

Licence MIASHS parcours MIAGE

## Rapport de stage

L3 MIAGE, parcours classique

# Implémentation d'algorithmes pour modèles de jeu stochastiques

Entreprise d'accueil : Université Paris Nanterre  
Stage réalisé du 23 mars 2020 au 25 mai 2020

*présenté et soutenu par*

**Avi ASSAYAG**

le 1 juin 2020

### Jury de la soutenance

M. François Delbot,  
M. François Delbot,  
M. Emmanuel Hyon ,

Maître de conférences  
Maître de conférences  
Maître de conférences

Responsable du L3 MIAGE  
Tuteur enseignant  
Maître de stage

## Remerciements

Merci à Monsieur Hyon, maitre de conférence à l'Université de Nanterre et chercheur dans l'équipe SYSDEF du Lip6 , d'avoir accepter le poste de tuteur pour mon stage de Licence 3 MIAGE. Grâce a son accompagnement personnel j'ai pu solidifier mes compétences algorithmiques (Java et Python) mais aussi découvert d'autre aspect de la programmation linéaire.

Cette opportunité n'a été seulement possible que par la collaboration de Monsieur Emmanuel Hyon, mon tuteur ainsi que Monsieur François Delbot, responsable de la Licence 3, et les remercie de leur patience , de leur encouragement et de le encadrement tout au long de ce stage.

Merci aussi aux autres professeurs qui ont contribué tout a long de l'année à parfaire toutes nos compétences autant sur le plan théorique que techniques.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Contexte du Stage</b>	<b>5</b>
2.1	Présentation de l'entreprise . . . . .	5
2.2	Présentation de l'équipe . . . . .	5
2.3	Mission proposé . . . . .	5
2.4	Cahier des charges . . . . .	5
<b>3</b>	<b>Outils utilisés</b>	<b>6</b>
3.1	GitHub . . . . .	6
3.1.1	Initialisation . . . . .	6
3.1.2	Branches . . . . .	6
3.1.3	Dépôt et mises à jours . . . . .	7
3.1.4	Clonage . . . . .	7
3.2	Python . . . . .	7
3.2.1	Installation de Python . . . . .	8
3.2.2	Un petit code Python . . . . .	8
3.2.3	La programmation objet en Python . . . . .	9
3.3	Gurobi . . . . .	10
3.3.1	Installation de Gurobi . . . . .	10
3.3.2	Méthode via Anaconda . . . . .	11
3.4	Shell interactive . . . . .	11
3.5	Exemple Gurobi programmation linéaire (biere.py) . . . . .	12
<b>4</b>	<b>La théorie des jeux</b>	<b>13</b>
4.1	Jeux statiques et dynamiques . . . . .	13
4.1.1	Statiques . . . . .	13
4.1.2	Dynamiques . . . . .	13
4.2	Stratégies . . . . .	13
4.3	Jeu bimatriciel . . . . .	13
4.4	Jeu à sommes nulles . . . . .	14
<b>5</b>	<b>Webographie</b>	<b>15</b>
<b>6</b>	<b>Annexes</b>	<b>16</b>
<b>A</b>	<b>Python en général</b>	<b>16</b>
A.1	Structure Conditionnelle If . . . . .	16
A.2	Structure Conditionnelle Else . . . . .	16
A.3	Structure Conditionnelle Elif . . . . .	16
A.4	Boucle For . . . . .	16
A.5	Boucle While . . . . .	17
A.6	Les fonctions . . . . .	17
<b>B</b>	<b>L'orienté objet en Python</b>	<b>17</b>
<b>C</b>	<b>GitHub</b>	<b>18</b>

<b>D Gurobi</b>	<b>19</b>
<b>E CV</b>	<b>20</b>

# 1 Introduction

Pendant ces semaines de stage, nous allons essayer d'implémenter des algorithmes pour résoudre des modèles de jeux stochastiques, plus précisément des jeux de gain à somme nul (que nous représenterons sous forme bimatricel).

L'objectif est dans un premier temps de concevoir une modélisation informatique de ces jeux puis dans un second temps implémenter un ou des algorithmes permettant de résoudre ces jeux.

Pour parfaire à ces attentes, nous allons utiliser le langage **Python**, non utilisé durant le cursus scolaire actuel, le solveur **Gurobi**, que nous utiliserons afin de résoudre des programmes linéaires et pour l'orienté objet **Python** et la plateforme **GitHub**, l'hébergeur de code, pour avoir accès à tout les codes sources, document qui m'ont aidé à réaliser ce stage.

L'utilisation de GitHub n'était pas obligatoire, mais elle plus que judicieuse pour que mon tuteur Mr Emmanuel Hyon puisse avoir accès en temps réel à mon code afin de m'orienter si je m'écarte du sujet. C'est donc à son initiative que nous avons utilisé **Github** tout au long de ce stage.

Dans les chapitres qui suivront nous allons expliciter différents concepts relatifs aux modèles stochastiques (notamment le principe même de la théorie des jeux) mais aussi les outils utilisés ; comment les installer et les utiliser.

Enfin nous tenterons de rédiger et de résoudre trois modèles de jeu à sommes nulles , c'est à dire un jeu ou le gain d'un des acteurs représente la perte exacte des autres acteurs de ce jeu, mais nous l'expliquerons en détails dans une prochaine section [6.4]. Nous modéliserons alors :

- Le jeu pierre feuille-papier-ciseau
- Le jeu pile ou face
- Le jeu tir au but

## 2 Contexte du Stage

### 2.1 Présentation de l'entreprise

### 2.2 Présentation de l'équipe

### 2.3 Mission proposé

### 2.4 Cahier des charges

## 3 Outils utilisés

### 3.1 GitHub

GitHub est un service d'hébergement web (un peu comme une sorte de Drive) et de gestion de développement de logiciel lancé en 2008. Ce dernier est codé principalement en Ruby et Erlang par différents programmeurs : Chris Wanstrath, PJ Hyett et Tom Preston-Werner.

Aujourd'hui cette plateforme compte plus de 15 millions d'utilisateurs et enregistre environ 40 millions de dépôts de fichiers, se plaçant donc en tête du plus grand hébergeur source code mondial.

Le fonctionnement de Git est assez simple, on crée un répertoire (un référentiel / repository) dans lequel on va stocker tout les fichiers que l'on désire et on peut soit rendre l'accès public (au quel cas tout le monde peut rejoindre et consulter ces fichiers) ou alors le restreindre en accès privé (au quel cas c'est le créateur qui décide quels seront les collaborateurs ayant droit de consultation des fichiers).

Ensuite cela s'agit comme une sorte de réseau constitué de branches (branch) où chaque branches représentent un collaborateur ainsi que la master qui correspond au créateur du référentiel.

Une des caractéristiques de Git repose sur le fait que c'est un outil de versionning (gestion de version) est donc permet de si le fichier a été modifié ; si oui par qui et quand a eu lieu la modification et quel fichiers ont été affectés. Cela permet notamment de pouvoir faire des travaux de groupe sur le même sujet (un site ou une application par exemple) où chacun doit travaillé sa partie mais nécessite les parties des autres membres du groupes (mis à jours régulièrement). Évidemment toutes les étapes (initialisation, dépôts, fusion et clonage) se font à l'aide de lignes de commandes sur le terminal (en bash) que j'expliquerai un peu plus loin ainsi que les commandes principales pour chaque étapes.

#### 3.1.1 Initialisation

Pour créer un projet il suffit d'aller sur le site <https://github.com/> puis Repositories → New et remplir les informations données avant de valider. Ensuite pour initialiser le Git (et que la branch master existe ; elle sera créée automatiquement à l'instanciation du projet) il faut se placer dans le dossier (en ligne de commande cd) et taper : git init

Ensuite il faudra taper : git remote add origin < lien donnée par git hub > puis git push -u origin master qui respectivement créons le répertoire du projet et ensuite la zone de dépôt.

#### 3.1.2 Branches

Comme nous l'avons dit plus haut le projet est contenu dans la branche principale la master et grâce à des copies de branches le projet acquiert une plus grande flexibilité

qui permet d'incrémenter au fur et mesure le projet.

Pour ajouter une branche il suffira simplement de taper `git branch < nom de la branche >` et pour supprimer une branche il faut rajouter l'option `-d` a la commande soit : `git branch -d < nom de la branche >`.

Pour changer de branche (afin d'effectuer un dépôt ou autre) il faudra taper : `git checkout < nom de la branche >` et enfin pour visualiser l'ensemble des branches existantes on devra taper : `git branch` .

### 3.1.3 Dépôt et mises à jours

Avant toute chose il faut savoir sur quelle branche déposer le fichier puis il faudra taper les commandes suivantes pour les ajouter au fichier : `git add < nom des fichier >` (ou `*` pour tout ajouter) puis `git commit -m message` et enfin pour finir `git push origin < nom de la branche>`.

Pour récupérer des modifications faites sur le projet il suffit à l'inverse de taper : `git pull origin master`.

### 3.1.4 Clonage

Une fois les autres branches (celles des différents collaborateurs) créées il faut juste qu'il copie le lien du git pour pouvoir travailler dessus et effectuer les futurs dépôts. En premier lieu il faudra taper : `git clone < lien du git >` puis effectuer la commande `git pull origin master` pour récupérer les fichiers de la branche master et enfin faire les commandes relatives au dépôt (vu plus haut).

## 3.2 Python

Python est un langage de programmation à part entière dont la première version fut développée par **Guido van Rossum** et lancée en 1991. Ce langage est facile d'utilisation et ne possède pas forcément de syntaxe particulières seulement une indentation permettant au compilateur intégré de suivre les blocs d'instructions.

Ce langage permet donc une multitude de possibilité de code mais aussi d'action puisqu'il existe des bibliothèques déjà implémentées et il suffira seulement des les utiliser comme bon nous le semble (par exemple Matplotlib ou encore Networkx etc...). Malheureusement Python n'est pas le langage le plus rapide d'exécution contrairement au C ou C++ et Java mais il permet tout de même d'accéder à des fonctionnalités que d'autres langages ne peuvent proposer.

Contrairement au C, Python admet des types sophistiqués supplémentaires tel que les **Listes**, les **Dictionnaires**, les **Sets** et les **Tuples**. Il en va de soit que les types primitifs sont aussi présent **int, float, double, boolean** etc... Mais le réel avantage du langage repose sur le fait que l'on ne se soucie pas du type de retour d'une fonction ni de la déclaration du type du paramètre ainsi que le langage admet la possibilité d'être orienté objet.

Python est un langage interprété et donc n'a pas besoin de passer par un compilateur comme GCC (GNU Compiler Collection), tout se fait directement sur la console

une fois l'environnement installé.

Quant à l'installation de Python, cette dernière est assez simple ; il suffit d'aller sur le site officiel et télécharger la version en question (aujourd'hui version 3.8.2) et ensuite de l'installer. Il existe différentes méthodes d'activation du langage, qui représente chacune l'environnement de la machine (Windows, Mac OS ou encore Linux).

A savoir que sur Mac Os et Linux, Python est déjà préinstallé et il faudra peut être seulement mettre à jours la version qui pourrai être obsolète ou dépassé.

### 3.2.1 Installation de Python

#### 3.2.1.1 Méthode packages

Pour cela il faut allez télécharger les packages en question sur le site officiel de Python puis les interpréter c'est dire ouvrir la console (terminal python) et demander à python d'exécuter le fichier `.py` en question via la commande :

```
python setup.py install
```

#### 3.2.1.2 Méthode module Pip

Il s'agit d'une des méthodes les plus simple, après avoir téléchargé les packages Python sur le site, on installe tout les modules externes (pip , Django etc ...) que l'on pourrai avoir besoin d'utiliser par la suite via le terminal :

```
pip install <nom_module>
```

### 3.2.2 Un petit code Python

Pour déclarer une variable il suffit seulement de la nommé, Python n'attend pas forcément le type de la variable ; tout comme pour une fonction il n'attend pas le type de retour de la fonction. Ensuite pour les boucles et les conditions il suffit d'utiliser le mot clé en question suivit de `:"` et donnez les instructions de façon indenter

Rien de mieux qu'un petit code Python pour mieux comprendre la syntaxe et la facilité d'utilisation du langage. Ainsi je vais vous présenter un code source de la fonction Tri à bulles :

```
def tri_a_bulle(tab) :  
    taille=len(tab)  
    for i in range(taille) :  
        for j in range(taille-1) :  
            if tab[j] > tab[j+1] :  
                tab[j] , tab[j+1] = tab[j+1] , tab[j]  
    return tab
```



### 3.2.3 La programmation objet en Python

Python permet aussi l'utilisation de l'orienté objet, c'est donc un des autres plus de ce langage puissant et aux vagues possibilité. Dans cette partie nous allons vous montrer comment coder un objet en Python et aussi le construire. Nous allons donc voir la syntaxe générale d'une **Classe** et celle d'un **Constructeur**. Enfin pour terminer cela nous implémenterons un objet "Pullover" avec différents attributs et son propre constructeur.

#### 3.2.3.1 Code générique Classe "Lambda"

Pour déclarer un objet il suffit simplement d'utiliser le mot `Class` suivit de `:"` et ensuite déclarer des variables ou autres instructions.

```
Class Personne :  
    name  
    age
```

#### 3.2.3.2 Constructeur de la Classe "Lambda"

Pour déclarer le constructeur d'un objet il faut utiliser la méthode `init()` au sein de la classe en passant en paramètre ceux de l'objet en question. La petite différence par rapport à d'autres langages de programmation orienté objet (C++ ou Java) est l'utilisation du paramètre (mais aussi mot clé) `self` au seins du constructeur. En réalité `self` n'est autre que la première référence de l'instance de l'objet que l'on va créer.

```
def __init__(self, name, age) :  
    self.name = name  
    self.age = age
```

#### 3.2.3.3 Exemple de class : Pullover

Maintenant que nous avons une première approche de la syntaxe objet essayons de mettre cela en application avec quelque chose de plus concret. Nous allons créer un "Pullover" avec comme attribut : une marque, une taille, un nom de modèle, une couleur et un prix

```
Class Pullover :  
    brand  
    size  
    model_name  
    color  
    price  
  
    def __init (brand, size, model_name, color, price) :           #constructeur  
        self.brand=brand  
        self.size=size  
        self.model_name=model_name  
        self.color=color  
        self.price=price  
  
    Pull1 = Pullover("ZARA", "XS", "AED934", "black", 19)       #instanciation
```

### 3.3 Gurobi

La plateforme Gurobi est un solveur mathématique autrement dit c'est une optimisation mathématique. Il traduit un problème commercial en un énoncé mathématique. Gurobi a été écrit pour prendre en considération différentes interfaces sous différents langage : C, C++, Java, Python et R.

Il y a deux méthodes d'installation soit directement avec une licence (payante ou gratuite) ou alors avec la distribution **Anaconda** que nous allons tenter d'expliquer.

Travaillant sur MacOS, j'ai opter pour l'installation de **Gurobi** en privé sur ma machine et donc en gérant l'installation sur mon environnement Python, et utiliserait donc le module *gurobipy*. De plus il m'a fallu créer un compte chez **Gurobi** pour utiliser une licence académique gratuite bien évidemment.

#### 3.3.1 Installation de Gurobi

##### 3.3.1.1 Méthode classique

Il est aussi possible d'installer **Gurobi** directement sur la machine en gardant notre environnement configuré par nos propre soins puisque l'environnement Python a pensé cela lors de sa conception.

##### 3.3.1.2

Pour cela il faudra au préalable télécharger le solveur sur le site web de **Gurobi** (le lien est en annexe) et attendre le téléchargement. Une fois terminé il suffit d'exécuter le fichier télécharger (en double cliquant dessus) pour démarrer l'installation. Durant cette dernière le système d'exploitation nous demandera dans quel dossier stocker les packages nécessaire a **Gurobi**. Ensuite il faudra se rendre à cette emplacement, via un terminal et exécuter la commande suivante :

```
python setup.py install
```

**3.3.1.3 Après avoir créer son compte afin d'obtenir un licence il faudra l'enregistrer sur la machine afin de pouvoir utiliser le solveur sans souci, pour cela il faudra ouvrir le terminal et exécuter la commande suivante :**

```
grbgetkey 4fd46a16-7d9c-11ea-809f-020d093b5256
```

**3.3.1.4 Une fois tout ceci effectué et donc paramétré il faudra, pour utiliser le solveur, faire un `import` du module et donc de la bibliothèque **Gurobi** au début du script python que l'on élabore.**

```
import gurobipy as gp
from gurobipy import *
```

### 3.3.2 Méthode via Anaconda

3.3.2.1 Pour essayer de faire simple, Anaconda est une solution libre office, c'est à dire à téléchargement gratuit, qui permet une installation rapide et simple de **Python** avec un interpréteur (IDE) ainsi que de nombreuses bibliothèques (les plus utilisées et les plus utiles),gurobi par exemple. Par la suite si ont veux ajouter des modules ou bibliothèques supplémentaires on le fait comme avec python sauf qu'au lieu d'écrire **pip** on écrit **conda**.

```
via Python :      pip install <module>
via Anaconda :    conda install <module>
```

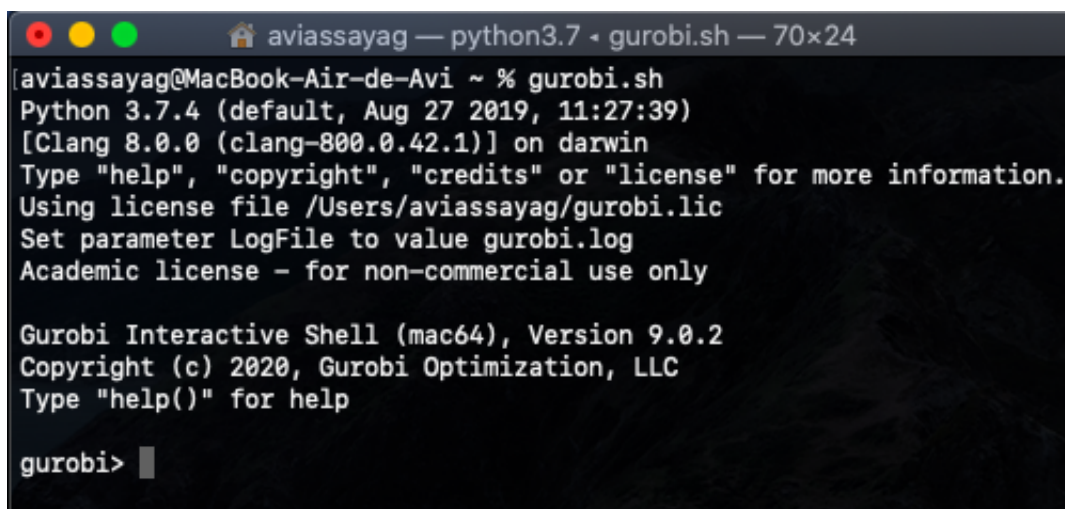
3.3.2.2 Via cette méthode, l'environnement est déjà préinstaller pour l'utilisateur et comporte une interface graphique **Spyder** ainsi qu'un éditeur de texte **Jupyter**

.Pour cela il suffira simplement de télécharger les fichiers nécessaires sur <https://www.gurobi.com/anaconda/> puis lancer Anaconda via le terminal et enfin installer le package de Gurobi.

```
python | Anaconda
conda install gurobi
```

## 3.4 Shell interactive

3.4.0.1 Lorsque que **Gurobi** mentionne son "shell intercative" il s'agit en fait d'un script (fichier ".sh") qui est fournit avec le téléchargement du solveur. En en le lançant, c'est a dire en le tapant a la console le terminal lancera une console **gurobi** ou il faudra directement écrire le code a exécuter. Ainsi un interpréteur **Gurobi** sera ouvert et attendra des instructions, au même titre qu'un interpréteur **Python**.



```
aviassayag@MacBook-Air-de-Avi ~ % gurobi.sh
Python 3.7.4 (default, Aug 27 2019, 11:27:39)
[Clang 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Using license file /Users/aviassayag/gurobi.lic
Set parameter LogFile to value gurobi.log
Academic license - for non-commercial use only

Gurobi Interactive Shell (mac64), Version 9.0.2
Copyright (c) 2020, Gurobi Optimization, LLC
Type "help()" for help

gurobi>
```

FIGURE 1 – Shell interactive Gurobi

### 3.5 Exemple Gurobi programmation linéaire (biere.py)

**3.5.0.1** Soit  $x_1$  et  $x_2$  les quantités (en volume) respectives produite pour les bières  $b_1$  et  $b_2$ . Les quantités sont soumises à des contraintes (3) pour chaque ingrédients utilisé :

**Contraintes :**

- Contrainte C1 :  $2,5 x_1 + 7,5 x_2 \leq 240$  (pour le maïs)
- Contrainte C2 :  $0,125 x_1 + 0,125 x_2 \leq 5$  (pour le houblon)
- Contrainte C3 :  $17,5 x_1 + 10 x_2 \leq 595$  (pour le malt)
- Contrainte de positivité :  $x_1, x_2 > 0$

Objectif :

- Maximiser :  $\max 15 x_1 + 25 x_2$

```
import gurobipy as gp
from gurobipy import *

try :
    # Création du model
    m = gp.Model("Biere")

    # Déclaration des Variables
    x1 = m.addVar(vtype=GRB.INTEGER, name="x1")
    x2 = m.addVar(vtype=GRB.INTEGER, name="x2")

    # Maximisation
    m.setObjective(15*x1 + 25*x2, GRB.MAXIMIZE)

    # Contraintes des Variables
    m.addConstr(2.5 * x1 + 7.5 * x2 <= 240, "c1")
    m.addConstr(0.125 * x1 + 0.125 * x2 <= 5, "c2")
    m.addConstr(17.5 * x1 + 10 * x2 <= 595, "c3")

    # Résoud la solution objective
    m.optimize()

    for v in m.getVars():
        print('%s %d' % (v.varName, v.x))

    print('Obj: %s' % m.objVal)

except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

3.5.0.2 Ci dessus le code python utilisé pour permettre a **Gurobi** de trouver la solution de maximisation, soit :  $x_1 = 12$  et  $x_2 = 28$

## 4 La théorie des jeux

4.0.0.1 Comme nous l'avons expliqué un peu plus haut, l'un des objectifs de ce stage est la modélisation d'algorithmes afin de résoudre des jeux stochastiques. Mais tout d'abord détaillons un peu le concept des **jeux**.

**Définition 1** *Un jeu est une analyse des interactions stratégiques, intégrant des contraintes (si elles existent), sur les actions des différents acteurs (joueurs) au cours du jeu.*

4.0.0.2 Pour intégrer et comprendre ce concept, il y a d'autres notions à connaître telles que **jeux statiques**, **jeux dynamiques**, **stratégie** ou encore **jeux bimatrixiel** et enfin **gains à somme nulle**.

### 4.1 Jeux statiques et dynamiques

4.1.0.1 Comme vous l'avez compris, un jeu nécessite la présence d'acteurs ; dans la suite de nos explications lorsque nous parlerons de **joueurs** nous ferons donc référence aux acteurs du jeu.

#### 4.1.1 Statiques

4.1.1.1 Lorsque l'on parle de jeu statique il s'agit en réalité d'un jeu où chaque joueur effectue une seule action en simultané de l'autre mais sans avoir accès aux informations de l'action de l'autre joueurs.

#### 4.1.2 Dynamiques

4.1.2.1 A l'inverse, un jeu dynamique est un jeu qui se déroule en plusieurs étapes et non en simultané ; c'est-à-dire que chacun des joueurs a connaissance de l'action de l'autre et donc peut établir une stratégie avant chaque futures actions.

### 4.2 Stratégies

4.2.0.1 à définir

### 4.3 Jeu bimatrixiel

4.3.0.1 Un jeu bimatrixiel se caractérise comme son nom l'indique par deux matrices. Ces dernières ne sont autres que les gains des joueurs. Autrement dit les joueurs jouent de manière simultanée et on inscrit dans une matrice leurs gains (une matrice pour chaque joueur). Voici un exemple de deux matrices de gains pour deux joueurs A et B qui jouent respectivement les lignes et les colonnes.

$$\begin{array}{cc} \text{Joueur A} & \begin{pmatrix} 3 & 3 \\ 2 & 5 \\ 0 & 6 \end{pmatrix} & \text{Joueur B} & \begin{pmatrix} 3 & 2 \\ 2 & 6 \\ 3 & 1 \end{pmatrix} \end{array}$$

## 4.4 Jeu à sommes nulles

4.4.0.1 Comme annoncé dans notre introduction nous essayerons de résoudre des jeux à sommes nulles via des algorithmes que nous allons implémenter par la suite. Mais qu'est ce qu'un jeu à somme nul ?

4.4.0.2 Un jeu a somme nul est un jeu ou le gain d'un des acteurs (J1, par exemple) représente la perte équivalente réciproque des autres acteurs ((J2, par exemple). Pour faciliter la compréhension et la résolution de ces modèles, le nombre d'acteurs autrement dit de joueurs sera fixé à 2.

4.4.0.3 Le jeu est défini par :

- Un nombre fini  $I$  d'action pour le joueur 1 (J1)
- Un nombre fini  $J$  d'action pour le joueur 2 (J2)
- Une matrice de gain  $[I \times J]$

## 5 Webographie

### Références

[Python] <https://docs.python.org/fr>

[GitHub] <https://help.github.com/en>

[Gurobi] <https://www.gurobi.com/downloads/gurobi-software/>

[Gurobi] [https://www.gurobi.com/documentation/9.0/refman/py\\_model.html](https://www.gurobi.com/documentation/9.0/refman/py_model.html)

[Théorie des Jeux] [https://fr.wikipedia.org/wiki/Th%C3%A9orie\\_des\\_jeux](https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_jeux)

[Théorie des Jeux] <http://www.cril.univ-artois.fr/~konieczny/enseignement/TheorieDesJeux.pdf>

[Théorie des Jeux] <http://www.cril.univ-artois.fr/~konieczny/enseignement/TheorieDesJeux.pdf>

## 6 Annexes

### Annexe A : Python en général

**A.0.0.1** La syntaxe est assez similaire aux autres langages puisque python utilise les mêmes types de variables, sauf les types sophistiqués. A la différence des autres langages de programmation (C, C++, Java, php) la fin d'une instruction se termine par un caractère vide et non ; , avec python c'est l'indentation qui fait office d'instruction et donc de bloc de code.

#### Annexe A.1 : Structure Conditionnelle If

**A.1.0.1** La condition est suivie par : puis vient ensuite l'instruction à effectuer, si le test est vérifié, qu'il faudra indenter (d'un cran).

```
if <condition> :  
    <instruction>
```

#### Annexe A.2 : Structure Conditionnelle Else

**A.2.0.1** La condition est suivie par : puis vient ensuite l'instruction à effectuer, si le premier test n'est pas vérifié, qu'il faudra indenter (d'un cran) au même niveau que le test If.

```
if <condition1> :  
    <instruction1>  
else :  
    <instruction2>
```

#### Annexe A.3 : Structure Conditionnelle Elif

**A.3.0.1** La condition est suivie par : puis vient ensuite l'instruction à effectuer, si le premier test n'est pas vérifié, qu'il faudra indenter (d'un cran) au même niveau que le test If.

```
if <condition1> :  
    <instruction1>  
elif <condition2> :  
    <instruction2>  
else :  
    <instruction3>
```

#### Annexe A.4 : Boucle For

**A.4.0.1** La structure est composée de for puis de deux valeurs élément et sequence qui permette de suivre l'itération à effectuer. Le bloc est exécuté autant de fois qu'il y a d'éléments dans la sequence et se termine par : .

```
for element in sequence :
```



```
<instruction>
```

## Annexe A.5 : Boucle While

A.5.0.1 La structure est composée de `while` puis de la condition qui permet d'effectuer un test. Le bloc est exécuté tant que la condition est vérifiée et se termine par :

```
while <condition> :  
    <instruction>
```

## Annexe A.6 : Les fonctions

A.6.0.1 Quant à la fonction la définition se fait de manière très simple il suffit d'utiliser le mot clé `def` et cela est terminé, en python on ne prend pas en compte le type de retour d'une fonction comme en C, C++ ou en Java (`int`, `void`, `double`, `float` etc ...).

```
def fonction (param1 , param2) :  
    <instruction1>  
    <instruction2>  
    if <test1> :  
        <instruction3>  
    else :  
        <instruction4>  
    return <instruction5>
```

## Annexe B : L'orienté objet en Python

B.0.0.1 Python est un langage résolument orienté objet, ce qui signifie que le langage tout entier est construit autour de la notion d'objets. Quasiment tous les types du langage `String` / `Integer` / `Listes` / `Dictionnaires` sont avant tout des objets tout comme les fonctions qui elles aussi sont des objets.

B.0.0.2 Pour créer une classe, donc un Objet il suffit d'utiliser le mot clé `class` suivi de : et ne pas oublier l'indentation.

```
class < NomClasse> :  
    attribut1  
    attribut2
```

B.0.0.3 Ensuite il faudra définir un constructeur qui permettra d'instancier les objets dont nous aurons besoin, il faut donc utiliser la méthode `__init__` au sein de la classe sans oublier le paramètre obligatoire (mot clé de python) `self`.

```
class < NomClasse> :  
    attribut1  
    attribut2
```

```
def __init__(self):
    self.attribut1 = ... (str)
    self.attribut2 = ... (int)
```

**B.0.0.4** Si l'on définit une classe vide c'est à dire ou pour le moment il n'y aucune action à effectuer il faut rajouter le mot clé pass

```
class < NomClasse > :
    pass
```

**B.0.0.5** Comme nous l'avons également vu une classe mère peut hériter d'une autre et donc de ses attributs et de ses méthodes. la syntaxe est simple, il suffit de mettre en parenthèse la classe mère au moment de la déclaration de la classe fille. Voici un exemple avec < NomClasse > et < NomClasse2>

```
class < NomClasse> :                                #classe mère
    attribut1
    attribut2
```

```
class < NomClasse2> (< NomClasse >) :                #classe fille
    attribut1                                         #hérité
    attribut2                                         #hérité
    attribut3
    attribut4
```

**B.0.0.6** A ce niveau on peut se demander comment Python gère ces héritages. Lorsqu'on tente d'afficher le contenu d'un attribut de données ou d'appeler une méthode depuis un objet, Python va commencer par chercher si la variable ou la fonction correspondantes se trouvent dans la classe qui a créé l'objet.

**B.0.0.7** Si c'est le cas, il va les utiliser. Si ce n'est pas le cas, il va chercher dans la classe mère de la classe de l'objet si cette classe possède une classe mère. Si il trouve ce qu'il cherche, il utilisera cette variable ou fonction.

**B.0.0.8** Si il ne trouve pas, il cherchera dans la classe mère de la classe mère si elle existe et ainsi de suite. Deux fonctions existent pour savoir si l'objet est seulement une instance d'une classe et pour savoir si la classe en question a eu recours à de l'héritage : `isinstance()` et `issubclass()`.

## Annexe C : GitHub

En résumé les commandes principales de Github sont :

- git init
- git remote add
- git clone

- git checkout
- git branch < branche > à l'inverse git branch -d < branche >
- git add < fichier > ou alors git add \* (pour tous les fichiers)
- git commit -m ...
- git push origin master ou bien git push origin < branche >
- git pull origin master ou bien git pull origin < master >
- git merge

## Annexe D : Gurobi

D.0.0.1 Voici les deux méthodes (via `gurobi.sh` ou alors via le module `gurobipy`) que l'on peut utiliser pour résoudre un programme MIP nommé "biere.py" (voir exemple [5.3]) :

```

aviassayag@MacBook-Air-de-Avi ~ % gurobi.sh biere.py
Using license file /Users/aviassayag/gurobi.lic
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)
Optimize a model with 3 rows, 2 columns and 6 nonzeros
Model fingerprint: 0x31aa2bc2
Variable types: 0 continuous, 2 integer (0 binary)
Coefficient statistics:
  Matrix range    [1e-01, 2e+01]
  Objective range [2e+01, 2e+01]
  Bounds range    [0e+00, 0e+00]
  RHS range       [5e+00, 6e+02]
Found heuristic solution: objective 510.0000000
Presolve time: 0.00s
Presolved: 3 rows, 2 columns, 6 nonzeros
Variable types: 0 continuous, 2 integer (0 binary)

Root relaxation: objective 8.800000e+02, 2 iterations, 0.00 seconds

  Nodes |   Current Node |   Objective Bounds |   Work
Expl Unexpl | Obj Depth IntInf | Incumbent BestBd Gap | It/Node Time
*  0     0       0      880.0000000  880.00000  0.00% -    0s

Explored 0 nodes (2 simplex iterations) in 0.00 seconds
Thread count was 4 (of 4 available processors)

Solution count 2: 880 510

Optimal solution found (tolerance 1.00e-04)
Best objective 8.800000000000e+02, best bound 8.800000000000e+02, gap 0.0000%
x1 12
x2 28
Obj: 880.0
aviassayag@MacBook-Air-de-Avi ~ %

```

FIGURE 2 – Résolution via shell gurobi (`gurobi.sh`)

D.0.0.2 Il est logique que le résultat produit est le même sauf la commande utilisé n'est pas la même. L'avantage de la deuxième méthode est que l'on peut importer le module `gurobipy` dans n'importe quelle future création Python.

```

aviassayag ~ - zsh — 80x36
Last login: Tue Apr 28 11:12:22 on ttys001
aviassayag@MacBook-Air-de-Avi ~ % python3 biere.py
Using license file /Users/aviassayag/gurobi.lic
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64)
Optimize a model with 3 rows, 2 columns and 6 nonzeros
Model fingerprint: 0x31aa2bc2
Variable types: 0 continuous, 2 integer (0 binary)
Coefficient statistics:
  Matrix range    [1e-01, 2e+01]
  Objective range [2e+01, 2e+01]
  Bounds range    [0e+00, 0e+00]
  RHS range       [5e+00, 6e+02]
Found heuristic solution: objective 510.0000000
Presolve time: 0.00s
Presolved: 3 rows, 2 columns, 6 nonzeros
Variable types: 0 continuous, 2 integer (0 binary)

Root relaxation: objective 8.800000e+02, 2 iterations, 0.00 seconds

  Nodes      | Current Node | Objective Bounds | Work
  Expl Unexpl | Obj  Depth IntInf | Incumbent  BestBd  Gap   It/Node Time
*    0       0 |          0    880.000000  880.00000  0.00%    -    0s

Explored 0 nodes (2 simplex iterations) in 0.00 seconds
Thread count was 4 (of 4 available processors)

Solution count 2: 880 510

Optimal solution found (tolerance 1.00e-04)
Best objective 8.800000000000e+02, best bound 8.800000000000e+02, gap 0.0000%
x1 12
x2 28
Obj: 880.0
aviassayag@MacBook-Air-de-Avi ~ %

```

FIGURE 3 – Résolution via module gurobi (gurobipy)

## Annexe E : CV

### Table des figures

1	Shell interactive Gurobi . . . . .	11
2	Résolution via shell gurobi (gurobi.sh) . . . . .	19
3	Résolution via module gurobi (gurobipy) . . . . .	20
4	Curriculum Vitae Avi ASSAYAG L3 MIAGE . . . . .	21

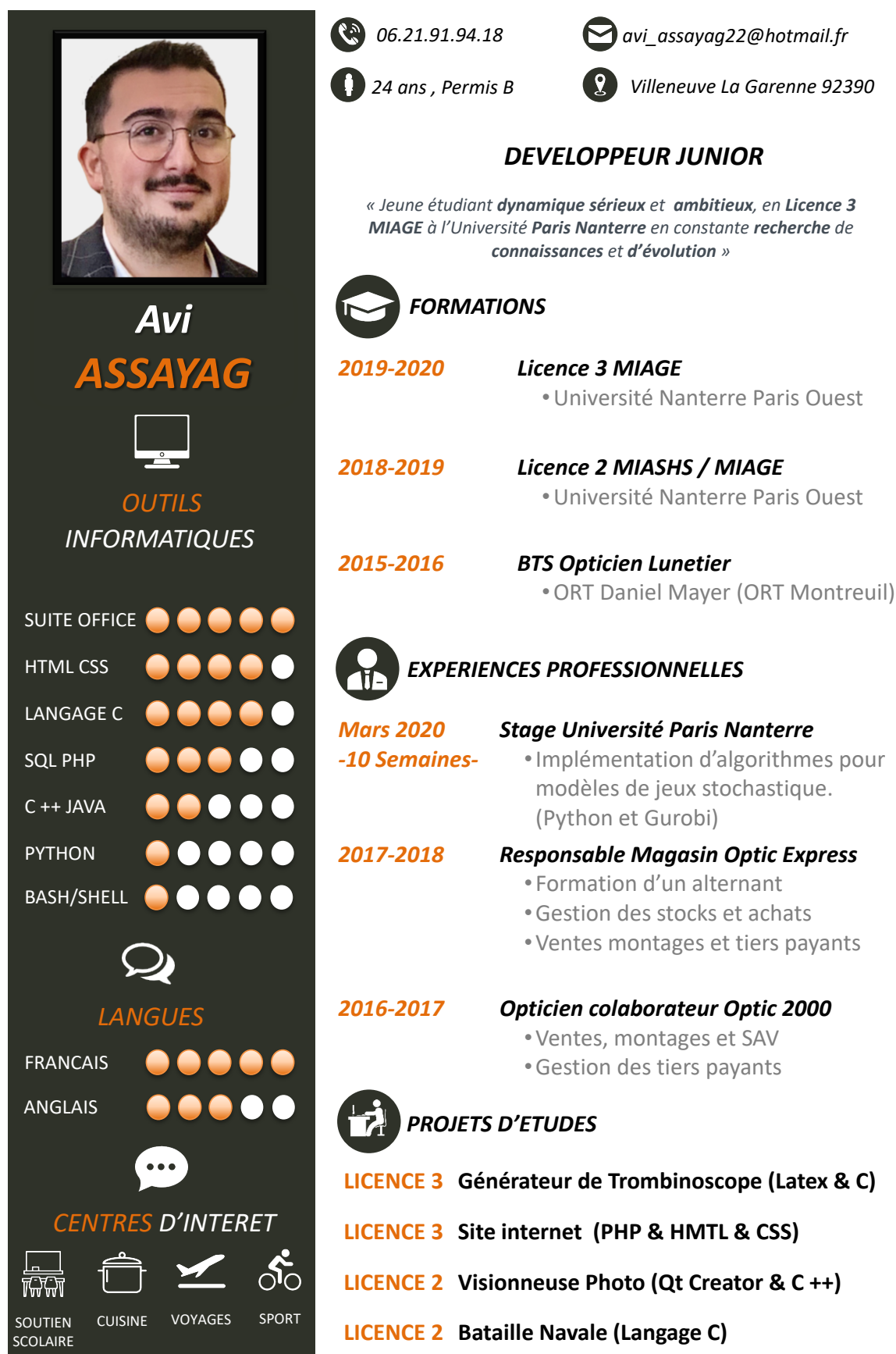


FIGURE 4 – Curriculum Vitae Avi ASSAYAG L3 MIAGE