile
    initialiser(c, 2);
    cout << "Test de conteneur d'Items" << endl;

    assert(c.capacite == 2);
    assert(c.tab != nullptr);

    // Test fonction ecrire
    strcpy(letter, "test1");
    // Ajout des Items dans le conteneur
    ecrire(c, 0, letter);
    strcpy(letter, "test2");
    ecrire(c, 1, letter);
    // Verifie si l'item ajouté est bien le meme au meme indice correspondant
    assert(strcmp(lire(c, 0), "test1") == 0);
    assert(strcmp(lire(c, 1), "test2") == 0);

    // Test fonction detruire
    detruire(c);
    assert(c.tab == nullptr);
    return 0;
}
```

Bilan de projet

Ce projet s'est plutôt bien passé, bien que plusieurs difficultés aient été rencontrées.

En effet, le développement d'une partie de quart de singe entre humains s'est faite assez rapidement.

Nous avons tout d'abord développé cette application en parcourant le dictionnaire à chaque entrée de lettres, sans le stocker. Cette application, bien que fonctionnelle, s'est révélée ne pas être très optimisée. En effet, nous devions à chaque lettre écrite comparer le nouveau mot avec ceux du dictionnaire. Notre algorithme avait donc une complexité élevée.

De ce fait, nous avons décidé de réessayer en enregistrant notre dictionnaire dans un ConteneurTD afin de manipuler toutes ces données de manière efficace. Cela nous a également permis d'implémenter un algorithme dichotomique ce qui a rendu l'application beaucoup plus performante.

Aussi, le jeu de test déposé sur Moodle fonctionne avec ces deux programmes.

Le reste de l'application a été développé rapidement et les robots sont fonctionnels.

De plus, nous avons essayé d'améliorer la lisibilité du code grâce à des commentaires, de la documentation pour chaque fonction et nous avons réduit toutes les lignes en dessous de 120 caractères et sous 80 caractères quand cela était possible.

Afin d'améliorer cet application nous pouvons améliorer les robots afin qu'ils jouent d'une meilleure manière et moins répétitive (Face à la même chaîne de caractère dans deux contextes différents, le robot répondra la même lettre, ce qui n'est pas très intéressant.)

Enfin, nous pouvons préciser que l'application n'a pas de fuite mémoire. Toutes les variables sont correctement allouées et libérées ce qui la rend moins vulnérable aux erreurs et pertes.

Annexes

Listing du code source

```
#ifndef _CONTENEURTD_
#define _CONTENEURTD_

/**
 * @file ConteneurTD.h
 * @brief Composant de conteneur d'items en mémoire dynamique à capacité fixe.
```

```
 */

// Spécialisation d'Item en pointeur de char
typedef char* Item;

/** @brief Type des conteneurs d'itemes alloués en mémoire dynamique
 *  et de capacité fixe. */
struct ConteneurTD {
        /// Capacite du conteneur.
        unsigned int capacite;
        /// Tableau d'items alloué en mémoire dynamique.
        Item* tab;
};

/**
 * @brief Initialise un conteneur d'items vide (allocation dynamique de mémoire - @see detruire pour sa
 * d'allocation en fin d'utilisation).
 * @param[out] c Conteneur d'items.
 * @param [in] capa Capacite du conteneur.
 * @pre capa > 0
 */
void initialiser(ConteneurTD& c, unsigned int capa);

/**
 * @brief Désalloue un conteneur d'items en mémoire dynamique.
 * @param[out] c Conteneur d'items.
 */
void detruire(ConteneurTD& c);

/**
 * @brief Lecture d'un item d'un conteneur.
 * @param[in] c Conteneur d'items.
 * @param[in] i Indice de l'item ' lire.
 * @return L'item se trouvant ' la position i dans le conteneur c.
 * @pre i < c.capacite.
 */
 Item lire(const ConteneurTD& c, unsigned int i);

/**
 * @brief Ecrire un item dans un conteneur d'items.
 * @param[out] c Conteneur d'items.
 * @param[in] i Position o' ajouter/modifier l'item.
 * @param[in] item Item ' 'crire.
 * @pre i < c.capacite.
 */
void ecrire(ConteneurTD& t, unsigned int i, const Item& it);

#endif
```

```
/**
 * @file ConteneurTD.cpp
 * @brief Composant de conteneur d'items en mémoire dynamique a capacite fixe.
 */

#include <iostream>
#include <assert.h>
#include <string.h>
```

```cpp
#include "ConteneurTD.h"
#pragma warning(disable : 4996);

using namespace std;

void initialiser(ConteneurTD& c, unsigned int capa) {
        assert(capa>0);
        c.capacite = capa;
        c.tab = new Item[capa];
}

void detruire(ConteneurTD& c) {
        for(unsigned int i = 0; i < c.capacite - 1; ++i){
                delete [] c.tab[i];
                c.tab[i] = NULL;
        }
        delete [] c.tab;
        c.tab = NULL;
}

 Item lire(const ConteneurTD& c, unsigned int i) {
        assert(i < c.capacite);
        return c.tab[i];
}

 void ecrire(ConteneurTD& c, unsigned int i, const Item& it) {
         assert(i < c.capacite);
         Item chaine = new char[26];
         strcpy(chaine, it);
         c.tab[i] = chaine;
 }
```

```
#ifndef _GAME_
#define _GAME_

/**
 * @file game.h
 * @brief Fichier contenant les fonctions de gestion d'une partie.
 */


#include <assert.h>
#include <iostream>
#include <string.h>
#include "ConteneurTD.h"
#pragma warning(disable: 4996)

using namespace std;

enum {
    MIN_PLAYERS = 2,
    MIN_SCORE = 0,
    MAX_SCORE = 4,
    MAX_LETTER = 26,
    MIN_LETTER = 2,
    MIN_ASCII = 65,
    MAX_ASCII = 90,
};

struct Player {
    unsigned int score; // Score d'un joueur
    char type; // Type d'une joueur (humain ou robot)
};

struct Game {
    unsigned int nbPlayers; // Nombre de joueurs d'une partie
    unsigned int lastPlayer; // Dernier joueur ayant donné une lettre
    Player* players; // Tableau des joueurs
};

/**
 * @brief Affiche le joueur qui joue actuellement et le mot actuellement forme.
 * @param[in] g La structure qui contient les informations sur la partie en cours.
 * @param[in] word Le mot actuellement forme.
 * @param[in] len La longueur du mot.
 * @pre g.nbPlayers > 0
 */
void showWord(const Game& g, const Item word, unsigned int len);

/**
 * @brief Cherche un mot dans un dictionnaire.
 * @param[in] dico Le dictionnaire dans lequel chercher.
 * @param[in] word Le mot à chercher dans le dictionnaire.
 * @pre dico.capacite > 0
 * @return true si le mot est trouvé dans le dictionnaire
 et a une longueur supérieure à MIN_LETTER, false sinon.
 */
bool findWord(const ConteneurTD& dico, const Item word);

/**
 * @brief Ajoute une lettre a un mot.
```

```
 * @param[in, out] word Le mot auquel ajouter la lettre.
 * @param[in] ind L'index de la lettre a ajouter dans le mot.
 * @param[in] letter La lettre a ajouter au mot.
 * @return Le mot avec la lettre ajouté.
 */
void addLetter(Item& word, unsigned int ind, const char letter);

/**
* @brief Détruit un mot et libère la mémoire allouée.
* @param[in, out] it l'Item à détruire.
*/
void destroyWord(Item& it);

#endif
```

```
/**
 * @file game.h
 * @brief Fichier contenant les fonctions de gestion d'une partie.
 */

#include "game.h"

void showWord(const Game& g, const Item word, unsigned int len) {
    assert(g.nbPlayers > 0);
    cout << g.lastPlayer + 1 << g.players[g.lastPlayer].type << ", (";
    for (unsigned int i = 0; i < len; ++i) {
        cout << word[i];
    }
    cout << ") > ";
}


bool findWord(const ConteneurTD& dico, const Item word) {
    assert(dico.capacite > 0);
    unsigned int start = 0, end = dico.capacite, mid = ((start + end) / 2);
    bool found = false;
    while (not found and start <= end) {
        if (strcmp(dico.tab[mid], word) == 0
            && strlen(dico.tab[mid]) > MIN_LETTER) {
            found = true;
            return found;
        }
        else {
            if (strcmp(dico.tab[mid], word) <= 0) {
                start = mid + 1;
            }
            else if (strcmp(dico.tab[mid], word) > 0) {
                end = mid - 1;
            }
        }
        mid = ((start + end) / 2);
        if (mid == 0) {
            return (strcmp(dico.tab[mid], word) == 0
                && strlen(dico.tab[mid]) > MIN_LETTER);
        }
    }
    return found;
```

```
}

void addLetter(Item& word, unsigned int ind, const char letter) {
    if (ind >= strlen(word)) {
        unsigned int newTaille = strlen(word) + 2;
        Item newWord = new char[newTaille];
        for (unsigned int j = 0; j < strlen(word); ++j) {
            newWord[j] = word[j];
        }
        delete[] word;
        word = newWord;
    }

    word[ind] = toupper(letter);
    word[ind + 1] = '\0';
}

void destroyWord(Item& it) {
    delete[] it;
    it = NULL;
}
```

```
#ifndef _HUMAN_
#define _HUMAN_

#include "game.h"

/**
 * @file human.h
 * @brief Fichier contenant les fonctions de gestion des joueurs humains.
 */

/**
 * @brief Initialise une partie de pendu, les joueurs sont alloués dynamiquement.
 * @param[in, out] g La structure à initialiser.
 * @param[in] players Le nombre de joueurs dans la partie.
 * @pre players >= MIN_PLAYERS
 */
void init(Game& g, unsigned int players);

/**
 * @brief Détruit une partie et libère la mémoire allouée pour les joueurs.
 * @param[in, out] g La partie à détruire.
 */
void destroyGame(Game& g);

/**
 * @brief Lit les informations d'un joueur dans une partie.
 * @param[in] g La structure qui contient les informations sur la partie en cours.
 * @param[in] i L'index du joueur à lire.
 * @pre i < g.nbPlayers
 * @return Les informations du joueur à l'index `i`.
 */
Player read(const Game& g, unsigned int i);

/**
 * @brief Ajoute un joueur à une partie. Allocation dynamique si on dépasse le tableau.
```

```
 * @param[in, out] g La structure qui contient les informations sur la partie.
 * @param[in] i L'index du joueur à ajouter.
 * @param[in] type Le type de joueur (humain ou robot).
 */
void addPlayer(Game& g, unsigned int i, const char type);

/**
 * @brief Affiche les scores des joueurs d'une partie.
 * @param[in] g La structure qui contient les informations sur la partie en cours.
 */
void showScore(const Game& g);

/**
 * @brief Demande le mot auquel pense un joueur d'une partie de quart de singe.
 * @param[in] g La structure qui contient les informations sur la partie en cours.
 * @param[in] dico Le dictionnaire contenant les mots.
 * @param[in] word Le mot en construction.
 * @param[in] before L'index du joueur (précédent) qui a demandé le mot.
 * @pre before >= 0 && before < g.nbPlayers
 * @return L'index du joueur qui a pris un quart de singe
 (`before` si le mot n'est pas valide, sinon l'index du joueur qui a donné le mot).
 */
int askWord(const Game& g, const ConteneurTD& dico, Item word, unsigned int before);

#endif
```

```
/**
 * @file human.h
 * @brief Fichier contenant les fonctions de gestion des joueurs humains.
 */

#include <iomanip>
#include <limits.h>
#include "human.h"

void init(Game& g, unsigned int players) {
    assert(players >= MIN_PLAYERS);
    g.nbPlayers = players;
    g.players = new Player[players];
    g.lastPlayer = 0;
}

void destroyGame(Game& g) {
    delete[] g.players;
    g.players = NULL;
    g.nbPlayers = 0;
    g.lastPlayer = 0;
}

Player read(const Game& g, unsigned int i) {
    assert(i < g.nbPlayers);
    return g.players[i];
}

void addPlayer(Game& g, unsigned int i, const char type) {
    Player pl;
    pl.type = type;
    pl.score = MIN_SCORE;
```

```
    if (i >= g.nbPlayers) {
        unsigned int newTaille = i + 1;
        Player* newT = new(nothrow) Player[newTaille];
        for (unsigned int j = 0; j < g.nbPlayers; ++j) {
            newT[i] = read(g, j);
        }
        delete[] g.players;
        g.players = newT;
        g.nbPlayers = newTaille;
    }
    g.players[i] = pl;
}

void showScore(const Game& g) {
    float res = 0.;
    for (int i = 0; i < g.nbPlayers; ++i) {
        res = float(g.players[i].score) / float(MAX_SCORE);
        cout << i + 1 << g.players[i].type << " : "
            << res;
        if (i != g.nbPlayers - 1) {
            cout << "; ";
        }
    }
    cout << endl;
}

int askWord(const Game& g, const ConteneurTD& dico, Item word, unsigned int before) {
    assert(before >= 0 && before < g.nbPlayers);
    char answer[MAX_LETTER];
    cout << before + 1 << g.players[before].type << ", saisir le mot > ";
    cin >> setw(MAX_LETTER) >> answer;
    cin.ignore(INT_MAX, '\n');

    for (unsigned int i = 0; i < strlen(answer); ++i) {
        answer[i] = toupper(answer[i]);
    }

    for (unsigned int j = 0; j < strlen(word); ++j) {
        if (answer[j] != word[j]) {
            cout << "le mot " << answer
                << " ne commence pas par les lettres attendues, le joueur "
                << before + 1 << g.players[before].type
                << " prend un quart de singe" << endl;
            return before;
        }
    }

    if (findWord(dico, answer)) {
        cout << "le mot " << answer << " existe, le joueur "
            << g.lastPlayer + 1 << g.players[g.lastPlayer].type
            << " prend un quart de singe" << endl;
        return g.lastPlayer;
    }

    cout << "le mot " << answer << " n'existe pas, le joueur " << before + 1
        << g.players[before].type << " prend un quart de singe" << endl;
    return before;
}
```

```
#ifndef _ROBOT_
#define _ROBOT_

#include "game.h"

/**
 * @file robot.h
 * @brief Fichier contenant les fonctions de gestion des joueurs robots.
 */

/**
 * @brief Compare une partie d'une chaîne de caractère avec la totalité d'une autre
 * @param[in] dico, un Item dont on comparera une partie.
 * @param[in] word, un Item le mot recherché au début de l'Item dico.
 * @pre dico != NULL && word != NULL
 * @return true si word est égal au début de dico, false sinon
 */
bool strcmpHalf(const Item dico, const Item word);

/**
 * @brief Cherche un mot dans un dictionnaire et renvoie la prochaine lettre du mot.
 * @param[in] dico Le dictionnaire dans lequel chercher.
 * @param[in] word Le mot à chercher dans le dictionnaire.
 * @pre dico.capacite > 0
 * @return La prochaine lettre du mot suivant si le mot est trouvé, '0' sinon.
 */
char closeWord(const ConteneurTD& dico, const Item word);

/**
 * @brief Permet à un robot de jouer en proposant une lettre.
 * @param[in] dico Le dictionnaire à utiliser pour trouver des mots.
 * @param[in] word Le mot actuellement formé par les adversaires.
 * @return La lettre proposée par le robot. '?' si aucun mot du dico ne commence par `word`.
 */
char robotPlay(const ConteneurTD& dico, const Item word);

/**
 * @brief Cherche le début d'un mot dans un dictionnaire et le renvoie en entier.
 * @param[in] dico Le dictionnaire dans lequel chercher.
 * @param[in] word Le mot à chercher dans le dictionnaire.
 * @pre dico.capacite > 0
 * @return Un mot commençant par `word` si trouvé, `word`sinon.
 */
Item getEnd(const ConteneurTD& dico, const Item word);

/**
 * @brief Demande au robot de donner le mot auquel il pense.
 * @param[in] g La structure qui contient les informations sur la partie en cours.
 * @param[in] dico Le dictionnaire à utiliser pour trouver des mots.
 * @param[in] word Le mot à compléter.
 * @return true si un mot du dico commence par `word`, false sinon.
 * @warning Si un mot est trouvé on l'affichera
 ainsi qu'un message indiquant que le joueur précédent prend un quart de singe.
 */
bool askRobot(const Game& g, const ConteneurTD& dico, const Item word);

#endif
```

```cpp
/**
 * @file robot.h
 * @brief Fichier contenant les fonctions de gestion des joueurs robots.
 */

#include "robot.h"

bool strcmpHalf(const Item dico, const Item word) {
    assert(dico != NULL && word != NULL);
    unsigned int lenDico = strlen(dico), lenWord = strlen(word);
    if (lenDico <= MIN_LETTER || lenDico <= lenWord) {
        return false;
    }
    for (unsigned int i = 0; i < strlen(word); ++i) {
        if (dico[i] != word[i]) {
            return false;
        }
    }
    return true;
}

char closeWord(const ConteneurTD& dico, const Item word) {
    assert(dico.capacite > 0);
    unsigned int start = 0, end = dico.capacite, mid = ((start + end) / 2);
    while (start <= end) {
        if (strcmpHalf(dico.tab[mid], word)) {
            return dico.tab[mid][strlen(word)];
        }
        else {
            if (strcmp(dico.tab[mid], word) <= 0) {
                start = mid + 1;
            }
            else if (strcmp(dico.tab[mid], word) > 0) {
                end = mid - 1;
            }
        }
        mid = ((start + end) / 2);
    }
    return '0';
}

char robotPlay(const ConteneurTD& dico, const Item word) {
    char letter = { 0 };

    // Si mot vide, lettre au hasard
    if (word == 0) {
        srand(time(NULL));
        letter = char(rand() % (MAX_ASCII - MIN_ASCII + 1) + MIN_ASCII);
    }
    else {
        letter = closeWord(dico, word);
        // Si il n'y a pas de mot commençant par cette lettre
        if (letter == '0') {
            letter = '?';
        }
    }
    return letter;
}
```

```
Item getEnd(const ConteneurTD& dico, const Item word) {
    assert(dico.capacite > 0);
    unsigned int start = 0, end = dico.capacite, mid = ((start + end) / 2);
    while (start <= end) {
        if (strcmpHalf(dico.tab[mid], word)) {
            return dico.tab[mid];
        }
        else {
            if (strcmp(dico.tab[mid], word) <= 0) {
                start = mid + 1;
            }
            else if (strcmp(dico.tab[mid], word) > 0) {
                end = mid - 1;
            }
        }
        mid = ((start + end) / 2);
    }
    return word;
}

bool askRobot(const Game& g, const ConteneurTD& dico, const Item word) {
    Item answer = getEnd(dico, word);
    cout << answer << endl;
    if (answer == word) {
        return false;
    }
    cout << "le mot " << answer << " existe, le joueur "
        << g.lastPlayer + 1 << g.players[g.lastPlayer].type
        << " prend un quart de singe" << endl;
    return true;
}
```

```
/**
 * @file main.cpp
 * @brief Fichier principal d'un jeu de quart de singe
 * @author Julia Leveque (108) et Paulo Martins (108)
 * @version 06/01/2023
 */

#include <fstream>
#include <iomanip>
#include <limits.h>
#include "human.h"
#include "robot.h"

using namespace std;

/**
 * @brief Joue une manche de quart de singe.
 * @param[in, out] g La structure qui contient les informations sur la partie.
 * @param[in] dico Le dictionnaire contenant les mots.
 * @pre g.nbPlayers >= MIN_PLAYERS : au moins deux joueurs
 * @return L'index du joueur qui a pris un quart de singe.
 */
unsigned int play(Game& g, const ConteneurTD& dico) {
    assert(g.nbPlayers >= MIN_PLAYERS);
    unsigned int ind = 0, before = (g.lastPlayer - 1) % g.nbPlayers;
    bool perdu = false;
```

```
    char letter = { 0 };
    Item word = { 0 };
    word = new char[MIN_LETTER];
    word[0] = '\0';
    do {
        // Affichage du joueur, son type et le mot déjà formé
        showWord(g, word, ind);

        if (g.players[g.lastPlayer].type == 'H') {
            // Vérifie que la lettre est valide (lettre ou '!' ou '?')
            do {
                cin >> letter;
                cin.ignore(INT_MAX, '\n');
            } while (letter != '!' && letter != '?'
                && not isalpha((unsigned char)letter)
                || (letter == '?' && ind == 0)); // Pas de '?' au premier tour
        }
        else {
            // Permet à un joueur robot de donner une lettre
            letter = robotPlay(dico, word);
            cout << letter << endl;
        }

        if (letter == '!') {
            cout << "le joueur " << g.lastPlayer + 1
                << g.players[g.lastPlayer].type
                << " abandonne la manche et prend un quart de singe" << endl;
            perdu = true;
            break;
        }
        else if (letter == '?') {

            if (g.players[before].type == 'H') {
                // On demande au joueur humain d'entrer son mot
                g.lastPlayer = askWord(g, dico, word, before);
            }
            else {
                // On demande au joueur robot d'entrer son mot
                cout << before + 1 << g.players[before].type
                    << ", saisir le mot > ";
                if (not askRobot(g, dico, word)) {
                    cout << "le mot " << word << " n'existe pas, le joueur "
                        << before + 1 << g.players[before].type
                        << " prend un quart de singe" << endl;
                    g.lastPlayer = before;
                    perdu = true;
                    break;
                }
            }
            perdu = true;
            break;
        }

        addLetter(word, ind, letter);

        // Si le mot est complet
        if (findWord(dico, word)) {
            cout << "le mot " << word << " existe, le joueur "
                << g.lastPlayer + 1 << g.players[g.lastPlayer].type
```

```
                << " prend un quart de singe" << endl;
            perdu = true;
            break;
        }

        // On determine le prochain joueur a jouer
        g.lastPlayer = (g.lastPlayer + 1) % g.nbPlayers;
        before = g.lastPlayer == 0 ? g.nbPlayers - 1
            : (g.lastPlayer - 1) % g.nbPlayers;
        ++ind;

    } while (not perdu);

    destroyWord(word);
    return g.lastPlayer;
}

/**
 * @brief Récupère le nombre de lignes dans un fichier `file`.
 * @param[in] file le nom du fichier
 * @pre Le fichier doit exister et être accessible en lecture.
 * @return Le nombre de lignes dans le fichier.
 */
int getLenght(const char* file) {
    ifstream in(file);
    assert(in);

    unsigned int len = 1;
    char line[MAX_LETTER];

    in >> setw(MAX_LETTER) >> line;
    while (in) {
        ++len;
        in >> setw(MAX_LETTER) >> line;
    }

    in.close();
    return len;
}

/**
 * @brief Initialise le dictionnaire avec les mots contenus dans un fichier.
 * @param[in, out] dico Le dictionnaire a initialiser.
 * @pre Le fichier doit exister et être accessible en lecture.
 * @pre Le dictionnaire doit être initialisé (alloue et vide).
 */
void initDico(ConteneurTD& dico) {
    ifstream in("ods4.txt");
    assert(in);

    int ind = 0;
    char line[MAX_LETTER];

    in >> setw(MAX_LETTER) >> line;
    while (in) {
        ecrire(dico, ind, line);
        ++ind;
        in >> setw(MAX_LETTER) >> line;
    }
```

```
    in.close();
}

/**
 * @brief Fonction principale du programme
 * @param[in] argc Nombre d'arguments passés au programme.
 * @param[in, out] argv Tableau de chaînes de caractères contenant les arguments.
 * @return 0 si la partie s'est déroulée avec succès, 2 sinon.
 */
int main(int argc, char* argv[]) {
    if (argc >= 2) {
        Game g;
        ConteneurTD dico;
        unsigned int countPlayer = 0, dicoLenght = getLenght("ods4.txt");

        // Vérification des types de joueurs
        for (unsigned int i = 0; i < strlen(argv[1]); ++i) {
            argv[1][i] = toupper(argv[1][i]);
            if (argv[1][i] == 'H' || argv[1][i] == 'R') {
                countPlayer += 1;
            }
        }

        if (countPlayer < MIN_PLAYERS) {
            cout << "Il faut au moins 2 joueurs pour lancer une partie."
                << endl;
            return 2;
        }

        init(g, countPlayer);
        initialiser(dico, dicoLenght);
        initDico(dico);
        //Ajout des joueurs dans la partie
        for (unsigned int j = 0; j < strlen(argv[1]); ++j) {
            if (argv[1][j] == 'H' || argv[1][j] == 'R') {
                addPlayer(g, j, argv[1][j]);
            }
        }
        // Lancement d'une manche
        int loser = g.lastPlayer;
        do {
            loser = play(g, dico);
            g.players[loser].score += 1;
            showScore(g);
        } while (g.players[loser].score < MAX_SCORE);

        detruire(dico);
        destroyGame(g);
        cout << "La partie est finie" << endl;
        return 0;
    }
    else{
        cout << "Il faut au moins 2 joueurs pour lancer une partie." << endl;
        return 2;
    }
}
```