**Information Set et Stratégie**

Un information Set = l’ensemble des informations dont dispose un joueur lorsqu’il doit prendre une décision.

Un information Set est donc composé de 2 parties :

* **l’historique des actions** passées pour la partie en cours
* les **informations sur les cartes** : ici la carte privée seulement et au poker Texas hold'em les 2 cartes privées plus les cartes publiques sur la table.

Il y a 12 informations Sets au Kuhn poker

|  |  |  |  |
| --- | --- | --- | --- |
| Liste des informations Set | | Actions possibles pour chaque information set | explications |
| Historique des actions | Carte privée du joueur |
| aucune | Valet | Check, Bet | Je joue en 1er et j’ai un Valet |
| aucune | Dame | Check, Bet | Je joue en 1er et j’ai une Dame |
| aucune | Roi | Check, Bet | Je joue en 1er et j’ai un Roi |
| check | Valet | Check, Bet | L’autre joueur à fait Check et j’ai un Valet |
| check | Dame | Check, Bet | L’autre joueur à fait Check et j’ai une Dame |
| check | Roi | Check, Bet | L’autre joueur à fait Check et j’ai un Roi |
| bet | Valet | Fold, Bet | L’autre joueur à fait Bet et j’ai un Valet |
| bet | Dame | Fold, Bet | L’autre joueur à fait Bet et j’ai une Dame |
| bet | Roi | Fold, Bet | L’autre joueur à fait Bet et j’ai un Roi |
| Check,bet | Valet | Fold, Bet | J’ai fait Check puis l’autre joueur à fait Bet et j’ai un Valet |
| Check,bet | Dame | Fold, Bet | J’ai fait Check puis l’autre joueur à fait Bet et j’ai une Dame |
| Check,bet | Roi | Fold, Bet | J’ai fait Check puis l’autre joueur à fait Bet et j’ai un Roi |

Une **Stratégie** pour un joueur consiste à avoir pour chaque information Set une probabilité de réaliser chaque action possible.

Ex : au Kuhn poker il y a 12 informations Sets et pour chacun il y a 2 actions possibles donc au total il y aura 24 probabilités à déterminer.

**Arbre du jeu poker**

Tout d’abord il faut construire **l’arbre du jeu poker** auquel on joue.

Cet arbre n’est pas le même au poker Khun et au poker hold them up à deux (cible pour le projet).

Cet arbre est constitué de **nœuds** (faire une classe Nœud).

Il y a 3 sortes de nœuds :

* **Nœud chance** : pas d’action des joueurs mais une action de l’ordinateur pour faire un choix de carte(s) aléatoire. Faire une classe NoeudChance qui hérite de Nœud. Notamment bien stocker la probabilité de tirage associée à chaque Nœud enfant du nœud chance (1/6 dans le cas du poker Khun => cette information doit descendre dans les nœuds de l’arbre).
* **Nœud décision** : un des deux joueurs doit prendre une décision parmi une liste de décisions possibles. Faire une classe NoeudDecision qui hérite de Nœud. Chaque nœud décision doit avoir un attribut **Information Set**.
* **Nœud terminal** : la partie est terminée. On peut alors calculer les gains/pertes de chacun des deux joueurs. NoeudTerminal qui hérite de Nœud.

L’arbre complet représente tout ce qui peut se passer dans une partie.

Dans le cas du poker hold them up on va réduire cet arbre complet pour des raisons de temps de calculs :

* Réduction du nombre de nœuds enfants des chances nodes. Ex : le 1er nœud chance donne 6 nœuds décision enfants dans le Khun poker car il y a seulement 6 tirages possibles des cartes des 2 joueurs. Dans le cas du poker hold them on va classer tout couple de 2 cartes distribuées à un joueur au sein d’environ 10 clusters (travail en cours) et il y aura donc 10\*10=100 nœuds décision enfants.
* Réduction du nombre de décisions possibles dans les nœuds décision. Dans un premier temps on pourrait se limiter à check, bet, fold (toutes les actions n’étant pas possible dans tous les nœuds décisions : pas de check si l’autre joueur à bet ; pas de fold si l’autre joueur à check). On étendra la liste des actions possible dans un second temps.

NB : l’arbre poker hold them up sera déjà gros même avec toutes ces réductions car il y a 4 noeuds chances : distribution des 2 cartes, distribution flop, distribution 4e carte, distribution 5e carte.

Si pour chacun on a 100 nœuds enfants possibles alors le total sera au moins de : 100^4 soit déjà 100 millions !

**Counterfactual regret minimization (CRM)**

**Introduction**

C’est une fonction récursive de parcours de l’arbre en profondeur qu’on appelle au départ à la racine de l’arbre (le nœud est un des paramètres de la fonction).

Cette fonction retourne l’utilité (ex : si je calcule l’utilité sur la racine de l’arbre alors ce sera une somme pondérée des utilités calculées sur les nœuds terminaux, pondérées par 3 paramètres : proba décisions joueur A de faire ce parcours dans l’arbre \* proba décisions joueur B de faire ce parcours dans l’arbre \* proba que les nœuds chances nous amènent à faire ce parcours de l’arbre.

Donc les paramètres = nœud, proba décisions joueur A, proba décisions joueur B.

Au départ, lors du 1er appel de la fonction ces paramètres valent : nœud racine,1,1

**Cœur de la fonction**

Si nœud en paramètre de la fonction = Nœud terminal

Calculer l’utilité : on peut le faire.

Si nœud en paramètre de la fonction = Nœud chance

Tirer au hasard un nœud enfant.

Si nœud en paramètre de la fonction = Nœud décision

Récupérer l’information set lié au Nœud (cela a été calculé lors de la construction de l’arbre cf point en jaune page précédente).

Parcourir les actions possibles (c’est dans l’information set) :

Pour chaque action : utilité nœud enfant = résultat de l’appel de la fonction CRM sur l’enfant en ayant pris soin de mettre à jour la proba décisions joueur A OU la proba décisions joueur B.

On stocke l’utilité de chaque nœud enfant et la somme des utilités cf code python infra.

|  |
| --- |
| child\_state\_utility = self.\_cfr\_utility\_recursive(state.play(action), child\_reach\_a, child\_reach\_b)  # value computation for current node  value += self.sigma[state.inf\_set()][action] \* child\_state\_utility  # values for chosen actions (child nodes) are kept here  children\_states\_utilities[action] = child\_state\_utility |

La grosse astuce c’est que :

* somme des utilités (paramètre value code Python) = utilité globale au niveau du décision node en cours d’instruction sous la stratégie actuelle.
* utilité de chaque nœud enfant = utilité du nœud enfant en supposant qu’on prend la décision d’aller vers ce nœud enfant dans 100% des cas. C’est pour cela que dans le code pour faire la MAJ de value il est fait child\_state\_utility \* **self.sigma[state.inf\_set()][action]**

Du coup, ensuite calculer le regret pour une action donnée est facile : on compare les deux :

**children\_states\_utilities[action] - value**

**Mises à jour des stratégies**

La mise à jour se fait au niveau de l’information Set (toujours l’information set associé au Nœud décision  évoqué dans le paragraphe précédent) :

S’il n’y a aucun regret pour aucune action de l’IS alors chaque action de l’IF à la même proba (1/2,1/2 dans le cas des IS du poker Khun).

Sinon : on ajoute pour chaque action le regret supplémentaire qu’on vient de calculer. Et la proba de chaque action = somme cumulées des regrets relatifs à l’action / somme cumulées des regrets relatifs à l’ensemble des actions de l’IS.

Pour plus de détail lire page 4 du document [[Todd W. Neller Marc Lanctot ] an introduction to counterfactual regret minimization](http://modelai.gettysburg.edu/2013/cfr/cfr.pdf) Ou il y a un exemple très simple avec exemple de pierre/feuille/ciseaux.

On vient de voir comment passer de stratégie n à stratégie n+1 (n étant l’itération dans laquelle on se trouve). On fera au total mettons 1000 itérations de Counterfactual regret minimization (CRM).

**Enfin, il faut stocker également les n stratégies** car une fois les n itérations faites, la stratégie finale est la moyenne des n stratégies c’est-à-dire qu’on fera un parcours de toutes les actions possible de tous les IS et que la proba finale sera la moyenne des probas.

Ex : pierre/feuille/ciseaux (un seul information Set)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Stratégie 0 | Stratégie 1 | Stratégie 2 | Stratégie finale : la moyenne |
| Proba pierre | 1/3 | 0 | 1/6 | 0,17 |
| Proba feuille | 1/3 | 1/3 | 3/6 | 0,39 |
| Proba ciseaux | 1/3 | 2/3 | 2/6 | 0,44 |

**Stratégie**

Il faut créer une classe stratégie.

Une stratégie c’est une liste **d’information Set.**

Il faut créer une classe Information Set. **Un information Set** est défini par :

* L’historique des actions passées (i.e décisions prises dans les Nœud décision).
* La/les cartes privées du joueur. Au poker Khun ce sera une carte. Au poker hold them ce sera l’appartenance à un cluster parmi 10.
* Les cartes publiques (None/vide dans le cas du poker Khun mais ce sera différent au poker hold them)
* Les actions possibles et les probabilités associées à chaque action.