Université Paris Nanterre

Licence MIASHS parcours MIAGE

Rapport de stage

L3 MIAGE, parcours classique

Contribution au développement et à la distribution d'une bibliothèque de calcul scientifique

Entreprise d'accueil : Université Paris Nanterre
Stage réalisé du 23 mars 2020 au 22 mai 2020

présenté et soutenu par

Steven EL KHOURY

le 25 mai-26 mai

Jury de la soutenance

M. François Delbot,	Maître de conférences	Responsable du L3 MIAGE
M. François Delbot,	Maître de conférences	Tuteur enseignant
M. Emmanuel Hyon ,	Maître de conférences	Maître de stage

Remerciements

Merci ‘a monsieur Hyon, maître de conférences à Nanterre Université, de m’avoir accepté pour réaliser ce stage. Grâce à son soutien, j’ai pu solidifier l’étendue de mes compétences en algorithmique et en langage orientée objet qui étaient à la base de mes points faibles.
Cela m’a permis de renforcer mon assurance quand à ma volonté de poursuivre mon parcours en MIAGE dans l’optique de devenir développeur. Cette opportunité m’a été donné car Messieurs Hyon et Delbot ont cru en moi et m’ont encadré et soutenu, c’est pourquoi je tiens à leur témoigner toute ma reconnaissance et ma gratitude.

Table des matières

1 Introduction

J'ai fait le choix de réaliser mon stage au sein de l'université même, avec pour maître de stage Monsieur Hyon dans l'optique de progresser et d'avoir un stage qui serait plus centrée sur l'apprentissage que sur une remunération financiere.
Déterminé, il me tient à cœur à terme de continuer mon parcours MIAGE pour atteindre mon objectif professionnel de développeur. C'est aussi pour moi le moyen de combler des lacunes d'un point de vue algorithmique, mais aussi d'asseoir mes compétences techniques acquises cette année, tout en contribuant à un réel projet enrichissant.

2 Entreprise et environnement professionnel

Mon enviromment de travail est celui de l'université Paris Nanterre. L'université Paris Nanterre , a été créée en 1965 avec l'idée qu'elle serait le lieu de l'enseignement du renouveau. De nombreux professeurs en provenance de la Sorbonne et d'ailleurs se sont lancé pour venir y enseigner à cette époque. Nanterre université accueille actuellement plus de trente-trois-mille étudiants, ainsi que plus de deux mille enseignants-chercheurs et maître de conférences, pour seulement sept-cent membres administratifs. Elle propose un large panel de formation (que ce soit en Droit ou en Lettres, en Langues ou en Economie, en gestion ou en Mathématique et informatique). J'appartiens 'a l'une des 8 Unité de Formation et de Recherche présentes sur le campus de Nanterre, elle est composée de 3 départements : Sciences economiques, Gestion, et Mathématiques-Informatique, dont mon maïtre de stage fait aussi partie. Avec l'aide de plus de trois-cent formateurs, cette unité assure la formation par an d'environ quatre mille étudiants tous niveaux confondus. Les cursus proposés sont conçus pour permettre et faciliter une insertion rapide dans l'univers professionnel. L'UFR est très impliquée dans la recherche. C'est pour cela qu'elle dispose de laboratoires de recherche constitués des enseignants-chercheurs. Ils participent à des conférences et des séminaires afin de partager le fruit de leurs recherches(Micro Big par exemple), et ils publient dans des revues de prestige des articles pour diffuser leur travaux. Dans le cadre de mon stage, je travail en télétravail du l'épidémie. Je fais des conférences web mon maître de stage au moins une fois par semaine. Nous faisons un bilan sur ce qui a été réalisé ainsi que les difficultés rencontrées, que ce soit au niveau de mon avancement sur les missions concernant le stage et les TD de java.

3 Missions confiées

3.1 Contexte Marmotte MDP

le logiciel marmoteCore est un environnement ouvert pour la modélisation avec les chaînes de Markov développées dans le cadre du projet MARMOTE. Cette plateforme vise à fournir l'utilisateur scientifique avec des outils pour créer des modèles de Markov et calculer ou simuler leurs comportement. Il fournit une bibliothèque de logiciels mettant en œuvre à la fois des modèles et des méthodes. marmoteCore est conçu pour être une plate-forme collaborative, capable d'intégrer des méthodes de solution et logiciel développé par des équipes indépendantes. Nous décrivons son architecture et certaines de ses fonctionnalités, y compris sa capacité à communiquer avec des logiciels scientifiques établis. La modélisation avec des chaînes de Markov est une activité commune à

de nombreux domaines scientifiques et techniques. Systèmes de particules en interaction de la physique, évolution du génome modèles de biologie, modèles épidémiques de médecine, modèles de population d'écologie, systèmes de recherche opérationnelle, tous ces modèles populaires sont basés sur des chaînes de Markov. Les simulations Monte-Carlo des chaînes de Markov sont couramment utilisées pour produire des échantillons de distributions d'objets combinatoires, de systèmes physiques.

Dans le cadre du Stage nous voulons l'utilisé dans un cadre trivial ; nous avons la vue suivante depuis la fenêtre de notre chambre et nous avons le plaisir de pouvoir observer une marmotte qui se situe sur l'une de ces montagnes à chaque fois. En voici la représentation en image.

FIGURE 1 – les 3 montagnes

A travers la bibliothèque marmotteMDP nous voulons nous assurer que le plaisir de voir la marmotte est optimal avec la vue que nous avons, cela depends de la position de la marmotte a un instant t.(voir modele mathématique)

3.2 Objectifs

Actuellement MarmotteMDP est fonctionnel sur Linux, Notre objectif à terme est d'assurer sa portabilité sur Windows, pour cela nous avons besoin de comprendre le modèle mathématique, la construction du makefile ainsi que divers outils d'usine logicielle.

3.3 Usine Logicielle

Une usine logicielle est un ensemble d'outils pré-configurés, de frameworks, de conventions, de processus, de documentations et de modèles de projets qui structurent les développeurs et leurs développements. L'objectif est d'automatiser au maximum la production et la maintenance L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée. L'intégration continue repose souvent sur la mise en place d'une brique logicielle permettant l'automatisation de tâches : compilation, tests unitaires et fonctionnels, validation produit, tests de performances. À chaque changement du code, cette brique logicielle va exécuter un ensemble de tâches et produire un ensemble de résultats, que le développeur peut par la suite consulter. Cette intégration permet ainsi de ne pas oublier d'éléments lors de la mise en production et donc ainsi améliorer la qualité du produit.

Pour appliquer cette technique, il faut d'abord que :

1. le code source soit partagé en utilisant des logiciels de gestion de versions tels que Git

2. les développeurs intègrent (commit) quotidiennement (au moins) leurs modifications ;

3. des tests d'intégration soient développés pour valider l'application.

L'objectif des test d'intégration esrt que chaque phase de test est de détecter les erreurs qui n'ont pas pu être détectées lors de la précédente phase.

Pour cela, le test d'intégration a pour cible de détecter les erreurs non détectables par le test unitaire et d'une "machine objective"

3.4 Modèle mathématique

3.4.1 Chaine de Markov

Une chaîne de Markov est une suite de variables aléatoires dans le temps ou conditionnellement au présent, le futur ne dépend pas du passé, ou autrement dit le futur ne dépend du passé que par le présent.Formellement, soit E un espace fini ou dénombrable. Ce sera l'espace d'états.

Soit $X = X_n; n \geq 0$ une suite de variables aléatoires à valeurs dans E. On dit que X est une chaîne de Markov si, pour tout $x_0, ...x_{n+1} \in E$

$$\mathbb{P}(\underbrace{X_{n+1} = x_{n+1}}_{\text{Le futur}} \mid \underbrace{X_0 = x_0, X_2 = x_2, \ldots, X_n = x_n}_{\text{Le passé (et le présent)}}) = \mathbb{P}(\underbrace{X_{n+1} = x_{n+1}}_{\text{Le futur}} \mid \underbrace{X_n = x_n}_{\text{Le présent}})$$

3.4.2 Modèle mathématique MDP

La marmotte peut se deplacer entre ces 3 montagnes dans un laps de temps définit, elle a un certaine probabilité de rester sur aussi place et en fonction de la position de la marmotte nous ressentons une satisfaction plus ou moins grandes. Pour modéliser cela nous avons besoins de matrice dite de transition (P0 ;P1 ;P2) qui represente les probabilités de mouvement de la marmotte, nous avons aussi une matrice de gain en fonction de la position de la marmotte nous avons donc les matrices de transitions suivantes :

$$\text{P0} = \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.4 & 0.2 & 0.4 \\ 0.4 & 0.3 & 0.3 \end{bmatrix}$$

$$\text{P1} = \begin{bmatrix} 0.5 & 0.25 & 0.25 \\ 0.6 & 0.2 & 0.2 \\ 0.5 & 0.4 & 0.1 \end{bmatrix}$$

$$\text{P2} = \begin{bmatrix} 0.25 & 0.55 & 0.2 \\ 0.2 & 0.4 & 0.4 \\ 0.3 & 0.4 & 0.3 \end{bmatrix}$$

ainsi que la matrice de gain associé :

$$\text{R0} = \begin{bmatrix} 0.5 & 0.25 & 0.25 \\ 0.6 & 0.2 & 0.2 \\ 0.5 & 0.4 & 0.1 \end{bmatrix}$$

pour obtenir la moyenne de la satisfaction nous avons donc V noté

$$V = E(\sum_{t}^{nbtransition} r_t \mid s_0 = s_i),\ \forall i\ \in [k, nbtransition[\quad (1)$$

avec s la position a l'instant t de la marmotte

3.5 Amélioration des exemples

4 Outils et Technologies

4.1 Environnement technologique

Mon environnement de travail est windows 10, j'ai en plus une machine virtuel Lubuntu sous Virtual Box pour la première compilation de MDPjouetSteven que j'ai coder sous Code Block, pour cette premiere compilation nous avons besoin d'un Makefile.

4.2 Makefile

Le Makefile est un fichier qui permet de réaliser des actions avec la commande make par exemple pour la compilationde projet, mais aussi d'autres actions encore, dans ce cadre, nous n'aborderons que la compilation de projet. les Makefiles ne sont pas normalisés, ils peuvent être rédigés manuellement, et il existe de nombreux utilitaires qui permettent d'en générer automatiquement. Un Makefile est composé de règles, nous allons en decrire la syntaxe. Pour commencer on nomme le fichier cible qui sera l'exécutable, puis ses dépendances (qui sont les fichiers objets et les headers' files dont l'exécutable a besoin) et enfin, un saut de ligne plus loin, la(es) commande(s) qui est précédée d'une tabulation. Cela se présente comme suit.

Une désignant le compilateur utilisée nommée CC (une telle variable est typiquement nommé CC pour un compilateur C, CXX pour un compilateur C++). CFLAGS regroupant les options de compilation (Généralement cette variable est nommées CFLAGS pour une compilation en C, CXXFLAGS pour le C++). LDFLAGS regroupant les options de l'édition de liens. EXEC contenant le nom des exécutables à générer

l'exemple suivant, plus concret sera plus simple a appréhender

```
CC= gcc
CFLAGS = -W -Wall -ansi -pedantic
g=g++

LIBRARIES=$(addprefix -l, MarmoteBase,MarmoteMDP)
INCLUDEDIR=$(MARMORTEDIR)/include
LIBDIR=$(MARMOTEDIR)/lib
MDPDIR=./marmotteMDP/lib
MDPINCLUDE=./marmoteMDP/include

APPLIS= MDP_jouetSteven
all:$(APPLIS)

MDP_jouetSteven: MDP_jouetSteven.cpp
    $g$(CLFLAGS) -I$(INCLUDEDIR)-I(MDPINCLUDE)$^ -o$@ -L$(LIBDIR)-L$(MDPDIR)$(LiBRA
```

4.2.1 Automatisation de la construction

4.3 Les bibliothèques

Comment se servir d'une bibliothèque précompilées en C++ sous linux windows (mettre les chemins vers la position de la bibliothèque).

Voixi les étapes de la création d'un programme :

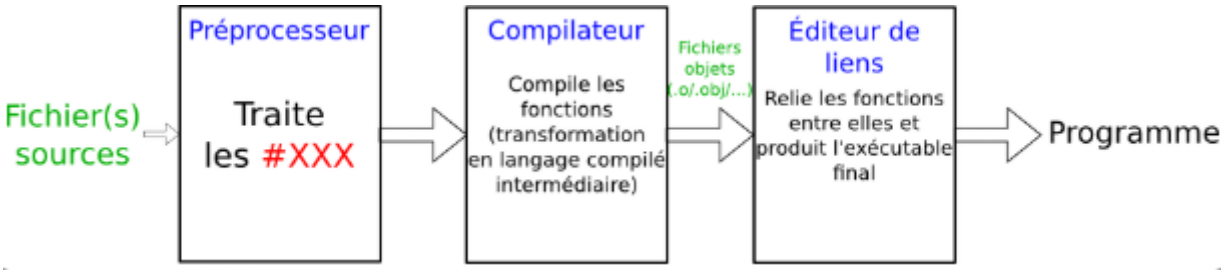

FIGURE 2 – Etapes création programme

Le préprocesseur dans un premier temps s'occupera de traiter les commandes XXX (define, ifdef...) pour produire un fichier source où ces directives ont été exécutées.

le compilateur dans un second temps transforme le source en un fichier intermédiaire (.o/.obj), compilé, mais qui n'est pas exécutable. Le langage utilisé est spécifique au compilateur. Pour valider le code, le compilateur a besoin de connaître tous les symboles que vous utilisez. Le nom d'une fonction, autant que le nom d'une variable sont des symboles. Si on utilise une fonction qui est inconnue, il faut le faire savoir avec un message d'erreur. En effet, pour lui, si la fonction est inconnue, il est incapable de traiter le symbole et donc, incapable de compiler le code. En C et C++, les fichiers .h/.hpp permettent de déclarer les symboles au compilateur et donc de lui dire : " cette fonction existe, voici les paramètres qu'elles acceptent, le type de valeur qu'elle retourne ".

Enfin l'editeur de liens prends les fichier objet du programme pour les assembler et produire un exécutable. Si le code des fonctions que vous utilisez n'est pas dans vos fichiers objets, il va les chercher dans les bibliothèques que vous lui aurez indiquées.

4.3.1 Différence entre bibliothèque statique et dynamique

La bibliothèque statique sera compilée avec le programme et directement intégrée dans l'exécutable final. Il n'y aura donc pas besoin de .so/.dll lors de l'exécution.L'avantage de la compilation statique est de rendre l'exécutable indépendant (il peut fonctionner sur n'importe quelle plateforme compatible, sans pour autant nécessiter l'installation de la bibliothèque

La bibliothèque dynamique fera que l'exécutable ira chercher les fonctions à exécuter dans des fichiers séparés (.so/.dll).

4.4 Git

Git est logiciel de gestion de versions décentralisé. Cela permet avoir une tracabilité de fichier pour savoir qui la modifié, quand à t'il été modifié, qu'est ce qui a été modifié et pourquoi via une zone dite de dépôt.

Git fonctionne par branche, par défault il y a la branche master qui contiendra le projet final,d'autre branche viendront s'ajouter a celle du master qui en seront des copies au moment de leur création, ces branches permettent de definir différentes tache du projet sur lequel on va travaillé et de les implémenté sur le master une fois le travail réalisé.

Le dépôt distant fonctionne à travers une interface web conviviale, GitHub. Un répetoire est créer dans lequel on va pouvoir stocker tout les fichiers que l'on a besoin et de le rendre publique (dans ce cas tout le monde peut y avoir accès) le créateur peut donc en restreindre l'accès et choisir les différents collaborateurs qui vont participer au projet.

Cela permet donc de savoir quel fichier a été modifié, quand a t'il été modifié, par qui, et pourquoi. De plus le système de branche permet de definir les différentes fonctionnalité qui s'ajoute au projets au fur et à mesure de son avancement et d'attribuer les tâches plus clairement.

Voici quelque commande utile à Git :

```
- git remote add origin <url à placer donné par github>

- git push -u prigin master
ces 2 commandes permettent la creation du projet sur git
par rapport aux fichier sur lequel on se trouve

git add <nom du fichier à mettre à jour sur github> <*> pour tout mettre à jour
git commit -m "commentaire sur ce que l'on met à jour"
git push origin <nom de la branche que l'on veut mettre à jour>

git branch pour la liste des branches.

git branch <nom de la branche> permet de crée la branche.

git checkout <nom de la branche> permet d'acceder à la branche.
```

4.5 Compilateur Mingw

MinGW ou Mingw32 (Minimalist GNU for Windows) est une adaptation des logiciels de développement et de compilation du GNU (GCC - GNU Compiler Collection), à la plate-forme Win32. Le développement du projet MinGW s'est ralenti depuis la création en 2005-2008 d'un projet alternatif appelé Mingw-w64. C'est ce dernier que nous allons utilisé.

Pour le projet le compilateur Mingw64 est nécessaire à la création du Makefile permettant de faire compiler un projet utilisant la bibliothèque marmotecore sous windows. je l'ai utiliser à travers MSYS2 qui permet en plus d'utiliser les makefiles

5 Travail réalisé

5.1 Réussite

Dans un premier temps j'ai du comprendre les notions mathématique et les chaine de markov. En parrallèle j'ai du installer la configuration de l'environnement de travail soit Mingw-64 sous Code Block et les différentes options de compilation sous windows, Code block sous Linux. Recuperer l'architecture de MarmotteMDP avec GitHub en utilisant un fork puis un clone de ce dernier et l'initialiser. L'installation de marmotte MDP Dans un seconds temps les makefiles sous Linux et windows (MSYS2 pour windows). J'ai commencé par faire un fichier nommé mdpjouetsteven.cpp qui illustre mon modèle mathématique des processus de Markov en C++. Cela m'a permis de mettre en pratique toute la théorie apprise jusqu'alors, ainsi que de mieux maîtriser MarmotteMDP en la maniant. Ce fichier est sur machine virtuelle sous Linux via Lubuntu.

Il reprend les valeurs que j'ai indiquées dans les exemples de mon modèle mathématique. C'est un modèle à 3 états et 2 actions.

Mes matrices de transitions (celles du modèle mathématique) sont stockées dans des objets qui sont des matrices, elles utilisent le constructeur qui porte le nom de sparceMatrix. Il en va de même pour le stockage du reward et celui du gain associés.

On peut y voir les étapes que sont la construction du processus décisionnel de Markov et le résultat du calcul de la fonction de valeurs. A partir de celui-ci, par observation du résultat pour chaque état on sait quelle est la solution optimale, c'est-à-dire quelle action est la meilleure sachant l'état initial.

enfin rajouter dans le makefile son nom de fichier comme montrer plus haut.

5.2 Difficultés rencontrées

5.3 Perspectives

6 Bibliographie/Webographie

6.1 Binliographie

Références

[label] Auteur, TITRE, editeur, annee

[LAM94] L. LAMPORT, *LaTeX : A Document preparation system, Addison-Wesley, 1994*

6.2 Webographie

Références

[CAT] `savoircoder.fr/cat`

7 Annexes

Annexe A : Exemple d'exécution du projet

Annexe B : Convention de nommage makefile

FIGURE 3 – Convention de nommage des bibliothèques

Annexe C :

usine logiciel et la construction automatique
insertion de bibliothèque pour windows vers le logiciel précompilé en C++ sur Aide