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 à 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.

Je tiens aussi à remercier tout particulièrement certain camarades de promotion nottament Romain Vacherais, Chakir Kraini et Avi Assayag qui ont été pour moi une aide que ce soit d'un point de vue psychologique ou d'une clarification de notion de l'orientée objet. Merci à eux et merci à vous.

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.

Dans le cadre de mon stage, je travail en télétravail dû à l'épidémie. Je fais des réunions avec 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, cela permet de partir avec des bases solide, vérifié et validé, permettant aussi d'avoir des nouvelles missions directement en lien avec ce qui à été réalisé.

2 Entreprise et environnement professionnel

2.1 Présentation de l'entreprise

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).

L'Université est organisé autours de 8 UFR :

— UFR Langues et cultures étrangères (LCE)
— UFR Littérature, langages, philosophie et arts du spectacle (PHILLIA)
— UFR Droit et science politique (DSP)
— UFR Sciences psychologiques et sciences de l'éducation (SPSE)
— UFR Sciences sociales et administratives (SSA)
— UFR Sciences et techniques des activités physiques et sportives (STAPS)
— UFR Systèmes industriels et techniques de communication (SITEC)
— UFR Sciences économiques, gestion, mathématiques et informatique (SEGMI)

J'appartiens à l'Unité de Formation et de Recherche SEGMI, dont mon maitre de stage fait aussi partie, monsieur Emmanuel Hyon est enseignant chercheur mais aussi maitre de conférence à l'université de Paris Nanterre depuis 2004 et nous à donner cours d'algorithme et programmation C lors du premier semestre de la licence MIAGE.

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 confé-

rences 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.

2.2 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 MonteCarlo 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.

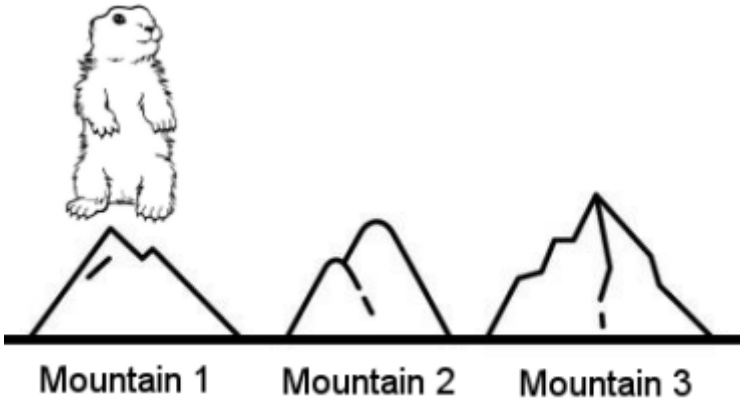

FIGURE 1 – les 3 montagnes

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.

en nous basant sur une matrice de gain RΠ nous obtenons une matrice de gain qui équivaut 0.5 pour la premiere montagne, 0.2 pour la deuxieme montagne et 0.1 pour la 3eme montagne.

Voici donc un exemple avec 7 jours de vacances avec gain :

Jour (instant t)	1	2	3	4	5	6	7
Etat courant de la marmotte(quelle montagne)	1	1	3	2	1	3	2
Gain associé	0.5	0.5	0.1	0.2	0.5	0.1	0.2

2.3 Objectifs

Il m'à été confié de crée des exemples donc de modéliser les différents états possible de la marmotte, les possibilité de transition (en C++). De plus actuellement MarmotteMDP est fonctionnel sur Linux, Notre objectif à terme est d'assurer sa portabilité vers Windows, pour cela nous avons besoin de comprendre le modèle mathématique, la construction du makefile ainsi que divers outils d'usine logicielle.

2.4 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 Outils et Technologies

3.1 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, nous avons besoin d'un Makefile.

3.2 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.
```

3.3 Makefile

3.3.1 but d'un 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

3.3.2 exemple d'un makefile

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
```

3.3.3 Automatisation de la construction

L'automatisation de la construction fait référence à la pratique de la construction automatique de votre logiciel à chaque commit ou une fois par jour. Dans le logiciel, la génération désigne le processus qui convertit les fichiers et autres actifs sous la responsabilité du développeur en un produit logiciel sous sa forme finale ou consommable. L'automatisation de la construction automatise la manière dont le logiciel est construit à l'aide d'outils de construction.

Dans le cas de MarmotteMDP nous avons besoin de plusieurs outils tel que MySYS2 et make. Avec MySYS2 on peut installer make avec la commande

```
pacman -SYu
```

3.4 Les bibliothèques

3.4.1 but d'une bibliothèque

Les bibliothèques peuvent être comparées à des boites à outils. En effet, celles-ci apportent des fonctionnalités supplémentaires que le programmeur va pouvoir utiliser pour réaliser son projet. De plus nous pouvons nous meme créer des bibliothèques qui peuvent etre appelé sur le meme projet.

3.4.2 creation d'une bibliothèque

Comme cité plus haut nous pouvons créer des bibliothèques avec des programmes annexes aux projet que l'on pourra appellé plusieur fois à travers les fonction existante de la bibliothèque en question.

3.4.3 utilisation d'une bibliothèque

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

Nous utilisons régulièrement dans le projet marmotteMDP des bibliothèques que plusieur programmeur du lip6 ont crées nottament celle qui crée les matrice sparseMatrix.h ou celle qui utilise les processus de markov pour arrivé à une solution discountedMDP.h

Il suffit de l'appellé comme une bibliothèque classique. Si elle se trouve sur un autre document il suffit de mettre le chemin sur le include

Voixi les étapes de la création d'un programme avec les différente bibliothèque associé :

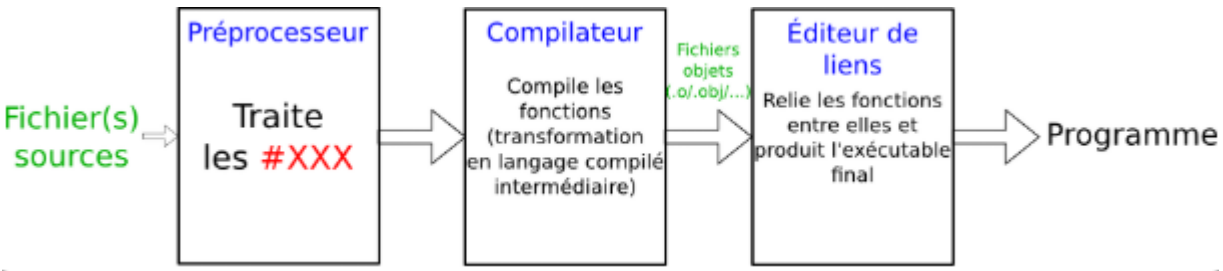

FIGURE 2 – Etapes création programme

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.

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).

3.5 Compilateur architecture

3.5.1 compilation et l'architecture cible

Pour commencer on nomme le fichier cible qui sera l'exécutable, puis ses dependances (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 : cible : dépendances commandes Soit par exemple :

```
exec_hello : hello.o main.o
gcc -o hello hello.o main.o
```

Lorsqu'on utilise un Makefile la première commande, par exemple ici make va évaluer la première règle rencontrée. Cela consiste à vérifier si les fichiers de dépendance du fichier cible ont eux aussi des dépendances. Si c'est le cas on a une nouvelle règle : le(s) fichier(s) concerné(s) devient fichier cible. Et ainsi de suite jusqu'à ce que toutes les dépendances aient été analysées. Lorsque ceci est fait, la réalisation de la(les) commande(s) de la règle est faite si la cible est plus ancienne que les dépendances. Illustrons :

```
exec_hello : hello.o main.o
gcc -o hello hello.o main.o
hello.o : hello.c
gcc -o hello.o -c hello.c -W -Wall -ansi -pedantic
main.o : main.c hello.h
gcc -o main.o -c main.c -W -Wall -ansi -pedantic
```

Pour créer l'exécutable exec hello, make a besoin du fichier hello.o, il va donc faire l'analyse de la règle de hello.o en tant que fichier cible. Sa dépendance est uniquement constituée de hello.c qui n'est pas cible d'une autre règle. Alors make peut exécuter la commande associée.

3.5.2 compilateur mingw sous windows

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.

GCC est une collection d'outils venant tout droit du monde de linux. A ce titre, il est particulièrement attaché aux outils GNU pour sa compilation. L'idéal est donc de disposer d'un système qui puisse émuler linux alors que nous seront sous windows. L'outil que j'ai utilisé pour ce faire est donc Mingw-64.

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

4 Creation développement tutoriaux

4.1 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 3 matrices de transitions suivantes :

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

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

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

ainsi que la matrice de gain associée :

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

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

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

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

4.2 Extension des exemples

4.2.1 Manipulation des matrices

Pour manipuler et declarer les matrices ils nous faut utilisé la classe sparseMatrix et lui declarer sa taille en parametre. Il faut aussi gerer avec min et max (ligne 38 et 39 de l'exemple 1). Attention c'est un tableau donc il faut utilisé les pointeurs.

ensuite il faut remplir la matrice en selectionnant la matrice cible suivi de addToEntry puis les position en commençant par la ligne puis la colonne en partant de 0.

exemple :

```
1    sparseMatrix *Bou = new sparseMatrix(2)
2        Bou->addToEntry(0,0,0.1);
3        Bou->addToEntry(0,1,0.9);
4        Bou->addToEntry(1,0,0.4);
5        Bou->addToEntry(1,1,0.6);
```

Marmotte Intervalle lui sert à definir le nombre d'action effectué par la marmotte. Dans le 1er exemple nous avons 2 matrice de transition donc l'intervalle sera de 0 à 1. dans le second exemple ou le nombre d'action effectué par la marmotte est de 4 nous avons un intervalle de 0 à 3. cela va servir à instancier les differentes matrices de transitions.

5 Portabilité windows

Comme dit précédemment la bibliothèque MarmotteMDP est fonctionnelle sur Linux. Le projet à terme serait de le rendre disponible sur windows. Dans un premier temps Il nous faudra mettre à jour le makefile ainsi que le compilateur choisi. Il existe cependant plusieurs outils tel que CMake.

CMake aussi appelé cross platform make est un système de construction logicielle multiplateforme. Il permet de vérifier les prérequis nécessaires à la construction, de déterminer les dépendances entre les différents composants d'un projet, afin de planifier une construction ordonnée et adaptée à la plateforme. nottament de Linux à Windows.

CMake peut être utilisé à travers l'invite de commande (ou terminal). Il suffit de taper :

```
cmake -G générateur
```

générateur correspond à l'un des générateurs proposés par CMake. Vous devez prendre celui correspondant à l'environnement avec lequel vous voulez compiler

Pour installer les élément de la bibliothèque il faudra preciser le compilateur dans le makefile qui se trouve dans les fichiers sources en indiquant le chemin du compilateur Mingw. Il existe plusieur moyen d'avoir ce chemin il suffit de mettre le compilateur mingw par default sur codeblock, crée un projet, faire un code simple et le compiler. Dans les fichier du projet il y aura un makefile qui sera crée et à partir de ce makefile recuperer le chemin.

5.1 présentation des éléments de la bibliothèque

Dans les bibliothèques utilisé dans le projet nous avons la signature des différentes méthodes et les attributs de classes par exemple finiteHorizonMDP on y retrouve son constructeur, ces attributs et ses différents prototypes. Cela permet de declarer le type d'objet MDP que l'on veut obtenir(il en existe d'autre comme horizon non fini qui serait les prochain exemples) feedbackSolutionMDP.h Il sagit de la bibliothèque qui va servir à l'affichage de la solution

6 Conclusion

6.1 travail réalisé

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.

6.2 Réussite

Comme cité plus haut les premières taches que j'ai eu ont été essantiel a mon stage car il en va de la comprehension du sujet et de son application direct. En effet dans un premier temps les notions mathématique tel que les processus et les chaînes de Markov afin que je puissent les manipuler et y établir un modèle mathématique pour enfin le mettre en pratique en C++. J'ai du appréhender pour la première ce language et y manipuler des objets et bibliothèques. De surcroît la creation et la manipulation de makefile m'a du demandé de m'interrogé et d'utilisé différents compilateurs nottament MINGW . GCC est une collection d'outils venant tout droit du monde de linux. A ce titre, il est particulièrement attaché aux outils GNU pour sa compilation.

6.3 Difficultés rencontrées

Etant donné les circonstances dû à l'épidémie, le travail n'a pas été aisée et ne se sont pas passé dans les meilleures conditions, ce qui à entrainée des ralentissement sur ce qu'avais prévu mon tuteur de stage.

Les premières difficultés concernant le stage en lui-même sont l'installation de l'environnement de travail en effet celui ci etant assez complexe l'installation en est de meme que ce soit pour configurer l'éditeur de code pour gérer le make file ainsi que les projets. En plus d'utiliser C++ pour la premiere et ayant des difficultés avec la programmation orientée objet. La compréhension,l'utilisation et la manipulation des exemples ont été complexe pour moi. Une fois le programme prêt, il restait encore à configurer le makefile. Aujourd'hui je suis bien plus a l'aise avec cette bibliothèque.

6.4 Perspectives

Le stage est encore loin d'être fini, de futures missions seront donnée pour arrivée à l'objectif initial qui est la portabilité vers windows. à travers le make file. J'ai cependant enormément progresser dans l'orienté objet, j'ai pu apprendre beaucoup de notion matémathique à travers ce projet.

7 Bibliographie/Webographie

7.1 Bibliographie

Références

[1] Rémy Malgouyres, Initiation à l'algorithmique et la programmation en C, Dunod, 2008

[2] Frederick Garcia, Markov Decision Problem, pdf, 2008

[3] Emmanuel Hyon, An introduction to Markov Decision Processes, pdf, 2019

7.2 Webographie

Références

[CAT] `savoircoder.fr/cat`

[Makefile] `https://cognitivewaves.wordpress.com/makefiles-windows/`

[MINGW64] `http://wiki.codeblocks.org/index.php/MinGW_installation`

[TDM] `https://jmeubank.github.io/tdm-gcc/download/`

[C++] `https://cpp.developpez.com/cours/cpp/?page=page_8`

[C] `http://www.cplusplus.com/articles/yAqpX9L8/`

8 Annexes

Annexe A : Exemple 1 reward R0

```
sparseMatrix *R0  = new sparseMatrix(3);
R0->addToEntry(0,0,20);
R0->addToEntry(0,1,-5);
R0->addToEntry(1,0,5);
R0->addToEntry(1,1,5);
R0->addToEntry(2,0,10);
R0->addToEntry(2,1,0);
```

FIGURE 3 – configuration de la matrice de gain

Annexe B : Compilation du projet

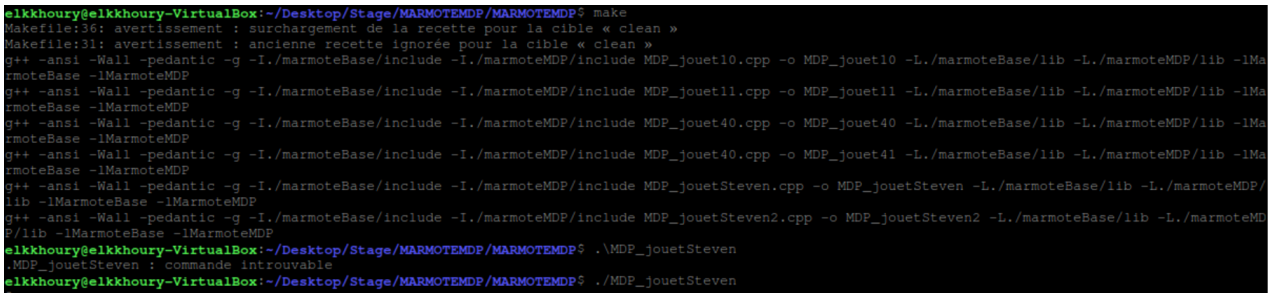

FIGURE 4 – compilation avec make du projet et construction automatique

Annexe C : Exemple d'exécution du projet

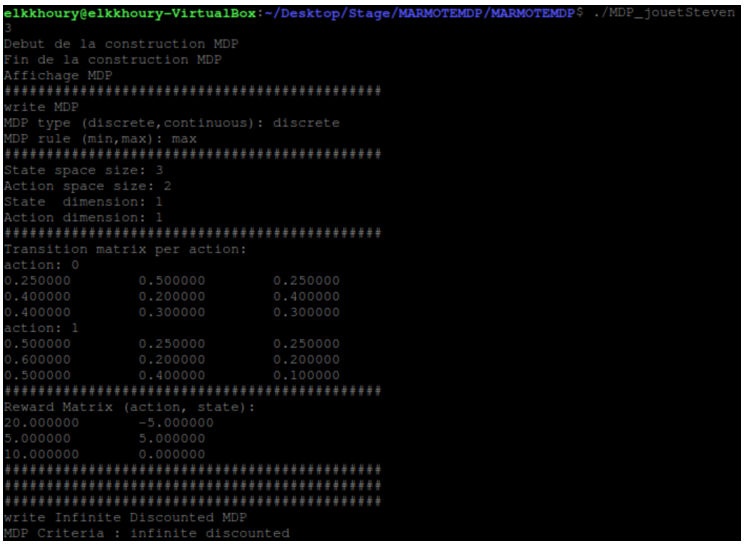

FIGURE 5 – execution du projet 1/2

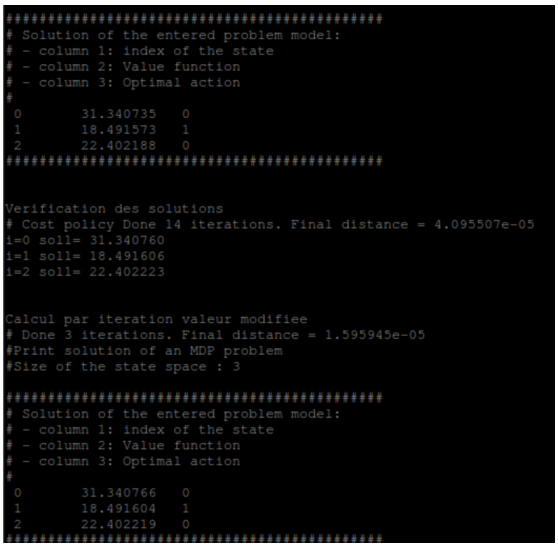

FIGURE 6 – execution du projet 2/2

Annexe D : installation Mingw

Pour installer Mingw ils vous 2 liens

`http://wiki.codeblocks.org/index.php/MinGW_installation`

`https://jmeubank.github.io/tdm-gcc/download/`

Il vous faut installer TDM puis à travers lui Mingw en selectionnant la version 64

Annexe E : Makefile (Linux)

```
MARMOTEDIR=./marmoteBase
CFLAGS=-ansi -Wall -pedantic -g -I
LDFLAGS = -L lib -lmingw32 -lSDL2main -lSDL2 -mwindows
CC=gcc

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

APPLIS= MDPJouetSteven MDPJouetSteven2

all: $(APPLIS)

MDPJouetSteven: Exemple/MDPJouetSteven.cpp
    $(CC) $(CFLAGS) -I$(INCLUDEDIR) -I$(MDPINCLUDE) $^ -o $@ -L$(LIBDIR) -L$(MDPDIR) $(LIBRARIES) -c Exemple/MDPJouetSteve

MDPJouetSteven2: Exemple/MDPJouetSteven2.cpp
    $(CC) $(CFLAGS) -I$(INCLUDEDIR) -I$(MDPINCLUDE) $^ -o $@ -L$(LIBDIR) -L$(MDPDIR) $(LIBRARIES) -c Exemple/MDPJouetSteve


clean:
    del *.o $(APPLIS)

cleanall: clean
    del *.exe
```

FIGURE 7 – Makefile (Linux)

Annexe F : Code exemple 1

:

```
    #include <iostream>
#include <sstream>
#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <values.h>
#include <time.h>
#include <list>
#include <vector>
#include <string>

using namespace std;

#include "Set/marmoteInterval.h"
#include "Set/marmoteSet.h"
#include "transitionStructure/sparseMatrix.h"
#include "discountedMDP.h"
#include "feedbackSolutionMDP.h"
#include "solutionMDP.h"


int main( int argc, char** argv )
{
  int i;
  //create and initialize the discount factor.
  double beta = 0.5;
  string critere("max");

  //create and initialize epsilon.
  double epsilon = 0.0001;

  //create and initialize the maximum number of iterations allowed.
  int maxIter = 700;

  //==================Test new features==================//

  //Create the MDP object to test 1-dimension example and fill all its fields.
  int min = 0;
  int max = 2;
  marmoteSet *actionSpace = new marmoteInterval(0,1);
  marmoteSet *stateSpace = new marmoteInterval(min,max);

  vector<sparseMatrix*> trans(actionSpace->cardinal());
  sparseMatrix *P1 = new sparseMatrix(3);
  P1->addToEntry(0,0,0.25);
  P1->addToEntry(0,1,0.5);
  P1->addToEntry(0,2,0.25);
  P1->addToEntry(1,0,0.4);
  P1->addToEntry(1,1,0.2);
  P1->addToEntry(1,2,0.4);
  P1->addToEntry(2,0,0.4);
  P1->addToEntry(2,1,0.3);
  P1->addToEntry(2,2,0.3);
  trans.at(0) = P1;

  sparseMatrix *P2 = new sparseMatrix(3);
  P2->addToEntry(0,0,0.5);
  P2->addToEntry(0,1,0.25);
  P2->addToEntry(0,2,0.25);
  P2->addToEntry(1,0,0.6);
  P2->addToEntry(1,1,0.2);
  P2->addToEntry(1,2,0.2);
  P2->addToEntry(2,0,0.5);
  P2->addToEntry(2,1,0.4);
  P2->addToEntry(2,2,0.1);
  trans.at(1) = P2;

  sparseMatrix *R1  = new sparseMatrix(3);
```

```
  R1->addToEntry(0,0,20);
  R1->addToEntry(0,1,-5);
  R1->addToEntry(1,0,5);
  R1->addToEntry(1,1,5);
  R1->addToEntry(2,0,10);
  R1->addToEntry(2,1,0);


  std::cout << critere.size() << std::endl;
  fprintf(stdout,"Debut␣de␣la␣construction␣MDP\n");
  discountedMDP *mdp1 = new discountedMDP(critere, stateSpace, actionSpace, trans, R1,beta);
  fprintf(stdout,"Fin␣de␣la␣construction␣MDP\n");

  fprintf(stdout,"Affichage␣MDP\n");
  mdp1->writeMDP();

  fprintf(stdout,"Affichage␣solution␣iteration␣valeur\n");
  //call the function to solve the MDP.
  solutionMDP *optimum = mdp1->valueIteration(epsilon, maxIter);
  optimum->writeSolution();

  fprintf(stdout,"\n\nVerification␣des␣solutions\n");
  double *sol1 = mdp1->policyCost(optimum,epsilon, maxIter);
  for(i=0;i<stateSpace->cardinal();i++){
         printf("i=%d␣sol1=␣%f␣\n",i,sol1[i]);
  }


  fprintf(stdout,"Destruction\n");
  mdp1->clearRew();
  delete mdp1;
  delete optimum;
  delete optimum2;


  fprintf(stdout,"Destruction␣2\n");
  delete actionSpace;
  delete stateSpace;

  fprintf(stdout,"Destruction␣3\n");

  delete P1;
  delete P2;

  free(sol1);

  return 0;
}
```

Listing 1 – MDPJouetSteven.cpp

Annexe G : Code exemple 2

:

```

#include <iostream>
#include <sstream>
#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <values.h>
#include <time.h>
#include <list>
#include <vector>
#include <string>

using namespace std;

#include "Set/marmoteInterval.h"
#include "Set/marmoteSet.h"
#include "transitionStructure/sparseMatrix.h"
#include "discountedMDP.h"
#include "feedbackSolutionMDP.h"
#include "solutionMDP.h"


int main( int argc, char** argv )
{
  int i;
  //create and initialize the discount factor.
  double beta = 0.5;
  string critere("max");

  //create and initialize epsilon.
  double epsilon = 0.0001;

  //create and initialize the maximum number of iterations allowed.
  int maxIter = 700;

  //==================Test new features==================//

```

```
  int min = 0;
  int max = 2;
  marmoteSet *actionSpace = new marmoteInterval(0,3);
  marmoteSet *stateSpace = new marmoteInterval(min,max);

  vector<sparseMatrix*> trans(actionSpace->cardinal());
  sparseMatrix *P1 = new sparseMatrix(3);
  P1->addToEntry(0,0,0.25);
  P1->addToEntry(0,1,0.5);
  P1->addToEntry(0,2,0.25);
  P1->addToEntry(1,0,0.4);
  P1->addToEntry(1,1,0.2);
  P1->addToEntry(1,2,0.4);
  P1->addToEntry(2,0,0.4);
  P1->addToEntry(2,1,0.3);
  P1->addToEntry(2,2,0.3);
  trans.at(0) = P1;

  sparseMatrix *P2 = new sparseMatrix(3);
  P2->addToEntry(0,0,0.5);
  P2->addToEntry(0,1,0.25);
  P2->addToEntry(0,2,0.25);
  P2->addToEntry(1,0,0.6);
  P2->addToEntry(1,1,0.2);
  P2->addToEntry(1,2,0.2);
  P2->addToEntry(2,0,0.5);
  P2->addToEntry(2,1,0.4);
  P2->addToEntry(2,2,0.1);
  trans.at(1) = P2;

  sparseMatrix *P3 = new sparseMatrix(3);
  P3->addToEntry(0,0,0.25);
  P3->addToEntry(0,1,0.55);
  P3->addToEntry(0,2,0.2);
  P3->addToEntry(1,0,0.2);
  P3->addToEntry(1,1,0.4);
  P3->addToEntry(1,2,0.4);
  P3->addToEntry(2,0,0.4);
  P3->addToEntry(2,1,0.3);
  P3->addToEntry(2,2,1);
  trans.at(1) = P3;

  sparseMatrix *P4 = new sparseMatrix(3);
  P4->addToEntry(0,0,0.1);
  P4->addToEntry(0,1,0.2);
  P4->addToEntry(0,2,0.7);
  P4->addToEntry(1,0,0.0);
  P4->addToEntry(1,1,0.1);
  P4->addToEntry(1,2,0.9);
  P4->addToEntry(2,0,0.0);
  P4->addToEntry(2,1,0.0);
  P4->addToEntry(2,2,1);
  trans.at(1) = P4;

  sparseMatrix *R1  = new sparseMatrix(4);
  R1->addToEntry(0,0,3.025);
  R1->addToEntry(0,1,0.225);
  R1->addToEntry(0,2,2.07);
  R1->addToEntry(0,3,0.77);
  R1->addToEntry(1,0,4.24);
  R1->addToEntry(1,1,1.22);
  R1->addToEntry(1,2,1.44);
  R1->addToEntry(1,3,-0.31);
  R1->addToEntry(2,0,4.33);
  R1->addToEntry(2,1,0.41);
  R1->addToEntry(2,2,2.43);
  R1->addToEntry(2,3,-0.4);



  std::cout << critere.size() << std::endl;
  fprintf(stdout,"Debut de la construction MDP\n");
  discountedMDP *mdp1 = new discountedMDP(critere, stateSpace, actionSpace, trans, R1,beta);
  fprintf(stdout,"Fin de la construction MDP\n");

  fprintf(stdout,"Affichage MDP\n");
  mdp1->writeMDP();

  fprintf(stdout,"Affichage solution iteration valeur\n");
  //call the function to solve the MDP.
  solutionMDP *optimum = mdp1->valueIteration(epsilon, maxIter);
  optimum->writeSolution();

  fprintf(stdout,"\n\nVerification des solutions\n");
  double *sol1 = mdp1->policyCost(optimum,epsilon, maxIter);
  for(i=0;i<stateSpace->cardinal();i++){
        printf("i=%d sol1= %f \n",i,sol1[i]);
  }

  //call the function to solve the MDP.
  fprintf(stdout,"\n\nCalcul par iteration valeur modifiee\n");
  solutionMDP *optimum2 = mdp1->policyIterationModified(epsilon, maxIter, 0.001, 100);
  optimum2->writeSolution();


  fprintf(stdout,"Destruction\n");
  mdp1->clearRew();
  delete mdp1;
  delete optimum;
  delete optimum2;
```

```
141
142
143   fprintf(stdout,"Destruction 2\n");
144   delete actionSpace;
145   delete stateSpace;
146
147   fprintf(stdout,"Destruction 3\n");
148
149   delete P1;
150   delete P2;
151   delete P3;
152   delete P4;
153
154   free(sol1);
155
156   return 0;
157 }
```

Listing 2 – MDPJouetSteven2.cpp