PYTHON Projet

*Calculateur Texas Hold’em*

Le but de ce projet est d’écrire un programme permettant le calcul de l’espérance de gain pour la variante « Texas Hold’em » du jeu de poker. L’utilisateur saisit ainsi ses deux cartes cachées, en utilisant un système de notation spécifique, et les cartes commune (de 0 à 5 selon l’étape de jeu à laquelle on se trouve).

La première difficulté a été de trouver une bonne manière d'encoder les cartes. Nous avons d’abord pensé à créer une fonction « séparateur » dont le but était, à partir d’une liste de carte, d’obtenir, soit les valeurs numériques, soit les couleurs. Ainsi, cela nous permettait, de coder plus facilement les fonctions déterminant s’il y a paire, double paire…En plus de cette fonction on avait besoin d’une fonction permettant de convertir les valeurs des cartes, en valeurs numériques et d’une autre fonction détectant les doublons. Cependant, après réflexion, nous avons cherché un encodage plus simple et plus court, pour homogénéiser l’entrée des fonctions et surtout pour accélérer le temps de calcul d’espérance final.

Le programme prend donc en entrée une liste de carte. Cette liste sera convertit au début du programme, la liste obtenue servira en entrée de chaque fonction. Cette fonction est la fonction « convert », ainsi pour une liste L de cartes on aura :

L=[‘As’,Td’,7c’,7h’,3c’]

convert(L)=[214,310,107,407,103]

La nouvelle liste obtenue est simple d’utilisation. Pour obtenir les valeurs, pour chaque carte, il suffit de diviser la carte modulo 100, et pour obtenir les couleurs d’effectuer la division entière par 100 (avec ‘h’=4, ‘d’=3, ‘s’=2, ‘c’=1)

Cette nouvelle liste obtenue, sera en entrée des fonctions, permettant de savoir si il y a paire, double paire, brelan, quinte, flush, full house, carre ou encore le coup fatal la quinte flush.

Chacune de ces fonctions sont testées dans le programme principal dans une boucle et dans un ordre bien précis, de la fonction renvoyant la meilleure combinaison (fonction « quinte flush ») à la moins bonne (fonction « hauteur »). Cela permet d’avoir des fonctions, avec un algorithme plus simple. Dès lors qu’une fonction renvoie une liste non vide la boucle s’arrête. Ainsi par exemple, la fonction « paire » teste simplement s’il y a une paire, et renvoie s’il y a, la paire. Sachant qu’auparavant les autres fonctions auront été testé et auront renvoyé une liste vide, la fonction « paire » n’a pas besoin de tester s’il y a une double paire…

Ces fonctions inscrites dans une boucle, qui testent toutes les combinaisons possible, renvoient la meilleure combinaison de 5 cartes. Une fonction au nom « compare », teste donc, avec en entrée, les mains et un bord, la main gagnante.

Pour finir il nous reste à calculer des probabilités sur les mains, en ayant que certaines cartes du bord, ou encore, en ne connaissant qu’une main. On a donc finalisé l’algorithme en créant 4 dernières fonctions :

* La fonction « main\_preflop », qui à partir de deux mains, nous renvoie la probabilité de gagner pour chacune des mains.
* La fonction « main\_preflop », qui à partir de deux mains, nous renvoie la probabilité de gagner pour chacune des mains.
* La fonction « main\_turn », qui à partir de deux mains, et un flop de trois cartes, nous renvoie la probabilité de gagner pour chacune des mains
* La fonction « main\_f », qui à partir de deux mains, et un flop de quatre cartes, nous renvoie la probabilité de gagner pour chacune des mains.

Chacune de ces fonctions, génère aléatoirement les cartes manquantes, avec un nombre de simulation donnée, pour pouvoir utiliser la fonction « compare » à plusieurs reprises, et ainsi calculer le nombre de fois qu’une main gagne. Il suffit pour calculer la probabilité, de gagner pour une main, de diviser le nombre de fois que la main gagne par le nombre d’utilisation de la fonction compare. C’est la méthode de Monte-Carlo. Mais cette méthode a ses limites, elle ne donne pas un résultat précis. Cependant, nous n’avons pas besoin d’avoir le résultat exact mais juste une probabilité assez proche. Et si le nombre de simulation effectué est trop élevé, le programme peut prendre plusieurs secondes avant de donner le résultat. Pour un résultat satisfaisant, un nombre de 10.000 simulations est nécessaire, car on obtient le même résultat à 2% près à chaque lancement du programme. Après, avec un nombre de simulation dépassant les 10.000, le temps d’attente du résultat peut être trop élevé. D’ailleurs on a importé la fonction time, qui nous a permis de tester le temps d’attente. Généralement on a un temps voisinant les 12 secondes.

On a ajouté également le « pot odd », soit la valeur maximale (exprimé en coefficient multiplicateur du pot) que le joueur peut investir, tout en préservant une espérance positive. On le calcule simplement en fonction des deux probabilités.

Pour finir ce projet, une fonction au nom « graphique », gère l’affichage en utilisant la bibliothèque graphique « Tkinter ». Cette fonction ouvre une nouvelle fenêtre, qui affiche, les cartes données par l’utilisateur, et la probabilité de gagner pour chacune d’elle. Ce projet peut être compléter. On peut ainsi calculer les probabilités pour plus de 3 mains. Ainsi les fonctions calculant la probabilité n’auront plus en entrée les deux mains mais une liste de mains. Puis juste quelques modifications, seront à apporter.