École Centrale Lyon

Rapport

Projet de recherche - IA et jeux

Élèves :
Oussama LAFDIL
Zakaria ABOU-ZHAR

Enseignant :
Phillipe Michel

23 juin 2023

Table des matières

1 Introduction

L'intelligence artificielle a ouvert de nouvelles perspectives dans de nombreux domaines, notamment dans le domaine des jeux. En effet, les jeux à information imparfaite, en particulier le Poker, est un sujet d'étude populaire dans le domaine de l'intelligence artificielle depuis de nombreuses années.

Dans ce contexte, le rapport suivant explore l'utilisation de l'algorithme CFR (Counterfactual Regret Minimization) pour résoudre le Kuhn poker. Nous allons présenter d'abord un contexte qui comporte des notions et des théorèmes importants utilisés dans la théorie des jeux, ainsi que des informations générales sur le poker et sur l'histoire des robots de poker. Nous allons ensuite aborder la méthodologie utilisée et donner le principe de l'algorithme CFR, puis fournir un pseudocode et expliquer comment il peut être appliqué au Kuhn poker. Nous présenterons ensuite les résultats obtenus par l'application de l'algorithme sur le jeu de Kuhn poker à deux joueurs et à plusieurs joueurs. Nous allons terminer par une section qui présente les limites et les perspectives, suivie d'une conclusion.

Dans l'ensemble, ce rapport fournit une vue d'ensemble de l'utilisation de l'algorithme CFR pour résoudre le Kuhn poker.

2 Background

2.1 Théorie des jeux

La théorie des jeux est une discipline mathématique qui permet d'analyser de manière formelle des situations où des acteurs, ou joueurs, agents ou preneurs de décision, interagissent. Un jeu est alors considéré comme un univers dans lequel chaque preneur de décision a un ensemble d'actions possibles déterminé par les règles du jeu. Ce domaine cherche à comprendre comment les agents prennent des décisions en tenant compte des actions et des choix des autres participants, ainsi que des conséquences possibles de ces décisions. Elle examine les différentes stratégies que les joueurs peuvent adopter et les résultats qui peuvent découler de ces interactions stratégiques. Cette discipline a de nombreuses applications qui permettent de comprendre divers phénomènes économiques, politiques ou biologiques : Elle est utilisée pour analyser les marchés concurrentiels, les négociations politiques, l'évolution des espèces, les conflits politiques, la prise de décision en entreprise. [1]

2.1.1 Notations

- L'ensemble des joueurs, chaque joueur est indéxé par i :

$$N = \{1, 2, \ldots, n\}$$

- Pour chaque joueur i, un ensemble de stratégies :

$$S_i = \{s_1, s_2, \ldots, s_{n_i}\}$$

- La fonction utilité pour le joueur i :

$$u_i : S_1 \times S_2 \times \ldots \times S_n \rightarrow u(i)$$

2.1.2 Modélisation et classification des jeux

La théorie des jeux classifie les jeux en plusieurs catégories :

Jeux simultanés :

Dans un jeu simultané, les joueurs choisissent leurs décisions en même temps, sans avoir connaissance des décisions choisies par les autres joueurs. Cela signifie qu'ils doivent prendre leur décision sans connaître le choix des autres joueurs.
Un jeu simultané à deux joueurs avec des ensembles de stratégies finis est souvent modélisé sous forme normale ou stratégique par un tableau, ou une matrice de gain, dont les lignes sont les stratégies du joueur 1, les colonnes sont les stratégies du joueur 2 et chaque case représente le gain des deux joueurs. [1]
Exemple : Pierre, feuille, ciseaux :

P1/P2	P	F	C
P	0,0	0,1	1,0
F	1,0	0,0	0,1
C	0,1	1,0	0,0

TABLE 1 – Tableau de gain.

Jeux séquentiels :

Dans un jeu séquentiel, les joueurs choisissent leurs décisions ou stratégies de manière séquentielle, en tenant compte des actions prises précédemment par les autres joueurs. En d'autres termes, les joueurs prennent des décisions dans un ordre spécifique. Parmi les moyens de modélisation et de représentation des jeux séquentiels, nous trouvons la modélisation sous forme extensive. En effet, c'est un modèle où les agents choisissent séquentiellement leurs actions, jusqu'au moment où le jeu est déclaré fini et le paiement ou les gains sont donnés. [2]

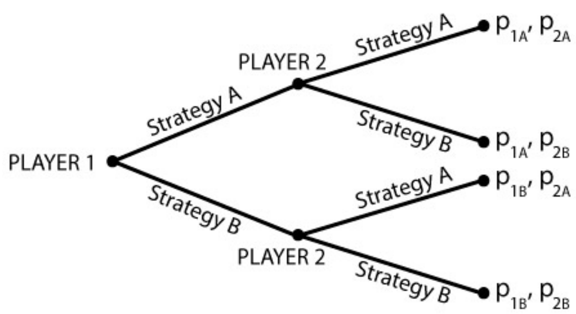


FIGURE 1 – Jeu sou forme extensive

Jeux à information complète et jeux à information incomplète :

Dans un jeu à information complète, chaque joueur possède une connaissance parfaite et complète de toutes les informations importantes concernant le jeu. En effet, chaque joueur connaît les actions passées, les choix disponibles, les préférences des autres joueurs et les résultats de toutes les actions possibles.
Contrairement aux jeux à information complète, dans un jeu à information incomplète, les joueurs n'ont pas une connaissance parfaite de toutes les informations importantes sur le jeu. [1]

Jeux à information parfaite et jeux à information imparfaite :

Dans un jeu à information imparfaite, les joueurs ont une connaissance limitée ou asymétrique des informations pertinentes sur le jeu. Contrairement aux jeux à information parfaite, où chaque joueur a une connaissance complète de toutes les informations, les jeux à information imparfaite impliquent une asymétrie d'information et une incertitude quant aux actions et quant aux préférences des autres joueurs.
Un bon exemple de jeux à information imparfaite est le poker ou tout jeu de cartes où les cartes de chaque joueur sont cachées aux autres joueurs. [1]

Jeux à somme nulle et jeux à somme non nulle :

Un jeu à somme nulle en théorie des jeux est un type de jeu dans lequel la somme des gains des joueurs est toujours nulle, c'est-à-dire que ce qui est gagné par un joueur est exactement ce qui est perdu par un autre joueur. Dans un jeu à somme nulle, les intérêts des joueurs sont directement opposés, et les gains d'un joueur sont symétriques aux pertes des autres joueurs.
Exemple : Poker, Tarot
Un jeu à somme non nulle est un type de jeu où la somme des gains des joueurs peut varier et n'est pas nécessairement égale à zéro. Contrairement aux jeux à somme nulle où les intérêts des joueurs sont directement opposés, dans un jeu à somme non nulle, il est possible pour les joueurs de réaliser des gains mutuellement bénéfiques.
Exemple : Le dilemme du prisonnier

2.1.3 Equilibre de Nash

l'équilibre de Nash est présenté comme une situation où chaque agent adopte la meilleure stratégie compte tenu du choix de stratégie des autres joueurs. C'est donc tel qu'aucun joueur ne regrette son choix au vu du choix des autres. L'équilibre de Nash traduit mathématiquement la notion de stabilité.

Dans l'ensemble, un individu ne peut tirer aucun avantage supplémentaire d'un changement d'action, en supposant que les autres joueurs maintiennent leurs stratégies constantes. Un jeu peut donc avoir plusieurs équilibres de Nash ou n'en avoir aucun.

2.1.4 Ensemble d'information

Un ensemble d'information est un ensemble de noeuds indiscernables pour le joueur qui doit prendre une décision. En outre, en deux nœuds qui appartiennent au même ensemble d'information, les actions possibles doivent être les mêmes, sinon les nœuds ne seraient pas indiscernables. [1]

2.1.5 Théorème de Nash

Enoncé :

Tout jeu fini en stratégie mixtes admet un équilibre de Nash. [6]

2.1.6 Théorème de minimax de John von Neumann

Le théroème de minimax de John von Neumann (ou le théorème fondamental de la théorie des jeux à deux joueurs) est un théorème qui garantit qu'en cas de jeu non coopératif à information complète et à somme nulle, et dans lequel deux joueurs s'opposent avec chacun un nombre fini de stratégies pures , il existe au moins une situation de jeu stable, à savoir une situation dans laquelle aucun des deux joueurs n'a intérêt à changer sa stratégie. [5]

2.2 Poker

Le poker est une catégorie de jeux de cartes multijoueurs qui comporte de nombreuses variantes. Ces variantes présentent plusieurs similitudes : elles utilisent un jeu de cartes standard (4 couleurs de 13 rangs), il y a des tours d'enchères et le système standard de classement des mains à cinq cartes est utilisé pour déterminer les gagnants. Parmi les variantes de poker les plus populaires, nous trouvons : [7]

Texas hold'em :

Le Texas hold'em est une variante du poker dans laquelle les joueurs cherchent à créer la meilleure combinaison de 5 cartes en utilisant les cartes privées que possède le joueur (2cartes) et les cartes communes (5cartes). Les mises ont lieu avant et après chaque distribution de cartes, avec des options telles que checker, suivre, relancer ou se coucher. Le joueur ayant la main la plus forte après tous les tours d'enchères emporte le pot.

Double hold'em :

Le Double Hold'em est une variante du Texas Hold'em dans laquelle le croupier ou le dealer, crée deux lignes de cartes ouvertes. Les joueurs reçoivent deux cartes fermées et peuvent choisir 0, 1 ou 2 cartes de leur main pour former la meilleure combinaison avec chaque ligne. Dans le Double Hold'em, chaque ligne représente la moitié du pot et le showdown suit les mêmes règles que le Hold'em classique.

Omaha hold'em :

L'Omaha hold 'em est variante de poker dans lequel les joueurs reçoivent quatre cartes privés au lieu de deux cartes et utilisent deux d'entre elles ainsi que trois cartes communes pour former la meilleure main. Le nom "Omaha" souligne l'obligation d'utiliser précisément deux cartes privées dans toutes les variantes du jeu.

Stud poker :

Le stud poker était populaire avant que le hold'em ne le devienne, ce jeu se caractérise par l'absence de cartes communes. Il comprend des variantes telles que le stud à 5 cartes, à 6 cartes et à 7 cartes. Les joueurs cherchent à former la meilleure combinaison de 5 cartes avec leurs cartes fermées et visibles, en suivant une séquence de jeu spécifique comprenant des tours d'enchères et la distribution de cartes ouvertes supplémentaires.

2.3 Kuhn Poker

Le Kuhn Poker est une jeu de poker simple à 3 cartes créé par Harold E. Kuhn. C'est un jeu à somme nulle à information imparfaite impliquant deux joueurs. Dans le Kuhn Poke, Le jeu se déroule en un seul tour d'enchères.
Les règles du jeu sont les suivantes :

Deux joueurs misent chacun 1 jeton avant la distribution des cartes. Trois cartes (Roi, Reine, Valet) sont mélangées, une carte est distribuée à chaque joueur et conservée comme information privée. Lors d'un tour, un joueur peut soit passer, soit parier. Un joueur qui mise place un jeton supplémentaire dans le pot. Lorsqu'un joueur se couche après que l'autre joueur a parié, l'adversaire prend tous les jetons du pot. Lorsqu'il y a deux passes successives ou deux mises successives, les deux joueurs révèlent leurs cartes et le joueur ayant la carte la plus élevée prend tous les jetons du pot.

Arbre de jeu du Kuhn poker :
La représentation sous forme extensive du jeu Kuhn Poker est la suivante :

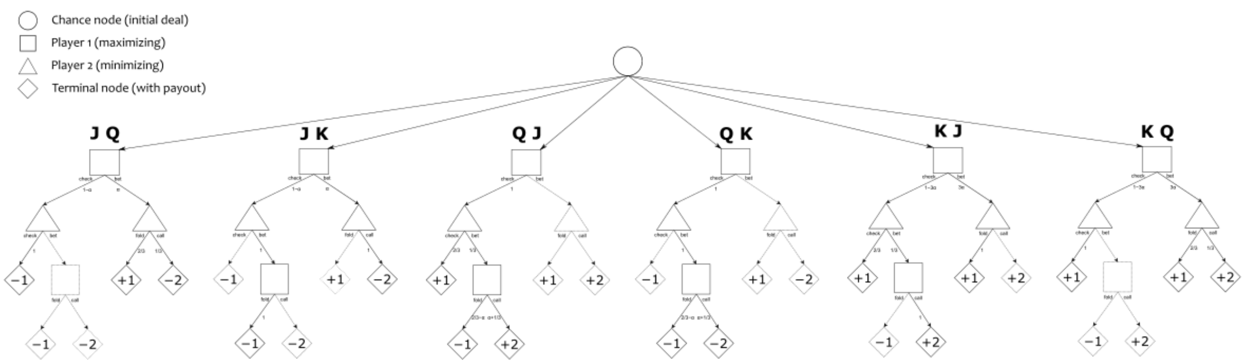


FIGURE 2 – Arbre de jeu du Kuhn Poker

Une autre variante du Kuhn poker pour trois joueurs a été introduite en 2010 par Nick Abou Risk et Duane Szafron. Dans cette version, le jeu comprend quatre cartes, trois cartes sont distribuées aux joueurs. Le système d'enchère est le même que celui de la version de deux joueurs. [8]

2.4 Historique des poker bots

Pluribus poker bot

Pluribus est un bot IA créé par le laboratoire d'IA de Facebook et l'université Carnegie Mellon en 2019. Il a été développé pour jouer au Texas hold'em et est connu pour être le premier robot IA à vaincre des joueurs humains lors d'une compétition de poker multijoueurs. Le développement de Pluribus visait à créer une IA surhumaine pour le poker multijoueur, ce qui a été considéré comme une étape majeure dans le domaine du poker informatique.
Les joueurs experts qui ont joué contre Pluribus ont exprimé qu'il était difficile de prédire ses mains. [7]

Claudico poker bot

Claudico est un robot joueur de poker créé par le professeur Tuomas Sandholm et ses étudiants de l'université Carnegie Mellon. Le nom Claudico signifie "I limp" en latin, ce qui fait référence à la stratégie du robot qui consiste à "call" dans une main sans relancer. Au lieu de s'appuyer sur l'expertise d'un joueur de poker professionnel pour guider sa stratégie, l'équipe a demandé à l'ordinateur de concevoir sa propre stratégie optimale.

Une version de Claudico a remporté un tournoi contre d'autres programmes informatiques en juillet 2014. Cependant, une version améliorée appelée Libratus a été développée pour succéder à Claudico. Comme Claudico, Libratus est conçu pour se mesurer aux meilleurs joueurs humains de poker. [7]

Cepheus poker bot

Cepheus est un programme de jeu de poker développé par le groupe de recherche Computer Poker Research Group de l'Université d'Alberta. Il a franchi une étape importante en "résolvant" le jeu du Texas hold'em en tête-à-tête. C'est la première fois qu'un jeu à information imparfaite joué en compétition par des humains est résolu.

La stratégie de Cepheus est très proche d'une stratégie d'équilibre de Nash pour le Texas hold'em en tête-à-tête. En d'autres termes, il joue de manière optimale en réponse à toute contre-stratégie. [7]

3 Méthodologie

3.1 Définition et principe de l'algorithme : Counterfactual Regret Minimization (CFR)

Principe :

L'algorithme Counterfactual Regret Minimization (CFR) est un algorithme utilisée pour résoudre des jeux à information imparfaite en améliorant progressivement les stratégies. Il commence par sélectionner une action à prendre, calculer la récompense associée à cette action, puis calculer les "counterfactual" récomponses basées sur les autres actions possibles. En soustrayant ces "counterfactual" récompenses de la récompense réelle, le regret est calculé. Les regrets sont ensuite stockés dans une table et cumulés au fil des itérations. Ensuite, les regrets sont sommés et normalisés pour obtenir une stratégie. [3] [4]

Définitions : [4]

- Soit A l'ensemble de toutes les actions du jeu.
- Soit I un ensemble d'informations.
- Soit A(I) l'ensemble des actions possibles pour l'ensemble d'informations I.
- Soit t et T des pas de temps.
- Soit σ^t le profil des stratégies formé par l'ensemble de stratégies des joueurs.
- Soit h est une séquence d'actions (y compris les actions aléatoires).
- Soit $\pi^\sigma(h)$ la probabilité (reach probability) d'atteindre l'historique du jeu h avec le profil de stratégie σ.
- Soit $\pi(I) = \sum_{h \in I} \pi(h)$ la probabilité d'atteindre l'ensemble d'information I avec le profil de stratégie
- Soit $u_i(z)$ l'utilité pour le joueur i après avoir atteint le passé final z (terminal history).
- Soit $v_i(\sigma, h) = \sum_{z \in Z, h \subseteq z} \pi_i(h)\pi(h, z)u_i(z)$ la valeur "counterfactual" au passé non final
- Soit $r(h, a) = v_i(I \to a, h) - v_i(\sigma, h)$ le regret "counterfactuel" de ne pas prendre l'action **a** au moment du passé h.
- Soit $r(I, a) = \sum_{h \in I} r(h, a)$ le regret "counterfactuel" de ne pas prendre l'action **a** à l'ensemble d'informations I.
- Soit $R_T^i(I, a) = \sum_{t=1}^{T} r_t^i(I, a)$ le regret "counterfactuel" cumulé de ne pas prendre l'action **a** à l'ensemble d'informations I du joueur i.

3.2 Pseudocode [4]

Initialize cumulative regret tables : $\forall I, r_I[a] \leftarrow 0$;
Initialize cumulative strategy tables : $\forall I, s_I[a] \leftarrow 0$;
Initialize initial profile : $\sigma_1(I,a) \leftarrow \frac{1}{|A(I)|}$;

Function CFR$(h_a, i, t, \pi_1, \pi_2)$;
if *h is terminal* **then**
  return $u_i(h)$;
else if *h is a chance node* **then**
  Sample a single outcome $a \sim \sigma_c(h,a)$;
  return CFR$(h_a, i, t, \pi_1, \pi_2)$;

Let I be the information set containing h;
$v_\sigma \leftarrow 0$;
$v_{\sigma I}[a] \leftarrow 0$; `// Initialize` $v_{\sigma I}[a]$ `to 0 for all` $a \in A(I)$

for $a \in A(I)$ **do**
  if $P(h) = 1$ **then**
    $v_{\sigma I \leftarrow a}[a] \leftarrow$ **CFR**$(h_a, i, t, \sigma_t(I,a) \cdot \pi_1, \pi_2)$;
  else if $P(h) = 2$ **then**
    $v_{\sigma I \leftarrow a}[a] \leftarrow$ **CFR**$(h_a, i, t, \pi_1, \sigma_t(I,a) \cdot \pi_2)$;
  end
  $v_\sigma \leftarrow v_\sigma + \sigma_t(I,a) \cdot v_{\sigma I \backslash a}[a]$;
end

if $P(h) = i$ **then**
  for $a \in A(I)$ **do**
    $r_I[a] \leftarrow r_I[a] + \pi_i \cdot (v_{\sigma I \leftarrow a}[a] - v_\sigma)$;
    $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma_t(I,a)$;
  end
  $\sigma_{t+1}(I,a) \leftarrow$ regret-matching values computed using Equation 5 and regret table r_I;
return v_σ;

Function *Solve()*

for $t = 1, 2, 3, \ldots, T$ **do**
  for $i \in 1, 2$ **do**
    CFR$(, i, t, 1, 1)$;
  end
end

Algorithm 1: Pseudocode

3.3 Kuhn poker et CFR

Nous pouvons représenter les décisions des joueurs dans un jeu de Kuhn Poker de deux joueurs sous la forme simplifiée suivante :

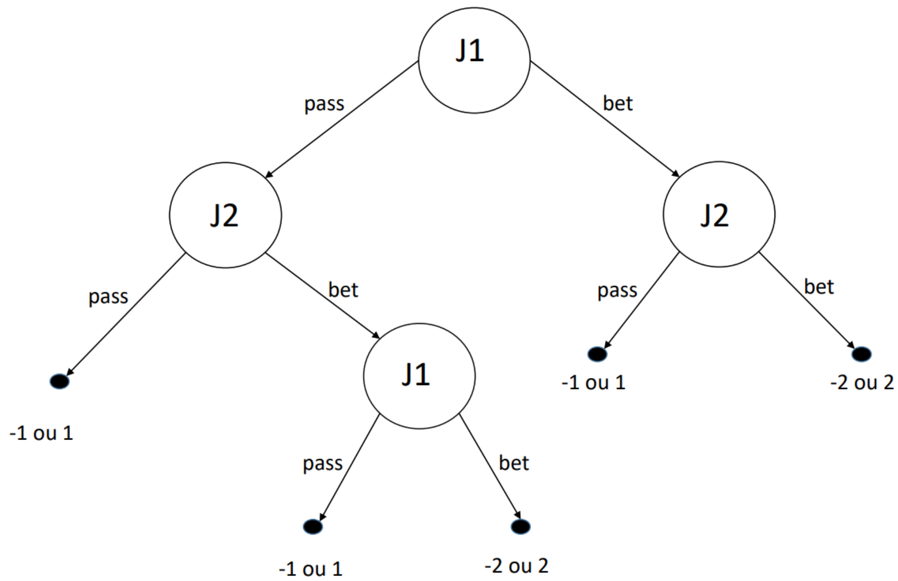


FIGURE 3 – Représentation simplifiée des décisions des joueurs

Le théorème de Nash et minimax (voir 2.1.5 et 2.1.6) garantissent l'existence d'au moins un équilibre de Nash pour le Kuhn Poker.
Nous allons donc utiliser l'algorithme CFR pour trouver une approximation du profil de stratégies d'équilibre de Nash.
Ceci nous permettra de ne pas perdre et de gagner surtout lorsque nous jouons contre des adversaires humains. En effet, jouer avec une stratégie d'équilibre de Nash contre un adversaire humain garantit d'avoir des gains puisque les adversaires humains sont suceptibles de faire des erreurs ce qui nous permettra de les expoloiter .

4 Résultats et discussions

4.1 Kuhn poker à 2 joueurs

Nous avons implémenté l'algorithme en Python. Après avoir exécuté le code pour 1000000 itérations, nous avons obtenu les résultats suivants :

Valeur de jeu :

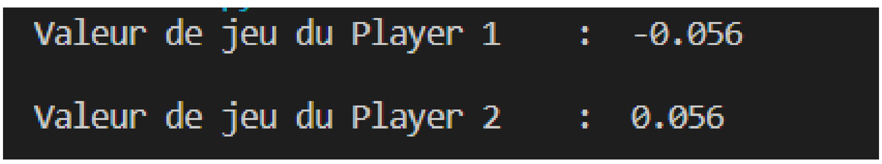


FIGURE 4 – Valeur de jeu attendue

Nous pouvons remaquer qu'en moyenne, le premier joueur perd(légèrement) et que le deuxième joueur gagne. Ceci peut être expliqué par le fait que le premier joueur agit en premier ce qui permet au deuxième joueur d'exploiter cette information.

Stratégies du premier joueur :

```
Player 1 strategies [Bet , pass]:

Q    :    [0.00 1.00]
K    :    [0.54 0.46]
J    :    [0.18 0.82]
Qpb  :    [0.51 0.49]
Kpb  :    [1.00 0.00]
Jpb  :    [0.00 1.00]
```

FIGURE 5 – Stratégies du premier joueur

Nous avons affiché les stratégies optimales pour les ensembles d'information du premier joueur.
Par exemple, pour l'ensemble d'information **Jpb**, qui correspond au cas où le premier joueur a une carte valet **(J)** et que le deuxième joueur a parié après un check de la part du premieur joueur, nous remarquons que la stratégie optimale est de toujours se coucher(fold), puisque nous somme sûr que le premier joueur perdera dans tous les cas, car le deuxième joueur aura la meilleure carte (K ou Q). Prenons aussi l'exemple de l'ensemble d'infomation **Kpb**, il est clair que la stratégie optimale est de toujours parier après un pari de la part du deuxième joueur puisque le premier joueur est sûr de gagner.
Nous pouvons aussi constater que **le bot apprend à bluffer.** En effet, pour l'ensemble d'information **J**, qui correspond au cas où le premier joueur a une carte de valet, nous remarquons que la probabilité de parier est de 0.18 alors qu'il a la main la plus faible. De même pour l'ensemble d'information **K** pour le premier joueur, la probabilité de passer(Check) est de 0.54 alors qu'il a la meilleure carte puisque le bot essaie de faire penser à l'adversaire qu'il ne possède pas de roi.

Stratégies du deuxième joueur :

```
Player 2 strategies [Bet , pass]:

Jb   :    [0.00 1.00]
Jp   :    [0.34 0.66]
Qb   :    [0.33 0.67]
Qp   :    [0.00 1.00]
Kb   :    [1.00 0.00]
Kp   :    [1.00 0.00]
```

FIGURE 6 – Stratégies du deuxième joueur

En suivant le même raisonnement, nous pouvons vérifier que les stratégies suivies par le deuxième joueur sont "rationnelles". Par exemple, au cas où le deuxième joueur possède une carte de Roi K et après un "check" ou un pari du premier joueur, la stratégie optimale est de toujours parier.

Evolution de la valeur du jeu en fonction du nombre d'itérations :

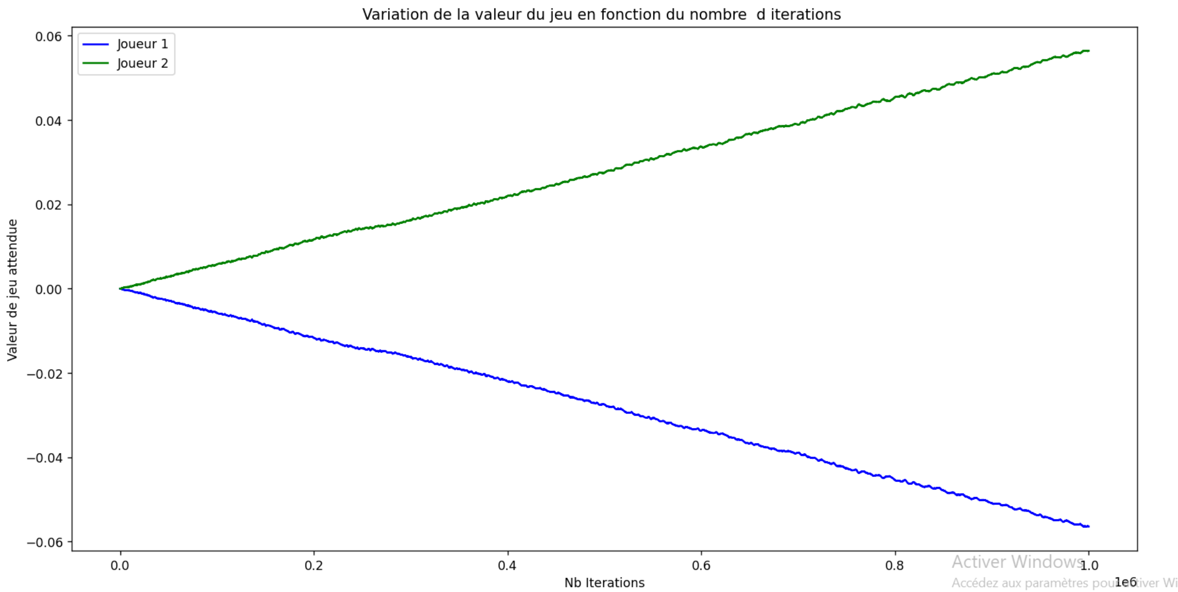


FIGURE 7 – Variation de la valeur du jeu

Nous remarquons que la valeur du jeu de deuxième joueur croît avec le nombre d'itérations, et que la valeur de jeu du premier joueur décroît avec le nombre d'itérations (Jeu à somme nulle).

Evolution de la probabilité de parier en fonction du nombre d'itérations :

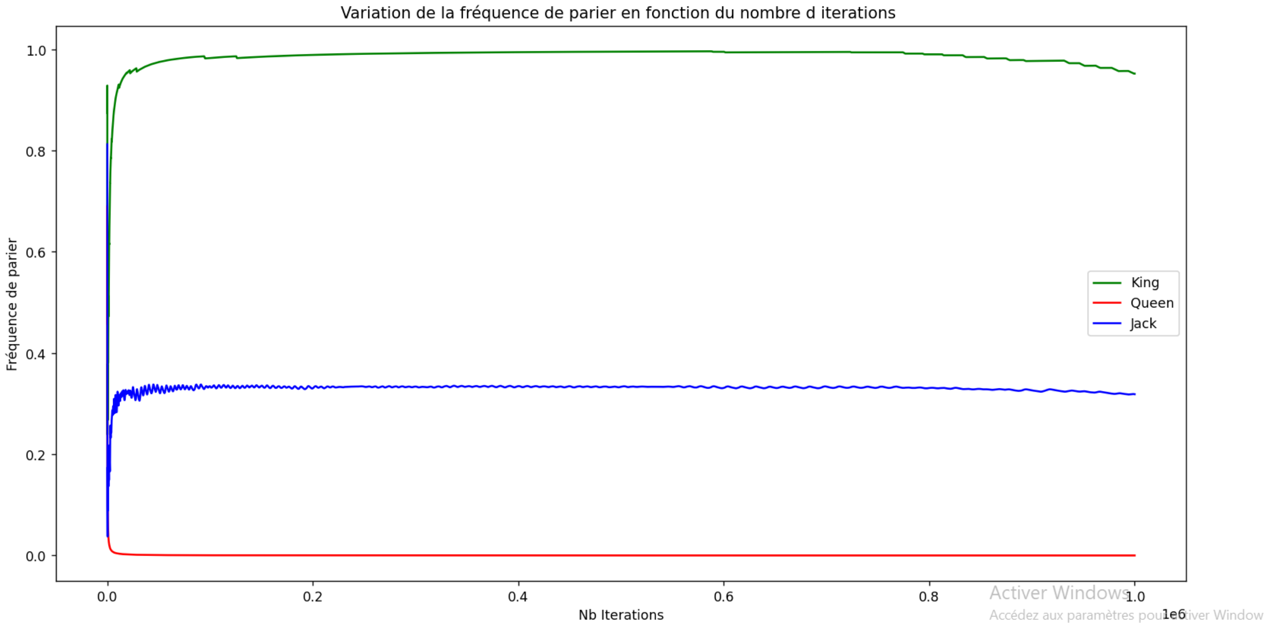


FIGURE 8 – Variation de la probabilité de parier

Nous remarquons que :

- La probabilité de parier avec un roi **K** Converge rapidement vers 0,90.
- De même, la probabilité de parier avec une reine **Q** converge rapidement vers 0.
- La probabilité de parier avec un valet **J** oscille avant de converger vers 0,30.

Après plusieurs exécutions du code, nous remarquons que la probabilité de parier avec un Roi **K** et la probabilité de parier avec un valet **J** pour le premier joueur varient d'une exécution à une autre. En effet, il existe plusieurs équilibres de Nash pour le Kuhn Poker. Et pour chaque exécution, nous tombons sur un différent équilibre et donc différents stratégies de jeu.

4.2 Kuhn poker Multijoueurs

Nous avons implémenté l'algorithme CFR en python pour une version Kuhn Poker multijoueurs (de 2 à 13 joueurs). Après avoir exécuté le code pour 100000 itérations, pour 4 joueurs et avec 5 cartes **(K,Q,J,10,9)**, Nous avons obtenu les résultats suivants :

Valeur de jeu :

```
Valeur de jeu du player 1 : -0.013
Valeur de jeu du player 2 : -0.019
Valeur de jeu du player 3 : -0.008
Valeur de jeu du player 4 : 0.039
```

FIGURE 9 – Valeur de jeu

Nous pouvons remarquer qu'en moyenne, les trois premiers joueurs perdent, alors que le quatrième joueur gagne. Il est donc clair que le fait de jouer en premier est un désavantage que le dernier joueur est en mesure d'exploiter.

Stratégies du premier joueur :
Nous avons utilisé la lettre **D** pour représenter la carte **10** pour avoir une chaîne de caractère qui contient un seul caractère au lieu de deux afin de ne pas avoir de problème lors de l'affichage des stratégies.

```
Player 1 strategies [Bet , pass]:

9    :    [0.01 0.99]
Q    :    [0.00 1.00]
K    :    [0.02 0.98]
D    :    [0.00 1.00]
J    :    [0.01 0.99]
9pbbb:    [0.00 1.00]
9pbbp:    [0.00 1.00]
9pbpb:    [0.00 1.00]
9pbpp:    [0.00 1.00]
9ppbb:    [0.00 1.00]
9ppbp:    [0.00 1.00]
9pppb:    [0.00 1.00]
Qpbbb:    [0.00 1.00]
Qpbbp:    [0.00 1.00]
Qpbpb:    [0.00 1.00]
Qpbpp:    [0.33 0.67]
Qppbb:    [0.00 1.00]
Qppbp:    [0.04 0.96]
Qpppb:    [0.06 0.94]
```

(a)

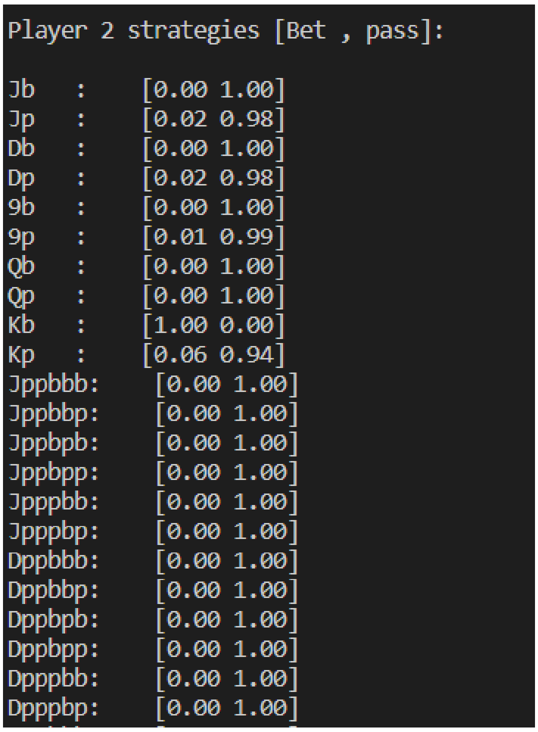
```
Kpbbb:    [1.00 0.00]
Kpbbp:    [1.00 0.00]
Kpbpb:    [1.00 0.00]
Kpbpp:    [1.00 0.00]
Kppbb:    [1.00 0.00]
Kppbp:    [1.00 0.00]
Kpppb:    [1.00 0.00]
Dpbbb:    [0.00 1.00]
Dpbbp:    [0.00 1.00]
Dpbpb:    [0.00 1.00]
Dpbpp:    [0.00 1.00]
Dppbb:    [0.00 1.00]
Dppbp:    [0.00 1.00]
Dpppb:    [0.00 1.00]
Jpbbb:    [0.00 1.00]
Jpbbp:    [0.00 1.00]
Jpbpb:    [0.00 1.00]
Jpbpp:    [0.12 0.88]
Jppbb:    [0.00 1.00]
Jppbp:    [0.00 1.00]
Jpppb:    [0.00 1.00]
```

(b)

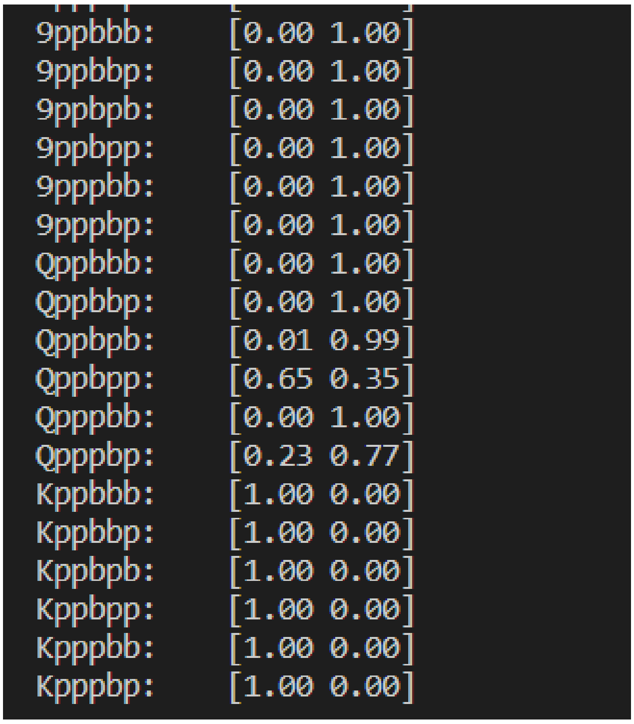
FIGURE 10 – Stratégies du premier joueur

Pour vérifier que les stratégies suivies par le premier joueur sont "rationnelles", Prenons l'ensemble d'information **Kppbp** ou **Kpppb** , il est clair que la stratégie optimale est de toujours parier puisque nous sommes sûr de gagner.

Stratégies du deuxième joueur :



(a)



(b)

FIGURE 11 – Stratégies du deuxième joueur

De même, Pour vérifier que les stratégies suivies par le deuxième joueur sont "rationnelles", Prenons l'ensemble d'information **9ppbbb** ou **Dppbbp** , il est clair que la stratégie optimale est de toujours passer (fold) puisque nous sommes sûr de perdre. Nous pouvons aussi remarquer que pour l'ensemble d'information **Kp** la probabilité de passer (check) est de 0,94. En effet, l'idée est de faire penser aux autres joueurs que nous nous possédons pas de roi **K**, ce qui montre que le bot apprend à bluffer.

Stratégies du troisième joueur :

```
Player 3 strategies [Bet , pass]:

Kbb  :    [1.00 0.00]
Kbp  :    [1.00 0.00]
Kpb  :    [1.00 0.00]
Kpp  :    [0.45 0.55]
Qbb  :    [0.00 1.00]
Qbp  :    [0.41 0.59]
Qpb  :    [0.02 0.98]
Qpp  :    [0.00 1.00]
Jbb  :    [0.00 1.00]
Jbp  :    [0.00 1.00]
Jpb  :    [0.00 1.00]
Jpp  :    [0.15 0.85]
9bb  :    [0.00 1.00]
9bp  :    [0.00 1.00]
9pb  :    [0.00 1.00]
9pp  :    [0.15 0.85]
Dbb  :    [0.00 1.00]
Dbp  :    [0.00 1.00]
Dpb  :    [0.00 1.00]
Dpp  :    [0.00 1.00]
```

(a)

```
Kpppbbb:    [1.00 0.00]
Kpppbbp:    [1.00 0.00]
Kpppbpb:    [1.00 0.00]
Kpppbpp:    [1.00 0.00]
Qpppbbb:    [0.50 0.50]
Qpppbbp:    [0.01 0.99]
Qpppbpb:    [0.00 1.00]
Qpppbpp:    [0.59 0.41]
Jpppbbb:    [0.00 1.00]
Jpppbbp:    [0.00 1.00]
Jpppbpb:    [0.00 1.00]
Jpppbpp:    [0.00 1.00]
9pppbbb:    [0.00 1.00]
9pppbbp:    [0.00 1.00]
9pppbpb:    [0.00 1.00]
9pppbpp:    [0.00 1.00]
Dpppbbb:    [0.00 1.00]
Dpppbbp:    [0.00 1.00]
Dpppbpb:    [0.00 1.00]
Dpppbpp:    [0.00 1.00]
```

(b)

FIGURE 12 – Stratégies du troisième joueur

De même, Pour vérifier que les stratégies suivies par le troisème joueur sont "rationnelles", Prenons l'ensemble d'information **Kpb** ou **Kpppbbb**, il est clair que la stratégie optimale est de toujours parier puisque nous sommes sûr de gagner avec un roi.
Par contre, pour l'ensemble d'information **Qbb**, la meilleur stratégie est de passer (fold) puisque l'un des deux premiers joueurs aura probablement un roi.

Stratégies du quatrième joueur :

```
Player 4 strategies [Bet , pass]:

Dbbb :    [0.00 1.00]
Dbbp :    [0.00 1.00]
Dbpb :    [0.00 1.00]
Dbpp :    [0.00 1.00]
Dpbb :    [0.00 1.00]
Dpbp :    [0.00 1.00]
Dppb :    [0.00 1.00]
Dppp :    [0.17 0.83]
9bbb :    [0.00 1.00]
9bbp :    [0.00 1.00]
9bpb :    [0.00 1.00]
9bpp :    [0.00 1.00]
9pbb :    [0.00 1.00]
9pbp :    [0.00 1.00]
9ppb :    [0.00 1.00]
9ppp :    [0.32 0.68]
```

(a)

```
Kbbb :    [1.00 0.00]
Kbbp :    [1.00 0.00]
Kbpb :    [1.00 0.00]
Kbpp :    [1.00 0.00]
Kpbb :    [1.00 0.00]
Kpbp :    [1.00 0.00]
Kppb :    [1.00 0.00]
Kppp :    [1.00 0.00]
Jbbb :    [0.00 1.00]
Jbbp :    [0.00 1.00]
Jbpb :    [0.00 1.00]
Jbpp :    [0.19 0.81]
Jpbb :    [0.00 1.00]
Jpbp :    [0.00 1.00]
Jppb :    [0.00 1.00]
Jppp :    [0.18 0.82]
Qbbb :    [0.50 0.50]
Qbbp :    [0.01 0.99]
Qbpb :    [0.00 1.00]
Qbpp :    [0.26 0.74]
Qpbb :    [0.00 1.00]
Qpbp :    [0.30 0.70]
Qppb :    [0.00 1.00]
Qppp :    [0.00 1.00]
```

(b)

Figure 13 – Stratégies du quatrième joueur

En suivant le même raisonnement, pour vérifier que les stratégies suivies par le dernier joueur sont "rationnelles", Prenons l'ensemble d'information **Kbbb** ou **Kbpp**, la meilleure stratégie est de toujours parier puisque nous sommes sûr de gagner avec un roi. Par contre, pour l'ensemble d'information **9ppb**, la meilleur stratégie est de passer (fold) puisque nous avons la carte la plus faible. Nous pouvons aussi remarquer que pour l'ensemble d'information **9ppp**, la probabilité de parier est de 0.32 ce qui montre de nouveau d'une part que le bot apprend à bluffer, et d'autre part l'avantage de jouer en dernier.

Variation de la valeur de jeu en fonction du nombre d'itérations :

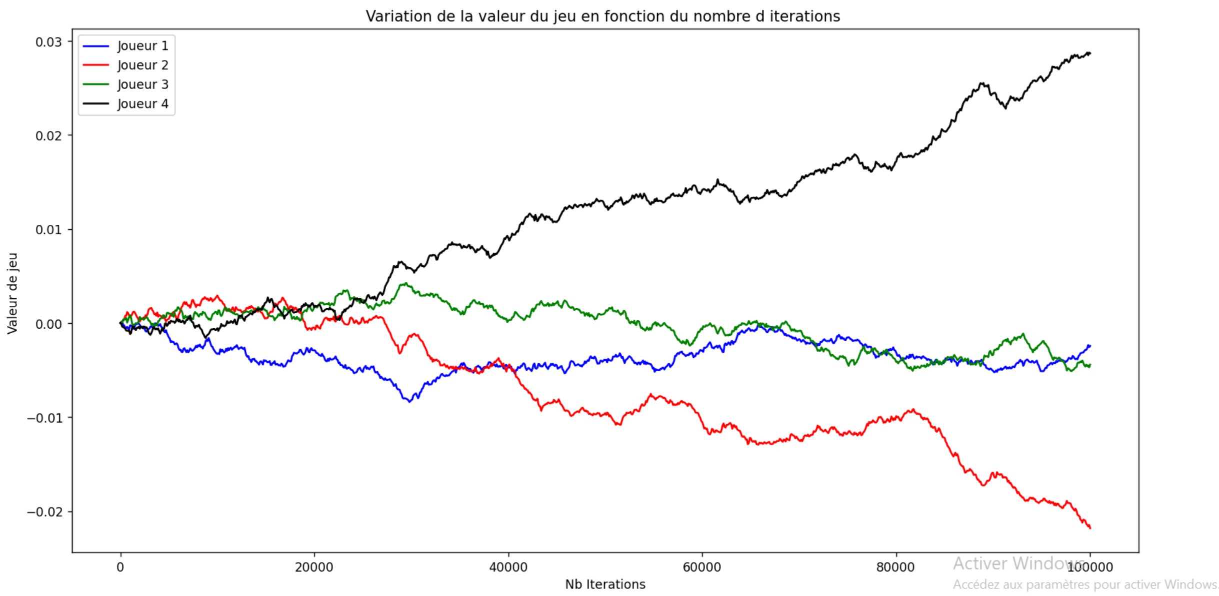


FIGURE 14 – Variation de la valeur de jeu

Nous remarquons que la valeur de jeu du dernier joueur croît avec le nombre d'itérations, alors que la valeurs des trois premiers joueurs décroît avec le nombre d'itérations.

D'après les résultats précédents, nous pouvons conclure que l'algorithme CFR est un outil efficace qui nous a permis de trouver les stratégies optimales de jeu et de résoudre le jeu Kuhn Poker.

5 Limites et perspectives

Limites :

Dans ce rapport, nous avons créee une intelligence artificielle qui est capable de résoudre une version "simple" du poker. En effet, la raison derrière le choix de cette version réside dans le fait que la version classique du poker (Texas Hold'em) est un jeu extrêmement complexe, avec un nombre important de combinaisons et actions possibles, ce qui rend la modélisation et l'apprentissage très difficile. De plus, la création d'une IA de poker classique nécessitera des ressources computationnelles importantes, en raison du grand nombre de calculs nécessaires. En outre, les ressources sur les travaux réalisés pour créer une intelligence artificielle qui joue au texas hold'em poker sont peu disponibles, car le partage de ces ressources de la part d'un établissement ne présente aucun intérêt pour cet établissement puisqu'il pourra facilement faire des grandes sommes d'argent en utilisant cette même IA.

Perspectives :

Nous visons à augmenter la complexité du jeu en ajoutant des règles supplémentaires, nous sommes aussi ouverts à l'exploration d'autres algorithmes ou d'autres variantes du

CFR, telles que le MCCFR (Monte Carlo Counterfactual Regret Minimization). Ceci permettra d'offir des nouvelles perspectives et des possibles améliorations en termes de performance.

6 Conclusion

Nous avons proposé dans ce rapport une application de l'algorithme CFR (counterfactual regret minmization) pour la résolution d'un jeu à information imparfaite.
Après avoir expliqué le princeipe de l'algorithme, analyser les résultats, nous pouvons conclure que cet algorithme est une approche efficace pour trouver des stratégies optimales dans des jeux à information imparfaite. En effet, la capacité de l'algorithme à converger vers un équilibre de Nash garantit que la stratégie trouvée est optimale et ne peut être améliorée par aucune autre stratégie.
C'est un outil puissant pour résoudre des jeux comme le Kuhn Poker grâce à sa capacité d'apprendre des expériences précédentes et ajuster sa stratégie de manière à maximiser les gains.

7 Annexe

7.1 Code du CFR Kuhn Poker(2 joueurs et 3 cartes) :

```
from matplotlib import pyplot as plt
import numpy as np
import random


def noeud_terminal(history):
    #on check si le noeud est terminal
    b = history in ['bp', 'bb', 'pp', 'pbb', 'pbp']
    return b

def get_gains(history, cards):
    if history in ['bp', 'pbp']:
        #si un joueur s'est couch
        gains = 1
        return gains
    else:
    # pp ou bb ou pbb (deux passes ou deux mises successives):
        if 'b' in history :
            #deux mises successives
            gains = 2
        else :
            #deux passes successives
            gains = 1
        current_player = len(history) % 2
        my_card = cards[current_player]
        opponent_card = cards[(current_player + 1) % 2]
        if my_card == 'K' or opponent_card == 'J':
            return gains
        else:
            return -gains
```

```


# parier et call(bet) ,  check et se coucher(pass):
possible_actions = ['b', 'p']

class InfoSet():
    def __init__(self):
        self.strategie_sum = np.zeros(len(possible_actions))
        self.regret_sum = np.zeros(len(possible_actions))
        self.n_actions = len(possible_actions)

    def get_strategie(self, r_p):
        strategie = np.maximum(0, self.regret_sum)
        # on normalise
        if sum(strategie) > 0 :
            strategie /= sum(strategie)
        else :
            strategie =np.repeat(1/self.n_actions, self.n_actions)
        self.strategie_sum += r_p * strategie
        return strategie

    def get_average_strategie(self):
        #on normalise
        if sum(self.strategie_sum) > 0 :
            return self.strategie_sum/sum(self.strategie_sum)
        else :
            return np.repeat(1/self.n_actions, self.n_actions)


class CFR():
    def __init__(self):
        # on initialise le dictionnaire des ensembles d'informations
        self.InfoSetMap = {}

    def get_infoset(self, card_history):
        #avoir l'ensemble d'information
        if card_history not in self.InfoSetMap:
            self.InfoSetMap[card_history] = InfoSet()
        return self.InfoSetMap[card_history]

    def cfr(self, cards, history, r_proba, current_player):
    ## fonction r cursive pour parcourir tous les noeuds possibles
        #si on est dans un noeud terminal on return les gains et on
    arrete la recursivit  du programme:
        if noeud_terminal(history):
            return get_gains(history, cards)

        my_card = cards[current_player]
        info_set = self.get_infoset(my_card + history)

        strategie = info_set.get_strategie(r_proba[current_player])
        opponent = (current_player + 1) % 2
        counterfact_act = np.zeros(len(possible_actions))

        for i, action in enumerate(possible_actions):
            action_proba = strategie[i]

```

```
                new_r_proba = r_proba.copy()
                new_r_proba[current_player] *= action_proba

                #on appelle la fonction cfr r cursivement
                counterfact_act[i] = -1 * self.cfr(cards, history + action,
    new_r_proba, opponent)

            valeur_noeud = counterfact_act.dot(strategie)
            for i, action in enumerate(possible_actions):
                info_set.regret_sum[i] += r_proba[opponent] * (
    counterfact_act[i] - valeur_noeud)

            return valeur_noeud

        def entrainement(self, nombre_iterations):
            #entraienement du modele
            val = 0
            P1 = []
            P2 = []
            P1K = []
            P1Q = []
            P1J = []
            iter = []
            deck = ['K', 'Q', 'J']
            for i in range(nombre_iterations):
                #shuffle deck:
                cards = random.sample(deck, 2)
                history = ''
                r_proba = np.ones(2)
                val += self.cfr(cards, history, r_proba, 0)
                # pour tracer la courbe des probabilit s de parier :
                """if i>10 :
                    iter.append(i)
                    for hist, I in cfr.InfoSetMap.items():
                        if hist == 'K':
                            P1K.append(I.get_average_strategie()[0])
                        elif hist == 'Q':
                            P1Q.append(I.get_average_strategie()[0])
                        elif hist == 'J':
                            P1J.append(I.get_average_strategie()[0])"""
                #pour tracer la courbe valeurs de jeu :
                """if i % 10 == 0:
                    iter.append(i)
                    P1.append(val/nombre_iterations)
                    P2.append(-(val/nombre_iterations))
                    #print(val/nombre_iterations)"""
            return val,iter,P1K,P1Q,P1J,P1,P2

if __name__ == "__main__":

    nombre_iterations = 100000

    #arrondir :
    np.set_printoptions(precision=2, floatmode='fixed', suppress=True)

    cfr = CFR()

```

```
    val,iter,P1K,P1Q,P1J,P1,P2 = cfr.entrainement(nombre_iterations)

    #affichage des listes de proba et des iter:
    """print ('iter---------------')
    print(iter)
    print(P1K)
    print(P1Q)"""

    #affichage des valeurs de jeu
    print(f"Valeur de jeu du Player 1    :  {round(val / nombre_iterations, 3)}\n")
    print(f"Valeur de jeu du Player 2    :  {round(-(val / nombre_iterations), 3)}\n")

    # pour filtrer les strategies de chaque joueur:
    P1strat = []
    P2strat = []
    for hist, I in sorted(cfr.InfoSetMap.items(), key=lambda x: len(x[0]) ) :
        if len(hist) % 2 == 0 :
            P2strat.append((hist,I.get_average_strategie()))
        else :
            P1strat.append((hist,I.get_average_strategie()))


    #print les strat du premier joueur
    print("\nPlayer 1 strategies [Bet , pass]:\n")
    for (hist, I) in P1strat :
        print(f"{hist:5}:    {I}")

    #print les strat du deuxi me joueur
    print("\nPlayer 2 strategies [Bet , pass]:\n")
    for (hist, I) in P2strat :
        print(f"{hist:5}:    {I}")


    # tracer la courbe des probabilit s de parier :
    """plt.plot(iter,P1K,'g', label = 'King')
    plt.plot(iter,P1Q,'r', label = 'Queen')
    plt.plot(iter,P1J,'b', label = 'Jack')
    plt.ylabel('Fr quence de parier')
    plt.xlabel('Nb Iterations')
    plt.title('Variation de la fr quence de parier en fonction du nombre d iterations ')
    plt.legend()
    plt.show()"""



    #tracer la courbe des valeur de jeu
    """plt.plot(iter,P1,'b',label = 'Joueur 1')
    plt.plot(iter,P2,'g', label = 'Joueur 2')
    plt.ylabel('Valeur de jeu attendue')
    plt.xlabel('Nb Iterations')
    plt.title('Variation de la valeur du jeu en fonction du nombre  d iterations ')
    plt.legend()
```

```
    plt.show()"""
```

7.2 Code du CFR Kuhn Poker Multijoueurs (de 2 joueurs à 13 joueurs, 3 cartes à 14 cartes) :

```
from matplotlib import pyplot as plt
import random
import numpy as np


def get_card(carte, n_joueurs):
    #get le nom de la carte : nombre -> nom
    return deck[-n_joueurs - 1:][int(carte[0])] + carte[1:]


def tous_fold(history, n_joueurs):
    return len(history) >= n_joueurs and history.endswith('p' * (
    n_joueurs - 1))


def get_gains(cards, history, n_joueurs) :
    #return les gains
    player = len(history) % n_joueurs
    player_cards = cards[:n_joueurs]
    num_opponents = n_joueurs - 1
    if history == 'p' * n_joueurs:
        gains = [-1] * n_joueurs
        gains[np.argmax(player_cards)] = num_opponents
        return gains
    elif tous_fold(history, n_joueurs):
        gains = [-1] * n_joueurs
        gains[player] = num_opponents
    else:
        gains = [-1] * n_joueurs
        active_cards = []
        active_indices = []
        for (i, j) in enumerate(player_cards):
            if 'b' in history[i::n_joueurs]:
                gains[i] = -2
                active_cards.append(j)
                active_indices.append(i)
        gains[active_indices[np.argmax(active_cards)]] = len(
    active_cards) - 1 + num_opponents
    return gains


def noeud_terminal(history, n_joueurs):
    #check si le noeud est terminal
    all_raise = history.endswith('b' * n_joueurs)
    all_acted_after_raise = (history.find('b') > -1) and (len(history) -
    history.find('b') == n_joueurs)
    all_but_1_player_folds = tous_fold(history, n_joueurs)
    return all_raise or all_acted_after_raise or all_but_1_player_folds


#cartes de jeu
deck = ['1','2','3','4','5','6','7','8','9', 'D', 'J', 'Q', 'K']
```

```python

# parier et call(bet) ,  check et se coucher(pass):
possible_actions = ['b', 'p']


class InfoSet():
    def __init__(self):
        self.n_actions = len(possible_actions)
        self.strategie_sum = np.zeros(len(possible_actions))
        self.regret_sum = np.zeros(len(possible_actions))

    def get_strategie(self, r_p):
        strategie = np.maximum(0, self.regret_sum)
        #on normalise
        if sum(strategie) > 0 :
            strategie /= sum(strategie)
        else :
            strategie =np.repeat(1/self.n_actions, self.n_actions)
        self.strategie_sum += r_p * strategie
        return strategie

    def get_average_strategie(self):
        #on normalise
        if sum(self.strategie_sum) > 0 :
            return self.strategie_sum/sum(self.strategie_sum)
        else :
            return np.repeat(1/self.n_actions, self.n_actions)



class CFR():
    def __init__(self, n_joueurs):
        self.InfoSetMap = {}
        self.n_joueurs = n_joueurs
        self.cards = [i for i in range(self.n_joueurs + 1)]

    def get_infoset(self, card_and_history) :
        #get l'ensemble d'information
        if card_and_history not in self.InfoSetMap:
            self.InfoSetMap[card_and_history] = InfoSet()
        return self.InfoSetMap[card_and_history]


    def cfr(self, cards, history, r_proba, current_player):
    #fonction r cursive pour parcourir tous les noeuds possibles
        #si noeud terminal:
        if noeud_terminal(history, self.n_joueurs):
            return get_gains(cards, history, self.n_joueurs)

        my_card = cards[current_player]
        info_set = self.get_infoset(str(my_card) + history)

        strategie = info_set.get_strategie(r_proba[current_player])
        opponent = (current_player + 1) % self.n_joueurs

        counterfact_act = [None] * len(possible_actions)

```

```python
        for i, action in enumerate(possible_actions):
            proba_act = strategie[i]
            new_r_proba = r_proba.copy()
            new_r_proba[current_player] *= proba_act
            #appel recursif de la fct
            counterfact_act[i] = self.cfr(cards, history + action, new_r_proba, opponent)

        val_noeud = strategie.dot(counterfact_act)

        for i, action in enumerate(possible_actions):
            counterfact_r_proba = np.prod(r_proba[:current_player]) * np.prod(r_proba[current_player + 1:])
            regrets = counterfact_act[i][current_player] - val_noeud[current_player]
            info_set.regret_sum[i] += counterfact_r_proba * regrets
        return val_noeud

    def entrainement(self, nombre_iterations):
        #entraitnement:
        val = np.zeros(self.n_joueurs)
        P1 = []
        P2 = []
        P3 = []
        P4 = []
        P1K = []
        P1Q = []
        P1J = []
        iter = []
        for i in range(nombre_iterations):
            cards = random.sample(self.cards, self.n_joueurs)
            history = ''
            r_proba = np.ones(self.n_joueurs)
            val += self.cfr(cards, history, r_proba, 0)
            #pour tracer la courbe de valeurs de jeu
            """if i % 100 == 0 or i == nombre_iterations-1:
                iter.append(i)
                P1.append(val[0]/nombre_iterations)
                P2.append((val[1]/nombre_iterations))
                P3.append((val[2]/nombre_iterations))
                P4.append((val[3]/nombre_iterations))
                #print(val/nombre_iterations)"""
        return val,iter,P1K,P1Q,P1J,P1,P2,P3,P4




if __name__ == "__main__":

    nombre_iterations = 100000


    #on arrondit:
    np.set_printoptions(precision=2, floatmode='fixed', suppress=True)


    n_joueurs = 4
```

```python
cfr = CFR(n_joueurs)


val,iter,P1K,P1Q,P1J,P1,P2,P3,P4 = cfr.entrainement(
nombre_iterations)
P1strat = []
P2strat = []
P3strat = []
P4strat = []

#print val de jeu
for joueur in range(n_joueurs):
    print(f"Valeur de jeu du player {joueur + 1} : {(val[joueur] /
nombre_iterations):.3f}")


#filtre les strat de chaque joueur
for hist, I in sorted(cfr.InfoSetMap.items(), key=lambda x: len(x
[0]) ) :
    if len(hist) == 1 or len(hist) == 5 :
        P1strat.append((get_card(hist,n_joueurs),I.
get_average_strategie()))
    elif len(hist) == 2 or len(hist) == 6 :
        P2strat.append((get_card(hist,n_joueurs),I.
get_average_strategie()))
    elif len(hist) == 3 or len(hist) == 7 :
        P3strat.append((get_card(hist,n_joueurs),I.
get_average_strategie()))
    else :
        P4strat.append((get_card(hist,n_joueurs),I.
get_average_strategie()))


#print val de jeu
for joueur in range(n_joueurs):
    print(f"Valeur de jeu du player {joueur + 1} : {(val[joueur] /
nombre_iterations):.3f}")


#print strat de chaque joueur
print("\nPlayer 1 strategies [Bet , pass]:\n")
for (hist, I) in P1strat :
    print(f"{hist:5}:    {I}")

print("\nPlayer 2 strategies [Bet , pass]:\n")
for (hist, I) in P2strat :
    print(f"{hist:5}:    {I}")

print("\nPlayer 3 strategies [Bet , pass]:\n")
for (hist, I) in P3strat :
    print(f"{hist:5}:    {I}")

print("\nPlayer 4 strategies [Bet , pass]:\n")
for (hist, I) in P4strat :
    print(f"{hist:5}:    {I}")

```

```

    #print les listes de valeur de jeu
    """print(iter)
    print('--------')
    print(P1)
    print('--------')
    print(P2)
    print('--------')
    print(P3)
    print('--------')
    print(P4)
    print('--------')"""


    #plot les valeurs de jeu
    """plt.plot(iter,P1,'b',label = 'Joueur 1')
    plt.plot(iter,P2,'r', label = 'Joueur 2')
    plt.plot(iter,P3,'g', label = 'Joueur 3')
    plt.plot(iter,P4,'k', label = 'Joueur 4')
    plt.ylabel('Valeur de jeu ')
    plt.xlabel('Nb Iterations')
    plt.title('Variation de la valeur du jeu en fonction du nombre d iterations ')
    plt.legend()
    plt.show()"""
```

8 Bibliographie

[1] : Renaud Bourlès and Dominique Henriet. Théorie des jeux. Support de cours de l'Ecole Centrale de Marseille, 2016-2017.

[2] : Policnonmics. Game theory. Disponible sur : https://policonomics.com/game-theory/

[3] : Proffessor BRYCE. Counterfactual Regret Minimization (AGT 26). Publié le 25 avril 2023. Disponible sur : https://www.youtube.com/watch?v=ygDt_AumPr0&t=1968s&ab_channel=ProfessorBryce

[4] : Todd W Neller and Marc Lanctot. An introduction to counterfactual regret minimiza- tion. In Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013), volume 11, 2013

[5] : Wikipedia. Théorème du minimax de von Neumann. Dernière modification : le 26 août 2022. Disponible sur : https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_du_minimax_de_von_Neumann

[6] : Gabriel LEPETIT. Théorie des jeux à somme nulle et Théorème du minimax. Publié en février 2020.

[7] : Wikipedia. Poker. Dernière modification : le 14 mai 2023. Disponible sur : https://fr.wikipedia.org/wiki/Poker

[8] : Wikipedia. Kuhn Poker. Denière modification : le 15 mai 2023. Disponible sur : https://en.wikipedia.org/wiki/Kuhn_poker