Croupier de Blackjack markovien

Février 2022

Alexandre Kha

Table des matières

Description du code

Le code est composé de 7 fichiers Python et de 4 fichiers texte :

- Le fichier **GameObjects.py** implémente les classe Game et Record, qui permettent respectivement de concevoir le déroulement de parties de Blackjack et donc le comportement du croupier ; ainsi que d'enregistrer les informations propres aux parties, ce qui permettra d'en faire des observations.

- Le fichier **GameSimu.py** implémente une classe fille de Game : GameSimulation. Cette classe simule des parties entre un joueur virtuel et le croupier, ce qui nous permettra de générer de très nombreuses observations sans devoir jouer répétitivement.

- Le fichier **MAIN_Player_VS_Dealer.py** est le « main » qui permet de lancer des parties Jouer contre croupier. Ce « main » est surtout à but récréatif, mais il permet aussi si l'on veut de stocker les informations de nos propres parties.

- Le fichier **MAIN_AI_VS_Dealer.py** est le « main » qui lance un nombre de parties spécifié entre un joueur artificiel et le croupier. Des parties IA contre IA sont ainsi jouées. Les informations de ces parties sont stockées dans des listes (liste des stratégies du croupier, liste des Record de parties). Les Record de la partie sont ensuite converties en observations, ce qui nous sert de base pour notre étude.

- Le fichier **Simulation_Markov.py** contient des fonctions et un « main » permettant de faire des calculs et des simulations liés au Modèle de Markov du croupier.

- Le fichier **Estimation_Prediction_Amelioration.py** est le fichier phare du code. Il contient toutes les fonctions permettant de répondre aux 3 problèmes des MMC : probabilité d'une séquence relativement au modèle, prédiction de séquences en MMC (méthode Backward-Forward) et en Programmation Dynamique (algorithme de Viterbi) ; et apprentissage du MMC grâce aux simulations (algorithme de Baum-Welch). Un « main » permet de répondre à ces problèmes dans le cadre de notre MMC. Ce « main » exploite les blocs de parties dont les informations sont contenues dans les fichiers **ensemble_states.txt** et **ensemble_obs.txt**.
  Au lancement du programme, il est possible que l'apprentissage prenne au moins 10min pour s'exécuter, mais les résultats précédents seront quand même inscrits avant dans le terminal.

- Le fichier **ML_Results.txt** recense le Modèle de Markov appris au cas où l'on ne voudrait pas attendre les 10min d'apprentissage qu'entreprend le programme précédent.

- Le fichier **Model_test.py** permet de tester la vraisemblance du modèle appris sur une séquence de parties dont les états sont inscrits en binaire dans **States_temoin.txt**, et les observations dans **Observations_temoin.txt.** On teste notamment la précision des solutions en Modèle de Markov Caché et en Programmation Dynamique.

Problématique

Nous jouons au Blackjack contre un croupier virtuel. Mais ce croupier est pour le moins original : nous savons qu'il a la possibilité de tricher selon certaines stratégies, notamment en donnant des cartes défavorables au joueur, ou en tirant des cartes favorables pour lui-même.

Dans la réalité il est plutôt difficile pour un croupier de mettre en œuvre des techniques de triche au Blackjack, mais il en existe bien : faire de faux mélanges sur le jeu de cartes, marquer les cartes avec ses ongles, voir les prochaines cartes du jeu grâce à un reflet sur sa bague …

Quoiqu'il en soit, ici le croupier est une intelligence artificielle qui peut facilement tricher sans qu'on le remarque.

Ce désavantage pour le joueur est compensé par un gain de 2 jetons à chaque partie gagnée, contre la perte que d'un seul jeton en cas de partie perdue contre le croupier.

Mais ce n'est pas là le plus grand gain qu'un joueur peut espérer en jouant à ce jeu. En effet, la plateforme de jeu offre un voyage à Las Vegas à quiconque parvient à déterminer 80% des stratégies employées par le croupier sur 200 parties …

Décidés à saisir cette opportunité et dotés de bonnes connaissances en probabilités et en programmation, nous essayons de déterminer le comportement de ce croupier en le modélisant par un Modèle de Markov Caché.

Règles du Blackjack et variante présentée

Le Blackjack est un jeu de cartes qui peut se jouer à 2 joueurs au minimum (ce qui est le cas dans notre étude) : le croupier et le joueur.
Au début d'une partie, le joueur mise un certain nombre de jetons, qu'il peut gagner du croupier en cas de victoire, ou perdre en cas de défaite. Puis le croupier mélange un jeu de cartes traditionnel (52 cartes) et en distribue 2 à face découverte au joueur et 2 dont une à face cachée pour lui-même.

Le but du jeu est de tirer des cartes afin d'avoir un jeu de cartes qui se rapproche le plus de 21 (Blackjack). Cependant il convient de ne pas dépasser la valeur de 21, sinon « nous sautons »
- en anglais « the player busts »- et c'est le joueur adverse qui gagne la partie. Il convient donc de s'arrêter de tirer des cartes au bon moment, mais il faut veiller à avoir un jeu plus grand que celui de l'adverse pour le battre. En cas de jeu égal avec le croupier, c'est ce dernier qui gagne.
C'est le joueur qui commence à piocher.
Si nous ne sautons pas et que nous ne piochons plus de carte, c'est au tour du croupier de jouer. Il dévoile sa carte face cachée, et si son jeu est inférieur à celui du joueur, il peut continuer de piocher des cartes. Il a l'obligation d'en tirer tant que son jeu n'a pas atteint une valeur de 17 (valeur qui peut cependant changer suivant la région où l'on joue).

Les valeurs des cartes sont invariantes pour les cartes entre 2 et 10. Les figures ont une valeur de 10. L'As a une valeur de 1 ou de 11 selon le choix du joueur. Usuellement, elle est prise à la valeur 11 tant qu'elle ne fait pas sauter le joueur.
Normalement, les cartes tirées ne sont pas remises dans le jeu, et ce dernier n'est mélangé qu'en début de partie. Si les 52 cartes ont été tirées, le croupier remélange le jeu avec les cartes tirées pour recommencer une partie.

La variante à laquelle on joue n'a pas de mise à proprement parler, le joueur gagne 2 jetons en cas de victoire et en perd 1 en cas de défaite, mais cette donnée ne nous importera peu.
Cette variante du Blackjack possède un jeu de cartes supposé infini (donc jamais remélangé), avec les 13 cartes usuelles (As, 2, …, 10, J, Q, K) équiréparties. Ce faisant, cela peut décourager les joueurs à compter les cartes, stratégie se basant sur les cartes déjà sorties pour déterminer quand se coucher ou piocher une carte, bien que les cartes soient équiréparties (donc peu de chance de retrouver de nombreuses occurrences de cartes dans des parties successives).
Ainsi, les cartes n'ont pas besoin d'être identifiées par leur couleur (Cœur, Trèfle, Carreau et Pique), mais seulement par leur valeur.

NB : Ce jeu de Blackjack ne permet également pas de jouer sur un autre tableau, ce qui est une possibilité du jeu classique.

I. Modèle de Markov de la stratégie du croupier

Dans cette partie et dans celles d'après, nous modélisons le croupier par un automate markovien, dont les états sont ses stratégies de jeu. Puis bien que ce soit l'inconnu que nous chercherons finalement à retrouver, nous ferons des manipulations à but pédagogique avec cet automate.

1. Graphe du Modèle de Markov

Nous remarquons que les tirages du croupier ainsi que les cartes qu'il nous distribue sont pour le moins étrange … En effet, il semble qu'il puisse nous distribuer de très mauvaises cartes dans les premiers tours ou en tirer de très bonnes pour lui, à une fréquence assez régulière.

La stratégie du croupier peut prendre 4 états possibles :

- Jeu classique
- Distribution cornélienne : lors de la distribution initiale, le croupier donne une 1ère carte au joueur, puis une deuxième de sorte que la valeur de son jeu ait une valeur entre 14 et 16. Si ce n'est pas possible (par exemple la 1ère carte du joueur vaut 2 et le croupier veut lui donner un jeu de 15), le croupier lui donne automatiquement un As.
- Faire sauter le joueur au premier tour : le croupier distribue initialement au joueur des cartes de valeur 12 ou 13. Puis il le fait sauter au prochain tour si ce dernier pioche une carte, en lui donnant une carte lui faisant dépasser 21
- Blackjack instantané : le croupier se distribue lui-même initialement des cartes qui valent le Blackjack, ce qui lui fera automatiquement gagner la partie

Par la suite, nous nommerons ces états en anglais :

- **S0 / Classical**
- **S1 / Cornelian distribution**
- **S2 / Busting the player**
- **S3 / Instant Blackjack**

La matrice de transition de ce modèle, avec les états ordonnés comme précédemment cités (S0 : « Classical », S1 : « Cornelian distribution », S2 : « Busting the player », S3 : « Instant Blackjack ») s'écrit :

```
A = np.array([[0.7, 0.1, 0.1, 0.1],
              [0.5, 0, 0.2, 0.3],
              [0.5, 0.4, 0, 0.1],
              [0.6, 0.1, 0.3, 0]])
```

Ce qui peut encore se représenter par le graphe :

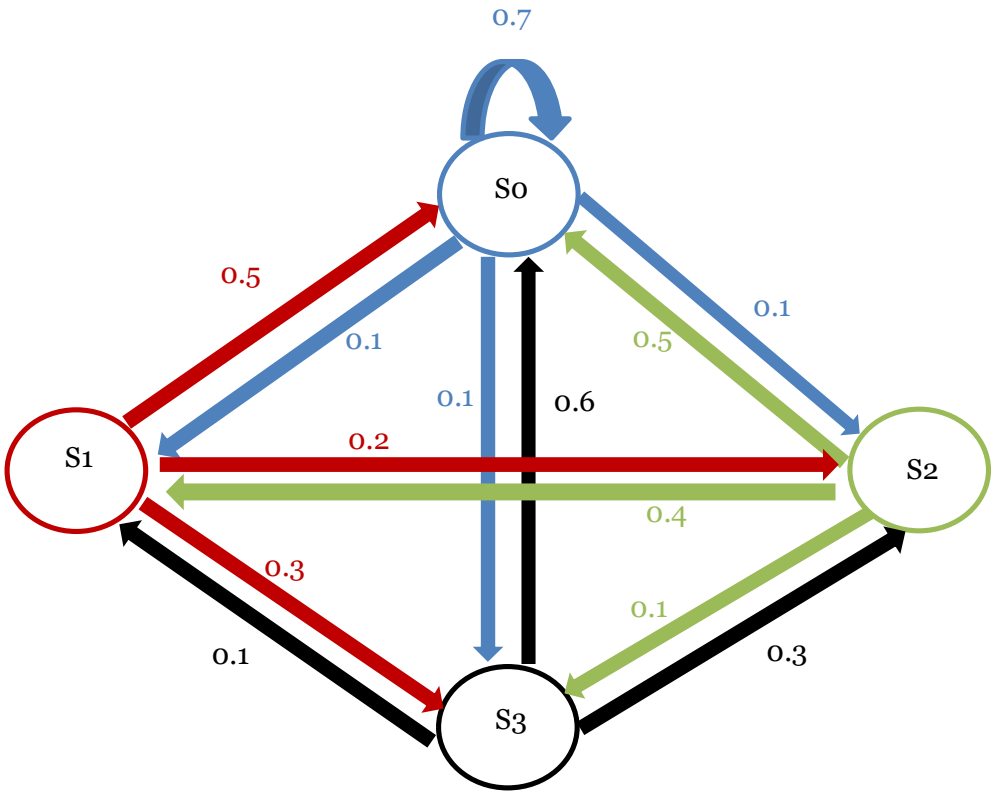

Figure 1: Graphe de transitions des états S0 (Classical), S1 (Cornelian distribution), S2 (Busting the player), S3 (Instant Blackjack)

2. Simulations du Modèle de Markov

Un des points intéressants à étudier en simulant ce modèle de Markov est la limite de la distribution des états, qui rappelons-le est définie par la relation de récurrence :

$$\pi_{n+1} = \pi_n . A$$

Or la chaîne de Markov associée au processus étudié est apériodique en raison d'une possibilité de la boucle en S0 et de plus, elle est irréductible, puisque chaque état est accessible par un autre en un nombre d'étapes fini. Cette chaîne est alors ergodique est la suite des distributions π_n converge vers une limite telle que : $\pi_l = \pi_l . A$.

Plusieurs méthodes permettent de calculer π_l.

Une première consiste à exploiter la relation de récurrence, qui permet d'expliciter le terme général de la suite comme celui d'une suite géométrique : $\pi_n = \pi_0 . A^n$.

En faisant tendre n vers l'infini, par exemple en prenant $n = 1000$, on obtient :

```
lim pi (via puissance de matrice) =
 [0.63919129 0.1244168  0.12286159 0.11353033]
```

Donc :

$$\pi_l \approx \pi_{1000} = [0.64\ 0.12\ \ 0.12\ 0.11]$$

On remarquera que l'on retrouve également le résultat connu que A^n tend vers la concaténation de ce vecteur :

```
lim A^n  =
 [[0.63919129 0.1244168  0.12286159 0.11353033]
 [0.63919129 0.1244168  0.12286159 0.11353033]
 [0.63919129 0.1244168  0.12286159 0.11353033]
 [0.63919129 0.1244168  0.12286159 0.11353033]]
```

Une deuxième méthode consiste à calculer un vecteur propre de A^T associé à la valeur 1, tout en veillant à changer son signe et à le normaliser pour bien obtenir une distribution, ce qui nous donne le même résultat :

```
lim pi (via vecteur propre) =
 [0.63919129-0.j 0.1244168 -0.j 0.12286159-0.j 0.11353033-0.j]
```

Finalement, une méthode plus empirique consiste à simuler de nombreux blocs de transitions et à relever leur état final. La loi des grands nombres permet de voir la proportion de chaque état final comme leur distribution limite. On retombe bien sur des résultats similaires :

```
Proportions en 10000 tirages =
[0.6338 0.1247 0.1235 0.118 ]
```

II. Observations retenues et Modèle de Markov Caché (MMC)

Il n'est pas possible de déterminer les stratégies du croupier directement. Cependant chaque état peut donner une ou plusieurs observations, qui renseignent donc sur ces états. Au final, c'est à partir de ces observations seulement que l'on déterminera les états pris par le croupier.

1. Observations du Modèle Caché

Les états pris par la stratégie du croupier sont observables selon la valeur du jeu des joueurs et leur disposition. On peut définir 4 observations du jeu :

- **Obs0 / Cornelian choice** : Le joueur dispose d'un jeu de valeur entre 14 et 16 à la distribution initiale
- **Obs1 / 1st Round Bust** : Le joueur saute au premier tour
- **Obs2 / Dealer's 1st Round Blackjack** : Le croupier dispose d'un Blackjack à la distribution initiale, et le joueur ne dispose pas d'un choix cornélien (mesure pour que les observations soient disjointes)
- **Obs3 / Else** : Observation autre que les précédentes

NB : Ces observations ressemblent beaucoup aux états du modèle mais n'y correspondent pas. Par exemple, nous pouvons avoir une observation de type « Cornelian choice » bien que le croupier ait joué classiquement. De même, si on se couche au premier tour, on pourrait avoir une observation de type « Else » au lieu de « 1st Round Bust », bien que la stratégie du croupier soit effectivement de nous faire sauter au premier tour.

2. Création de l'automate et génération des observations

Pour créer l'automate correspondant au croupier, nous faisons usage à de la Programmation Orientée Objet.
Nous implémentons 3 classes :

- La classe Records, qui instancie l'enregistrement des informations d'une partie : la valeur de la distribution initiale de cartes du joueur, de sa première main, de sa dernière main, du nombre de cartes tirées par le croupier, la valeur du jeu du croupier et la victoire ou non du joueur
- La classe Game qui initialise un jeu de cartes pratiquement infini, et est caractérisé par des listes contenant les stratégies prises par le croupier, les Records du jeu et les observations associées. Ces listes sont initialement vides mais se complètent au cours des parties. La classe définit également les stratégies du croupier et comment le croupier les suit (selon les probabilités de transition)
- La classe GameSimulation, classe fille de Game. Cette classe permet de simuler des parties entre le croupier et un joueur virtuel raisonnable (ne joue plus au-dessus de 16 et peut se coucher à 13 au minimum). Cela permet de générer un très grand nombre de parties et donc d'observations sans avoir à jouer manuellement avec l'automate.

NB : On peut se permettre d'utiliser des observations venant d'un joueur virtuel car elles dépendent peu de la stratégie du joueur pourvu qu'il soit raisonnable. De plus, le fait qu'on en simule plusieurs milliers permet de négliger des parties exceptionnelles (tirages risqués, se coucher malgré un faible jeu).

Nous ne nous attarderons pas sur les détails de cette implémentation puisque ce n'est pas le sujet principal de notre étude. L'automate peut toutefois être testé manuellement en lançant le fichier « MAIN_Player_VS_Dealer.py ».
On en dira toutefois quelques mots sur les points cruciaux.

Une Record d'une partie génère une observation via la méthode *observation*. La méthode vérifie les cas de figure par le biais des variables d'instance et renvoie l'observation adéquate. Par exemple si la distribution du joueur est de 15, la méthode renvoie « Cornelian choice » :

```
def observation(self):
    if ((self.player_distribution >=14) and (self.player_distribution <= 16)):
        return self.obs1
    elif ((self.player_distribution <= 13) and (self.first > 21)):
        return self.obs2
    elif ((self.last <= 21) and (self.dealer_value == 21) and (self.dealer_cards == 2)):
        return self.obs3
    else :
        return self.obs4
```

Figure 2 : Génération des observations

Pour simuler une partie, on regarde d'abord si c'est la première partie du jeu. Le cas échéant, une des stratégies et choisie aléatoirement selon la distribution initiale *pi*.
Sinon, on regarde la stratégie précédente dans la liste *AI_strat*. Puis le croupier choisit une des stratégies selon les probabilités de passage d'un état à un autre recensées dans la matrice *A* :

```
def simuplay(self):
    if len(self.AI_strat) == 0:
        strat = random.choices(self.all_strats, weights=[0.7, 0.1, 0.1, 0.1], k=1)[0]
        self.AI_strat += [strat]
        if strat == self.strat0:
            self.classical_simupick()
        elif strat == self.strat1:
            self.cornelian_simupick()
        elif strat == self.strat2:
            self.busting_simupick()
        else:
            self.evil_simupick()
    else:
        if self.AI_strat[-1] == self.strat0:
            strat = random.choices(self.all_strats, weights=self.A[0, :], k=1)[0]
        elif self.AI_strat[-1] == self.strat1:
            strat = random.choices(self.all_strats, weights=self.A[1, :], k=1)[0]
        elif self.AI_strat[-1] == self.strat2:
            strat = random.choices(self.all_strats, weights=self.A[2, :], k=1)[0]
        else:
            strat = random.choices(self.all_strats, weights=self.A[3, :], k=1)[0]
        self.AI_strat += [strat]
        if strat == self.strat0:
            self.classical_simupick()
        elif strat == self.strat1:
            self.cornelian_simupick()
        elif strat == self.strat2:
            self.busting_simupick()
        else:
            self.evil_simupick()
```

Figure 3 : Code de la fonction de simulation d'une partie entre le croupier et un joueur virtuel

Pour simuler un certain nombre de parties, on appelle la fonction précédente plusieurs fois jusqu'à atteindre ce seuil, tout en veillant à enregistrer les observations correspondantes dans la liste *obs* :

```
def complete_simugame(self, nb_games):
    count_games = 0
    while count_games <nb_games:
        self.simuplay()
        self.obs += [self.records[-1].observation()]
        count_games += 1
    print("END OF THE GAME")
```

Figure 4 : Code de la fonction exécutant un bloc de simulations

Finalement, nous devons définir les probabilités de la matrice d'émission B, c'est-à-dire les probabilités qu'un état S_i induise une observation O_j : $P(Y = O_j \mid X = S_i)$.
Pour ce faire, nous parcourons la liste des stratégies employées par le croupier et on regarde l'observation correspondante. Dans un dictionnaire on inscrit la proportion d'occurrences de chaque observation pour chaque stratégie.
Par exemple à la stratégie « Classical » on associe le dictionnaire {obs1 : 0, …, obs3 :0}.
Si « Classical » survient et donne l'observation obs1, on incrémente le compteur de obs1 dans le dictionnaire : $\{obs1 : 0, \dots, obs3 : 0\} \rightarrow \{obs1 : 1, \dots, obs3 : 0\}$. Et ainsi de suite.
Au final on normalise les compteurs pour obtenir des proportions :

```
def proportions(self):
    obs1 = "Cornelian choice"
    obs2 = "1st Round Bust"
    obs3 = "Dealer's 1st Round Blackjack"
    obs4 = "Else"
    classic = {obs1 : 0, obs2 : 0, obs3 : 0, obs4 : 0}
    cornelian = {obs1 : 0, obs2 : 0, obs3 : 0, obs4 : 0}
    buster = {obs1 : 0, obs2 : 0, obs3 : 0, obs4 : 0}
    blackjack = {obs1 : 0, obs2 : 0, obs3 : 0, obs4 : 0}
    tot_cl = 0
    tot_cor = 0
    tot_bust = 0
    tot_bl = 0
    for i in range(len(self.AI_strat)):
        if self.AI_strat[i]== self.strat0:
            tot_cl +=1
            if self.obs[i] == obs1:
                classic[obs1]+=1
            elif self.obs[i] == obs2:
                classic[obs2]+=1
            elif self.obs[i] == obs3:
                classic[obs3]+=1
            else:
                classic[obs4]+=1
        elif self.AI_strat[i]== self.strat1:
```

Figure 5 : Code partiel de la fonction calculant les proportions d'observations

Sur un très grand nombre de parties, la Loi des grands nombres permet de voir ces proportions comme des probabilités. Au bout de 20 000 parties, on obtient les probabilités d'émission :

```
prop_classic = {'Cornelian choice': 0.2512311420307981, '1st Round Bust': 0.06097084343000078, "Dealer's 1st Round Blackjack": 0.02986007973110295, 'Else': 0.6579379348080983}
prop_cornelian = {'Cornelian choice': 0.9299674267100978, '1st Round Bust': 0.0, "Dealer's 1st Round Blackjack": 0.004071661237785016, 'Else': 0.06596091205211727}
prop_busting = {'Cornelian choice': 0.0, '1st Round Bust': 0.8711507293354943, "Dealer's 1st Round Blackjack": 0.006077795786061588, 'Else': 0.12277147487844407}
prop_blackjack = {'Cornelian choice': 0.24441524310118265, '1st Round Bust': 0.0, "Dealer's 1st Round Blackjack": 0.668418747262374, 'Else': 0.08716600963644328}
```

Figure 6 : Proportions des observations selon chaque stratégie

Ou encore en arrondissant au millième :

```
B = np.array([[0.251, 0.061, 0.030, 0.658],
              [0.930, 0, 0.004, 0.066],
              [0, 0.871, 0.006, 0.123],
              [0.244, 0, 0.669, 0.087]])
```

3. Représentation partielle du MMC

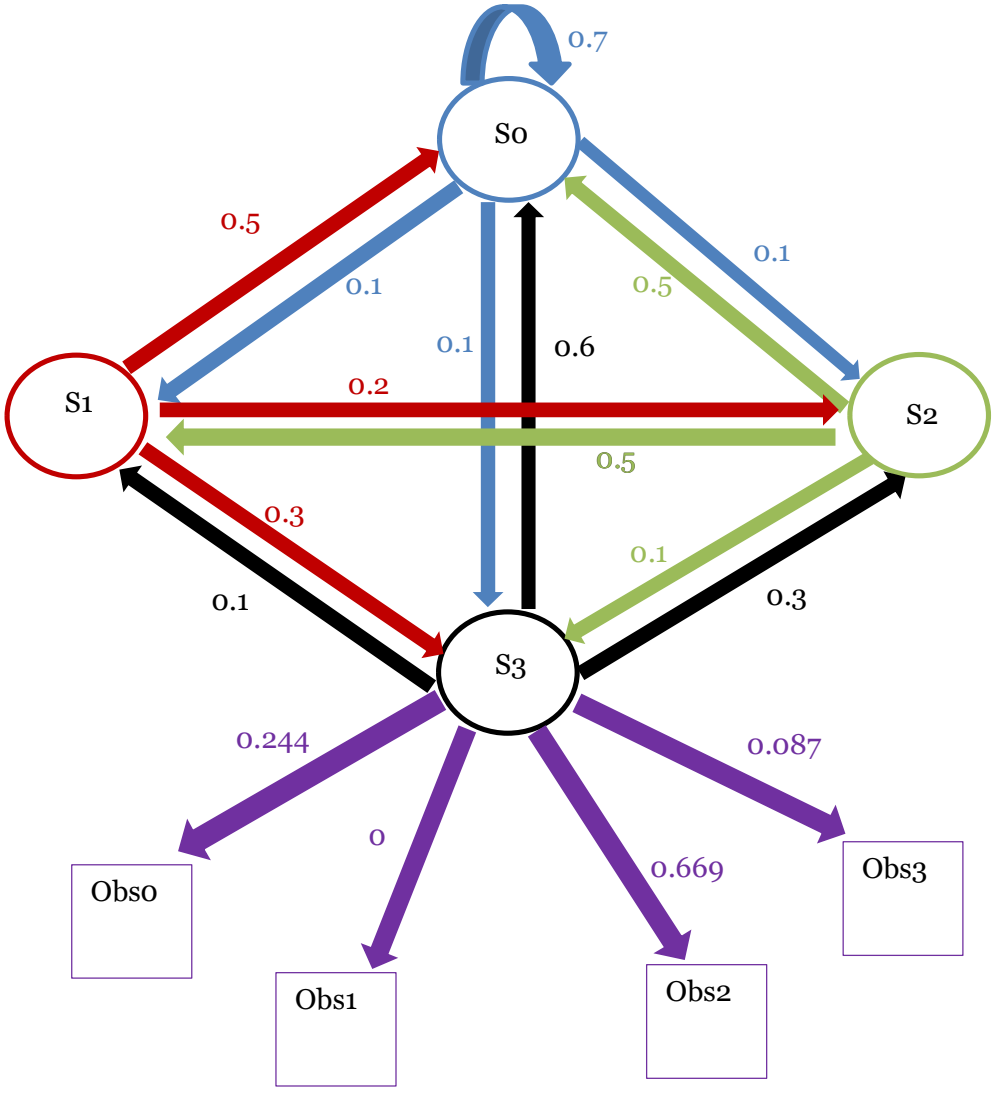

Figure 7 : Graphe partiel du MMC, avec émission des 4 observations par S3

NB : Dans le cadre de notre étude, nous n'avons pas besoin de générer artificiellement les observations à partir des états et des probabilités d'émission. C'est l'automate lui-même qui génère les états et recense les observations du jeu. C'est seulement après que l'on parvient à calculer les probabilités d'émission.

III. Réponses aux 3 problèmes soulevés par le MMC

Dans cette partie dédiée à la résolution de notre problème initial, nous détaillerons peu les algorithmes mobilisés, mais plutôt les méthodes ainsi que les résultats obtenus.

1. Probabilité d'une séquence d'observations relativement au modèle

Il est possible de calculer la probabilité d'obtenir une séquence d'observations sachant le modèle via un algorithme de type Backward-Forward.
A partir d'une séquence de 200 observations issue d'une simulation de 200 parties, nous obtenons le résultat suivant :

```
pY_knowing_lambda =  6.562888445242412e-103
```

C'est une probabilité extrêmement faible dont il est finalement assez difficile de connaître le poids.

2. Prédiction de la séquence d'états à partir de la séquence d'observations : solution en Modèle de Markov Caché et en Programmation Dynamique

Le deuxième grand problème soulevé par les MMC est de prédire une séquence d'états à partir de la séquence d'observations correspondante.
Il existe 2 approches pour résoudre ce problème :
L'approche en MMC, qui consiste à déterminer l'état de probabilité maximale à chaque étape de la séquence, puis l'approche en Programmation Dynamique (algorithme de Viterbi) qui chercherait plutôt à trouver la séquence de probabilité maximale parmi toutes les autres.
Sur la même séquence de 200 observations, nous obtenons des résultats probants en comparant la séquence d'états réels et les séquences prédites en MMC et PD :

```
Proportion de vraisemblance (HMM) avec la simulation réelle :  0.805
Proportion de vraisemblance (DP) avec la simulation réelle :  0.725
```

On remarquera que la solution en MMC est légèrement plus précise que la solution en PD ici, mais ce n'est pas forcément toujours le cas.
Ci-dessous est représenté le début de comparaison entre les 3 séquences :

```
Matrice de comparaison via HMM et DP
 [['Instant Blackjack' 'Instant Blackjack' 'Instant Blackjack']
 ['Busting the player' 'Busting the player' 'Busting the player']
 ['Cornelian distribution' 'Cornelian distribution'
  'Cornelian distribution']
 ['Classical' 'Classical' 'Classical']
 ['Instant Blackjack' 'Classical' 'Classical']
 ['Busting the player' 'Classical' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Busting the player' 'Busting the player' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Busting the player' 'Busting the player' 'Classical']
 ['Cornelian distribution' 'Cornelian distribution' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Busting the player' 'Busting the player' 'Classical']
 ['Cornelian distribution' 'Cornelian distribution' 'Classical']
 ['Busting the player' 'Classical' 'Classical']
 ['Cornelian distribution' 'Cornelian distribution' 'Classical']
 ['Busting the player' 'Busting the player' 'Classical']
 ['Cornelian distribution' 'Cornelian distribution' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Classical' 'Classical' 'Classical']
 ['Busting the player' 'Busting the player' 'Classical']
 ['Classical' 'Cornelian distribution' 'Classical']
```

Figure 8 : Comparaison (sur une portion) entre à gauche la séquence réelle, au milieu la séquence prédite en MMC, et à droite la séquence prédite en PD

3. Machine Learning : Retrouver le modèle de base à partir des observations

Le 3e problème des MMC est celui qui nous intéresse le plus ici : il s'agit d'apprendre le modèle de Markov (la matrice de transition A, la matrice d'émission B et la distribution initiale π) à partir de plusieurs séquences d'observations et d'approximations grossières de A, B et π. L'algorithme que nous utilisons est l'algorithme de Baum-Welch, se basant sur les calculs de gammas.

Avec 10 séquences de 200 observations enregistrées dans un fichier binaire « ensemble_obs.txt », ainsi que les matrices d'approximation :

```
A_approx = np.array([[0.8, 0.05, 0.1, 0.05],
                     [0.6, 0, 0.2, 0.2],
                     [0.6, 0.2, 0, 0.2],
                     [0.7, 0.1, 0.2, 0]])

B_approx = np.array([[0.15, 0.13, 0.02, 0.70],
                     [0.9, 0, 0.01, 0.09],
                     [0, 0.8, 0.1, 0.1],
                     [0.15, 0, 0.75, 0.1]])

pi_0 = np.array([[0.8, 0.08, 0.05, 0.07]])
```

Nous obtenons le modèle appris :

```
pi_mod = np.array([[1, 0, 0, 0]])

A_mod = np.array([[0.754, 0.077, 0.078, 0.091],
                  [0.442, 0, 0.230, 0.328],
                  [0.500, 0.331, 0, 0.169],
                  [0.662, 0.023, 0.315, 0]])

B_mod = np.array([[0.238, 0.059, 0.023, 0.680],
                  [0.853, 0, 0.002, 0.145],
                  [0, 0.861, 0.023, 0.116],
                  [0.121, 0, 0.735, 0.144]])
```

Nous sentons que le modèle appris converge vers le modèle réel bien qu'il existe encore quelques différences, qui pourraient être atténuées avec plus d'entraînement. On évitera toutefois ici d'entraîner le modèle avec plus de séquences, sachant qu'il faut déjà une vingtaine de minutes pour qu'il intègre 20 séquences de 200 observations. Ou alors on le fera sur un ordinateur performant. Malgré les différences soulevées précédemment, nous allons tester ce modèle sur une nouvelle séquence de parties, pour voir s'il reste vraisemblable.

IV. Test du modèle obtenu par Machine Learning

Maintenant dotés d'un modèle appris à partir d'observations seulement, nous devons maintenant le tester sur de nouvelles parties pour savoir à quel point le modèle est fiable (et si nous pouvons bien remporter un voyage à Las Vegas).

On génère 200 nouvelles parties dont la séquence d'états est stockée en binaire dans le fichier « States_temoin.txt » et la séquence d'observations correspondantes dans le fichier « Observations_temoin.txt ».

Nous prenons les matrices A, B et π qui proviennent du modèle appris par l'algorithme de Baum-Welch, contrairement à précédemment où on avait pris les matrices du modèle réel.

Nous calculons alors la matrice de différences ainsi que les proportions de vraisemblance entre les prédictions (MMC et PD) et la séquence réelle :

```
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Instant Blackjack' 'Instant Blackjack' 'Classical']
['Classical' 'Classical' 'Classical']
['Instant Blackjack' 'Instant Blackjack' 'Classical']
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Instant Blackjack' 'Instant Blackjack' 'Classical']
['Busting the player' 'Busting the player' 'Classical']
['Classical' 'Classical' 'Classical']
['Instant Blackjack' 'Instant Blackjack' 'Instant Blackjack']
['Busting the player' 'Busting the player' 'Busting the player']
['Cornelian distribution' 'Cornelian distribution'
 'Cornelian distribution']
['Instant Blackjack' 'Instant Blackjack' 'Instant Blackjack']
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Busting the player' 'Busting the player' 'Classical']
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Classical' 'Instant Blackjack' 'Classical']
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Classical' 'Classical' 'Classical']
['Cornelian distribution' 'Cornelian distribution'
 'Cornelian distribution']
```

Figure 9 : Début de comparaison entre la séquence de stratégies réelle (gauche) et les séquences prédites via MMC (milieu) et PD (droite) à partir du modèle appris

```
Proportion de vraisemblance (HMM) avec la simulation réelle :  0.82
Proportion de vraisemblance (DP) avec la simulation réelle :  0.705
```

Figure 10 : Proportion de vraisemblance entre la séquence réelle et les séquences prédites à partir du modèle appris

Dans ce cas-là, nous avons prédit en MMC une séquence vraisemblable à la séquence réelle à 82%. La solution en PD est considérablement moins précise sans être catastrophique, en étant vraisemblable à 71% à la séquence réelle.

Puisque l'on a réussi une prédiction correcte à plus de 80%, nous remportons un séjour à Las Vegas ! Espérons que nos connaissances en probabilités nous seront également utiles pour jouer à de vraies tables de Blackjack là-bas …

Conclusion

Nous récapitulons la démarche minimale qui répond à notre problème initial :

1) Nous observons une vingtaine de séquences de 200 parties avec le croupier
2) Grâce à l'algorithme de Baum-Welch, nous déterminons un MMC appris à partir de ces séquences
3) Nous appliquons le modèle appris à une nouvelle séquence d'observations pour en prédire les séquences de stratégies correspondantes en MMC et PD. Puis nous les comparons avec les stratégies réelles du croupier

La modélisation du problème en Modèle de Markov Caché est puissante, car elle permet de répondre à des problèmes non triviaux, comme la prédiction de séquence d'états et l'apprentissage d'un modèle de Markov à partir seulement des observations.

On notera cependant qu'il faut une grande puissance de calcul pour répondre à ces problèmes : nécessité de nombreuses longues séquences d'observations pour le Machine Learning et difficultés à calculer les faibles probabilités lorsque l'on prédit de longues séquences.

Des méthodes de re-scaling peuvent être implémentées pour gérer les très petites probabilités mises en jeu. Prendre le logarithme des variables peut également être utile, ce qui est déjà fait dans notre implémentation de l'algorithme de Viterbi.

Les prédictions en MMC sont plutôt fiables, à un seuil de 80% ici, pour des séquences d'au moins 200 observations. Les prédictions en PD sont moins fiables mais proposent une approche intéressante qui mérite d'être étudiée et qui peut servir dans d'autres types de problèmes.

On remarquera que l'une des sources d'erreur des prédictions vient en fait des observations, qui ne renseignent pas assez finement sur les états (malgré leurs ressemblances).

En effet, puisque les observations sont disjointes, au lieu d'observer un Blackjack du croupier, on peut très bien observer une distribution cornélienne si notre jeu initial vaut entre 14 et 16. Cela fait donc diminuer la probabilité d'observer le Blackjack du croupier bien que ce soit effectivement sa stratégie, ce qui a pour conséquence de faire diminuer la fiabilité des prédictions. Pour tout MMC, il faudrait alors trouver des observations qui seraient capables d'indiquer des états avec une probabilité plutôt extrême (par exemple 0.1, 0.2, 0.8 ou 0.9) que neutre (par exemple 0.3, 0.4, 0.5, 0.6), si on veut espérer faires des prédictions justes.