Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet de programmation et génie logiciel

2017-2018

Rapport de projet de programmation et génie logiciel

Développement d'un sous-projet de tests fonctionnels

Tuteur académique
Nicolas Ragot

Étudiants
Xavier Bouchenard (DI4)
Raphaël Lazzaroni (DI4)
Abdoulaye Kake (DI4)

8 Janvier 2018

Liste des intervenants

Nom	Email	Qualité
Xavier Bouchenard	xavier.bouchenard@etu.univ-tours.fr	Étudiant DI4
Raphaël Lazzaroni	raphael.lazzaroni@etu.univ-tours.fr	Étudiant DI4
Abdoulaye Kake	abdoulaye.kake@etu.univ-tours.fr	Étudiant DI4
Nicolas Ragot	nicolas.ragot@univ-tours.fr	Tuteur académique, Département Informatique

Avertissement

Ce document a été rédigé par Xavier Bouchenard, Raphaël Lazzaroni et Abdoulaye Kake susnommés les auteurs.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Nicolas Ragot susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document

Xavier Bouchenard, Raphaël Lazzaroni et Abdoulaye Kake, *Rapport de projet de programmation et génie logiciel: Développement d'un sous-projet de tests fonctionnels*, Projet de programmation et génie logiciel, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
    author={Bouchenard, Xavier and Lazzaroni, Raphaël and Kake, Abdoulaye},
    title={Rapport de projet de programmation et génie logiciel: Développement d'un sous-
        projet de tests fonctionnels},
    type={Projet de programmation et génie logiciel},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2017-2018}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document polytech.cls V2.2.5 (2017/05/02)

Table des matières

Table des figures

1 Introduction

L'application sur laquelle nous travaillons est une boîte à outils pour la comparaison de séquences qui peut être utilisée pour faire de l'analyse d'images et toutes sortes de documents. Elle permet d'analyser et de comparer des séquences comme des chaînes de caractères, des vecteurs, des numériques,...

Ce projet a été développé de sorte à ce que l'application puisse être exécutée sous plusieurs environnements différents comme Windows ou Linux par exemple. Ainsi, l'objectif de ce projet est d'être diffusé en open-source sur des plate-formes de partage de codes comme Git. Jusqu'au commencement de notre projet, trois sous-projets composent le projet global :

1. MatchingLibrary
   Contient toutes les classes de base comme l'implémentation des algorithmes
2. MatchingToolBox
   Contient tous les éléments permettant de lancement et l'utilisation des algorithmes de recherche grâce à des paramètres passés et l'affichage des résultats en mode console
3. MatchingToolBoxTest
   Contient les classes de tests unitaires effectués sur le sous-projet précédent pour la vérification du fonctionnement, que ce soit les algorithmes ou la lecture des fichiers

Aujourd'hui, le but de notre projet s'inscrit dans la thématique du génie logiciel. Il s'agit d'intégrer un nouveau sous-projet destiné à implémenter une classe de test par algorithme présent dans le projet principal **MatchingLibrary** et de tester son fonctionnement. Nous avons donc implémenté ces tests en tenant compte de l'environnement dans lequel le projet sera exécuté.

A la suite de ça, nous avons fait en sorte à ce que le projet soit le plus optimisé possible en essayant de trouver des erreurs d'implémentation,...

Nous allons voir dans une première partie, les spécifications auxquelles doivent correspondre notre projet. Ensuite, nous verrons la façon dont nous avons décidé de modéliser ces tests. Nous verrons après le choix de l'architecture de tests appliquée. Nous finirons sur une dernière partie dédiée aux corrections effectuées.

2 Spécifications du projet

Comme nous l'avons vu précédemment, la boîte à outils de comparaison de séquences doit pouvoir être exécutée sur divers environnements. Ainsi, les tests que nous devons implémentés doivent donc suivre la même logique. Le diagramme des cas d'utilisation ci-dessous résume le fonctionnement du projet.

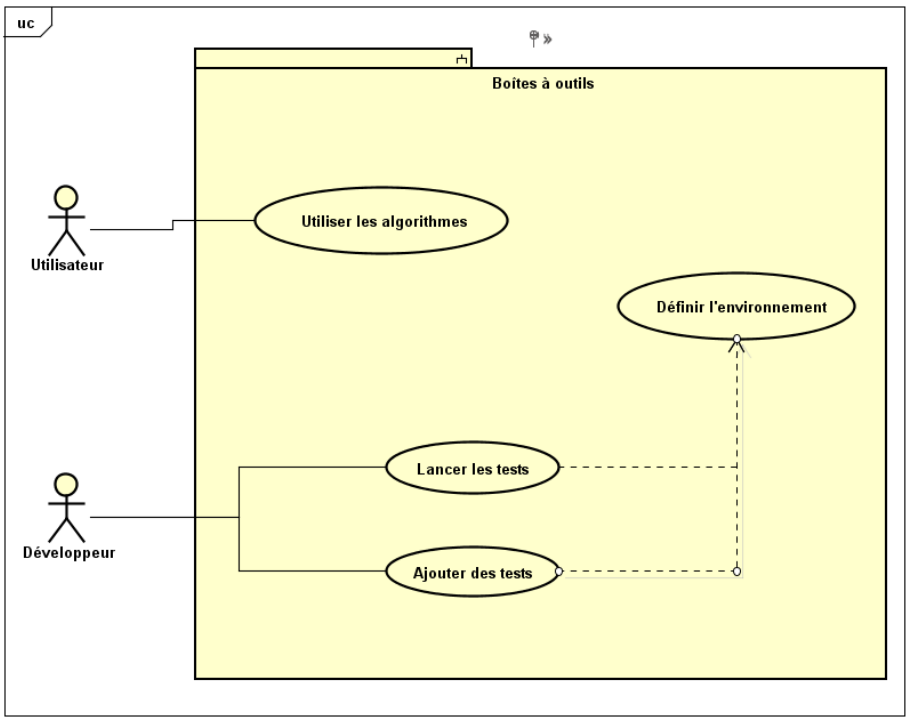

Figure 1 – *Diagramme des cas d'utilisations du projet*

Dans notre cas, l'utilisateur va pouvoir exécuter et utiliser la boîte à outils sans se soucier de l'environnement à partir duquel il l'exécute. Quant au développeur, il doit tenir compte de cette caractéristique s'il souhaite ajouter une classe de test supplémentaire. Nous verrons les détails de l'implémentation après.

3 Modélisation du projet

Nous avons structuré notre code en plusieurs classes de tests fonctionnels, chaque classe de tests étant spécifique à un algorithme. Pour montrer l'architecture de notre projet, voici le diagramme UML associé.

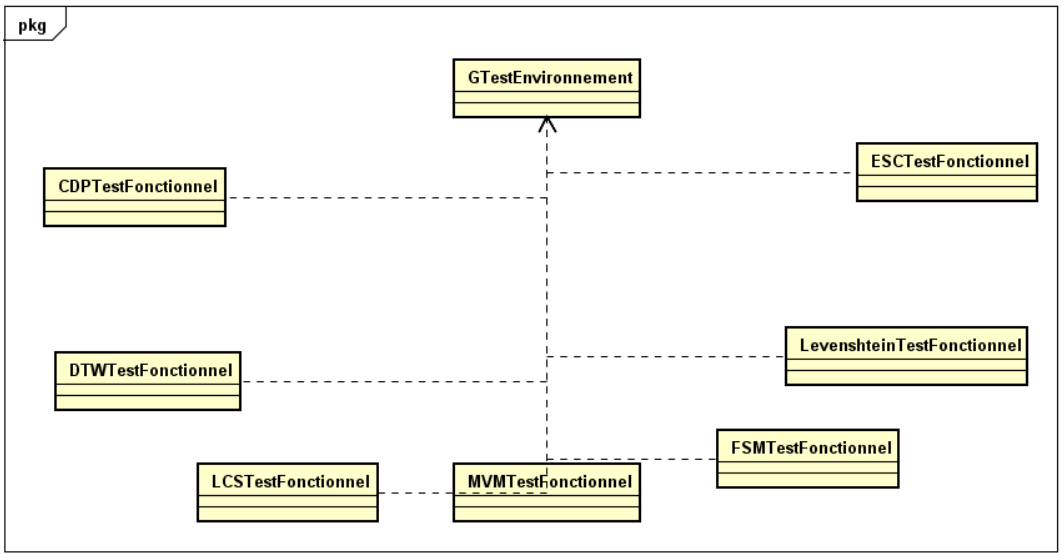

Figure 1 – *Représentation de l'architecture du projet*

Pour se faire, nous nous sommes appuyés sur le **framework Google Test**, qui avait déjà été inclus et utilisé pour le développement des tests unitaires. Nous l'avons utilisé en raison de sa simplicité de mise en place et d'utilisation.
Chaque classe de tests contient 9 méthodes de tests pour chaque type de donnée que l'on peut comparer et le type de fichier que l'on peut lire ou dans lequel on va enregistrer les résultats des différents algorithmes de recherche appliqués. Dans chacune des classes, les 3 premiers tests utilisent également un fichier de paramètres, un par type d'élément. Cela permet de tester tous les cas d'utilisation qui sont censés fonctionner.
Pour ne pas devoir implémenter le code permettant de vérifier le résultat retourné après exécution avec le résultat qu'on devrait avoir, nous avons choisi de mettre en place une nouvelle classe **ReadFile** qui sera utilisé à chaque test.

4 Choix de l'implémentation

Pour mettre en place nos tests, chaque méthode de test utilise la classe **ReadFile**. Cette classe a été développée afin de comparer chaque résultat retourné par la librairie avec un résultat attendu. La classe comporte la méthode **compareFile** qui retourne vrai si les deux fichiers sont considérés comme égaux en comparant les deux chaînes des fichiers de manière générique. Il n'y a aucune structure avec balise à respecter. Ce choix a été fait afin que le fonctionnement de cette classe ne soit obsolète après un changement de la structure des fichiers de sortie. Dans ce cas, seuls les fichiers correspondant aux résultats attendus devront être changés.

Deux fichiers peuvent ne pas avoir parfaitement leur contenus identiques et correspondre. La méthode **stringWhithoutCharac** appelée dans **compareFile** permet de supprimer des caractères dans une chaîne tel que les espaces et les sauts de lignes. Nous n'avons pas trouvé de solution efficace pour que les nombres du résultat suite à un arrondissement soient considérés comme identique tout en restant générique.

Dans la classe **GTestEnvironnement**, nous allons retrouver des variables préprocesseurs qui vont prendre des valeurs en fonction de l'environnement sur lequel le programme est en fonctionnement. En fonction de l'état de la valeur de la variable d'environnement, des chemins relatifs ont été déclarés permettant à chaque classe de tests d'être affecté à un répertoire unique.

Ce principe permet de rendre le fonctionnement des algorithmes indépendants des uns des autres car chaque ressource spécifique à un algorithme est directement stocké dans le répertoire lui étant associé. Cela permet également de pouvoir changer très facilement l'emplacement des fichiers en modifiant uniquement le contenu d'une seule variable.

Les classes de la librairie que l'on nous a fourni étaient partiellement programmées pour fonctionner sur plusieurs environnements. Il y avait des parties de code dans certaines classes permettant cela, mais il y avait beaucoup de variables pré-processeurs spécifique au logiciel Visual Studio. Nous avons effectué quelques ajustements précisés dans la partie optimisation du programme global afin d'avoir une librairie facilement adaptable aux différents environnements. De plus, cela permet par la suite, pour les tests unitaires et fonctionnels, de faire un test d'environnement.

```
#ifndef GTESTENVIRONMENT_H
#define GTESTENVIRONMENT_H

#include "Readfile.h"
#include "Command.h"
#if defined _WIN32 || defined _WIN64
    #include "gtest\gtest.h"

    #define _p "\\"
    #define _shared_path_CDP "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\CDP\\"
    #define _shared_path_DTW "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\DTW\\"
    #define _shared_path_ESC "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\ESC\\"
    #define _shared_path_FSM "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\FSM\\"
    #define _shared_path_LCS "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\LCS\\"
    #define _shared_path_LEVENSHTEIN "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\LEVENSHTEIN\\"
    #define _shared_path_MVM "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\MVM\\"

#elif defined __linux__
    #include "gtest/gtest.h"

    #define _p "/"
    #define _shared_path_CDP "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/CDP/"
    #define _shared_path_DTW "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/DTW/"
    #define _shared_path_ESC "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/ESC/"
    #define _shared_path_FSM "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/FSM/"
    #define _shared_path_LCS "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/LCS/"
    #define _shared_path_LEVENSHTEIN "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/LEVENSHTEIN/"
    #define _shared_path_MVM "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/MVM/"

#endif
```

Figure 1 – *Déclaration des chemins relatifs*

Comme on peut le voir sur l'image ci-dessus, en prenant en compte l'environnement sur lequel le programme fonctionne, chaque algorithme se voit affecter un chemin relatif le liant à son répertoire.

Une version Linux avec l'environnement de développement intégré *CodeBlocks* a été développée. Dans cette version, on retrouve un total de 4 projets correspondant exactement à sa version Windows. Pour ne pas dupliquer le code, des liens sur les mêmes fichiers **.cpp** et **.h** ont été fait. Cela signifie donc que toute modification faite sur *CodeBlocks* sera également fait sur *Visual Studio* et inversement. Par contre, les fichiers **.lib** et **.a** sont créés dans le bon répertoire **library stuff**. Une partie Linux a été ajoutée dans le guide utilisateur/développeur.

5 Optimisation du programme global

1 Détection de bugs trouvés dans la bibliothèque

L'implémentation du projet portant sur les tests systèmes nous a permis d'identifier puis par la suite de corriger d'importants bugs ainsi que d'apporter des améliorations sur son fonctionnement.

1. Les fichiers au format xml ne pouvaient pas être parsé et ainsi un crash de l'application en résultait
2. Certains algorithmes tel que MVM et ESC levaient une exception dans la totalité des cas avant même que les calculs ne soient lancés
3. Lorsqu'un fichier de paramètres valide était utilisé une exception était parfois levée
4. Le poids de chaque élément de la première séquence indiqué dans les fichiers paramètres n'étaient pas insérés dans l'objet paramètre utilisé par la suite par les algorithmes et prenaient toujours la valeur par défaut

2 Modification du code existant

L'un de nos objectif pour ce projet a été de toucher le moins possible au code existant. Ainsi nous avons ajouté des lignes de code lorsque l'on pouvait corriger un problème et procédé à des modifications que lorsque cela était strictement nécessaire.

1. Correction du problème avec les fichiers spécifiquement au format xml

2. Correction des levées d'exceptions anormales dans les algorithmes MVM et ESC provoqué par une méthode vérifiant les valeurs des paramètres

3. Correction du problème portant sur les fichiers paramètres. La première séquence correspond à la séquence cible et la seconde à la séquence référence. Pour les algorithmes FSM, ESC et MVM il fallait inverser l'ordre des deux séquences dans le fichier des paramètres pour que cela fonctionne correctement. Une nouvelle méthode **swapSequences** a ainsi été

ajoutée dans la classe **Parameters** et est appelée dans ces trois algorithmes lorsqu'un tel fichier est utilisé

4. Correction de la méthode readFile de la classe ParamParser afin que les poids des éléments de la première séquence soient ajoutés dans l'objet parameter

5. Suppression et déplacement de certaines partie du code incohérente ou inutile dans les classes MatchingExecutor et CommandLineApplication. Les mutateurs (setters) libèrent désormais la ressource éventuellement allouée. L'attribut isExternParametrage des classes LevenshteinCorrespondance, MVMCorrespondance et LCSCorrespondance était inutile et était confondu avec l'attribut isParametrageExtern de la classe mère Correspondance. Il a donc été supprimé

6. Revue de quelques documentations de méthodes afin de préciser lorsqu'une libération mémoire est nécessaire pour les classes filles de SequenceParser, les classes filles de Element et la classe Sequence

7. Remplacement de constantes préprocesseurs spécifique à Visual Studio tel que -min, –max, FLT-MAX et IN-MAX par des fonctions de la librairie standard afin de rendre la librairie multiplateforme. La méthode min3 de la classe CDP était inutile et a donc été supprimée

8. Ajout d'une classe GtestEnvironment dans le projet des tests unitaire. Cette classe est utilisée par le framework google test et permet l'utilisation d'une variable partagée factorisant en une variable le chemin des fichiers nécessaire à l'exécution des tests unitaires

9. Changement du nom du dossier créé par défaut contenant le résultat retourné afin de savoir quels algorithmes, types de données et fichiers ont été utilisés

3 Fuites mémoires

Le projet original qu'on nous a donné avait beaucoup de fuites mémoires. L'une des phases d'amélioration de celui-ci a donc été de supprimer un maximum de fuites mémoires tout en préservant le code existant. Toutefois, nous n'avons pas eu le temps de gérer tous les cas, particulièrement lorsqu'une exception est levée.

1. Classe CommandLine : La méthode prepareCommand allouait des pointeurs qui n'ont plus d'utilité à la fin de celle ci et n'étaient pas libérés. Ils ont été remplacés par de simple variables

2. Classe CommandLineApplication : Cette classe possède deux attributs de type map dont la clé est un pointeur. Un destructeur libérant les maps a été ajouté. La méthode run n'utilisait pas correctement la méthode readFile de la classe ParamParser qui retourne un vecteur d'objet et non l'objet lui même. De plus, run ne libérait pas ses propres allocations à la fin de son exécution

3. MatchingExecutor : Les méthodes match et exécute ne libéraient pas leurs propres variables allouées

4. Les classes MVMCorrespondance et ESCCorrespondance avaient des variables allouées et ou inutilisées

5. La méthode readFile des classes XMLParser et CSVParser des attributs de type Element n'étaient pas libérés et le parsage de CSV et EXT à XML n'était pas correctement fait

6. Les classes XMLParserTest et CDVParserTest du projet portant sur les tests unitaires utilisaient parfois des free sur des objets et ne libéraient pas le contenu des séquences

6 Conclusion

Nous avons pu voir que l'architecture de ce projet de tests fonctionnels est basé sur principalement sur le fichier GTestEnvironnement.h où tous les chemins relatifs donnant accès aux répertoires de toutes les classes de tests sont définis. Nous avions eu plusieurs idées concernant leur implémentation comme la déclaration de chaque chemin relatif dans une méthode de test. Nous avons tout de même gardé l'architecture gardée précédemment car trouvée plus optimisée.

Aujourd'hui, toutes les classes de tests sont programmées et sont fonctionnelles car elles vérifient bien le fonctionnement de l'algorithme en mode boîte noire. L'optimisation apportée par nos soins à l'ensemble du code est visible sur le temps d'exécution qui en est réduit grâce à la résolution des différents cas de fuites mémoires,... Le projet a abouti et est fonctionnel. Ce projet nous a permis de renforcer nos connaissances sur l'aspect de la qualité logicielle grâce à leur développement et à la manière de les mettre en place pour optimiser au maximum le programme.

7 Annexes

1 Annexe 1 : GTestEnvironnement.h

```
#ifndef GTESTENVIRONMENT_H
#define GTESTENVIRONMENT_H

#include "Readfile.h"
#include "Command.h"
#if defined _WIN32 || defined _WIN64
    #include "gtest\gtest.h"

    #define _p "\\"
    #define _shared_path_CDP "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\CDP\\"
    #define _shared_path_DTW "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\DTW\\"
    #define _shared_path_ESC "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\ESC\\"
    #define _shared_path_FSM "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\FSM\\"
    #define _shared_path_LCS "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\LCS\\"
    #define _shared_path_LEVENSHTEIN "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\LEVENSHTEIN\\"
    #define _shared_path_MVM "..\\MatchingToolBoxTestFonctionnel\\TestingMaterial\\MVM\\"

#elif defined __linux__
    #include "gtest/gtest.h"

    #define _p "/"
    #define _shared_path_CDP "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/CDP/"
    #define _shared_path_DTW "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/DTW/"
    #define _shared_path_ESC "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/ESC/"
    #define _shared_path_FSM "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/FSM/"
    #define _shared_path_LCS "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/LCS/"
    #define _shared_path_LEVENSHTEIN "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/LEVENSHTEIN/"
    #define _shared_path_MVM "../../../VS2015/MatchingToolBox/MatchingToolBoxTestFonctionnel/TestingMaterial/MVM/"

#endif

class GtestEnvironment : public testing::Environment {

    private:

        /*!
        *  \brief global SetUp
        *  Called before all the tests
        */
        virtual void SetUp() {}

        /*!
        *  \brief global TearDown
        *  Called after all the tests
        */
        virtual void TearDown() {}
};
#endif  // GTESTENVIRONMENT_H
```

Figure 1 – *Contenu de GTestEnvironnement.h*

La classe GTestEnvironnement est utilisée sur l'ensemble des tests du **framework Google**

Test. On a des variables pré-processeurs correspondant au chemin d'accès aux fichiers pour faire les tests. On y trouve également les méthodes **SetUp** et **TearDown** ici non utilisés automatiquement appelés respectivement avant et après tous les tests.

2 Annexe 2 : CDPTestFonctionnel.cpp

```
#include "CDPTestFonctionnel.h"

/* System testing with the method Continuous Dynamic Programming*/

TEST_F(CDP_SystemTest, CDP_character_csv) {
    string expected_file = (string)_shared_path_CDP + "expected_files" + _p + "expected_target_character_ref_character_cdp_char_csv.txt";
    string result_path = (string)_shared_path_CDP + "resultat_target_character_ref_character_cdp_char_csv";
    string result_file = result_path + _p + "1.txt";
    string target_file = (string)_shared_path_CDP + "target_character.csv";
    string ref_file = (string)_shared_path_CDP + "ref_character.csv";
    string param_file = (string)_shared_path_CDP + "parameters_char_csv.xml";
    char* args[] = { "MatchingToolBox.exe",
            "-sequences",
            &target_file[0u],
            &ref_file[0u],
            "-method",
            "cdp",
            "-parser",
            "csv",
            "-type",
            "character",
            "-param",
            &param_file[0u],
            "-result",
            &result_path[0u]};
    commandline::CommandLineApplication c = commandline::CommandLineApplication(14, args);
    ASSERT_NO_THROW(c.run());
    Readfile r(expected_file, result_file);
    ASSERT_EQ(r.compareFile(), true);
}

TEST_F(CDP_SystemTest, CDP_numeric_csv) {

    string expected_file = (string)_shared_path_CDP + "expected_files" + _p + "expected_target_numeric_ref_numeric_cdp_num_csv.txt";
    string result_path = (string)_shared_path_CDP + "resultat_target_numeric_ref_numeric_cdp_num_csv";
    string result_file = result_path + _p + "1.txt";
    string target_file = (string)_shared_path_CDP + "target_numeric.csv";
    string ref_file = (string)_shared_path_CDP + "ref_numeric.csv";
    string param_file = (string)_shared_path_CDP + "parameters_num_csv.xml";
    char* args[] = { "MatchingToolBox.exe",
            "-sequences",
            &target_file[0u],
            &ref_file[0u],
            "-method",
            "cdp",
            "-parser",
            "csv",
            "-type",
            "numeric",
            "-param",
            &param_file[0u],
            "-result",
            &result_path[0u]};
    commandline::CommandLineApplication c = commandline::CommandLineApplication(14, args);
    ASSERT_NO_THROW(c.run());
    Readfile r(expected_file, result_file);
    ASSERT_EQ(r.compareFile(), true);
}
```

Figure 2 – *Contenu de CDPTestFonctionnel.cpp*

Voici 2 tests de la classe **CDP-SystemTest**. Ils utilisent la variable pré-processeur suivis d'un éventuel sous dossier et du nom du fichier. Le premier teste le fonctionnement de deux séquences contenues dans des fichiers au format **CSV** avec des éléments de type caractère et le second avec des éléments de type numérique.

3 Annexe 3 : ReadFile.cpp

```cpp
#include "Readfile.h"

Readfile::Readfile()
{}

Readfile::Readfile(string & file1 , string & file2) {
    filename = file1;
    filename2 = file2;
}
bool Readfile::compareFile() {
    bool compare = true;
    string rowfile1;
    string rowfile2;

    /* Check if files exists */
    ifstream file1(filename.c_str(), ios::in);
    ifstream file2(filename2.c_str(), ios::in);

    if (file1.fail()) {
        cerr << "The file 1 : " << filename << " doesn't exist or a problem of reading ! " << endl ;
        return false;
    }
    if (file2.fail()) {
        cerr << "The file 2 : " << filename2 << " doesn't exist or a problem of reading ! " << endl ;
        return false;
    }

    /* Pull all lines from the file 1 */
    while (!file1.eof()) {
        string row = "";
        getline(file1, row);
        rowfile1 += row;
    }
    /* Pull all lines from the file 2 */
    while (!file2.eof()) {
        string row = "";
        getline(file2, row);
        rowfile2 += row;
    }
    /* Remove the character \n \t \r */
    rowfile1 = stringWhithoutCharac(rowfile1, " \n\t\r");
    rowfile2 = stringWhithoutCharac(rowfile2, " \n\t\r");
    if (rowfile1.length() > rowfile2.length() || rowfile1.length() < rowfile2.length()) {
        compare = false;
    }
    else {
        for (unsigned int index = 0; index < rowfile1.length() && compare == true ; index++) {
            if (rowfile1[index] != rowfile2[index]) {
                compare = false;
            }
        }
    }
    return compare ;
```

Figure 3 – *Contenu de ReadFile.cpp*

Cette classe est très importante pour l'ensemble de tous les tests fonctionnels car c'est cette classe qui va faire la lecture du fichier de sortie du premier algorithme et qui va permettre de faire la comparaison.

```
}

string Readfile::stringWithoutCharac(string & sequence, string seqCharactere) {
    string newSequence = "";
    for (unsigned int index = 0; index < sequence.length(); index++) {
        if (seqCharactere.find(sequence[index]) == string::npos) {
            newSequence += sequence[index];
        }
    }
    return newSequence;
}

Readfile::~Readfile()
{}
```

Figure 4 – *Methode de ReadFile.cpp*

4 Annexe 4 : Main du projet

```
#include <iostream>
#include "GtestEnvironment.h"
#if defined _WIN32 || defined _WIN64
    #include "vld.h"
#endif

int main(int argc, char *argv[])
{
    int testResult = 0;
    // Runs tests
    ::testing::InitGoogleTest(&argc, argv);
    testing::AddGlobalTestEnvironment(new GtestEnvironment);
    testResult = RUN_ALL_TESTS();

    std::getchar();
    return testResult;
}
```

Figure 5 – *Contenu du main.cpp*

Rapport de projet de programmation et génie logiciel

Développement d'un sous-projet de tests fonctionnels

Résumé

Un sous-projet de tests a été créé dans le but de tester et de vérifier le comportement des algorithmes de recherche. Des erreurs restaient après l'implémentation de ceux-ci, nous avons donc cherché à optimiser le code après avoir trouvé des fuites mémoire dans la partie de code dédiée aux algorithmes ou les utilisant.

Mots-clés

test fonctionnel, fuite mémoire, algorithme, code, test, optimisation, méthode, génie logiciel

Abstract

A sub-project was created to test the implementation and the working of matching algorithms. A test class was written for each algorithm implemented in the main project. Some errors have also been found during the development phase especially memory leaks. So this project has been optimised to allow the user to execute the test classes as fast as possible.

Keywords

memory leak, algorithm, project, test, method, architecture, optimization

Tuteur académique
Nicolas Ragot

Étudiants
Xavier Bouchenard (DI4)
Raphaël Lazzaroni (DI4)
Abdoulaye Kake (DI4)