

*Interface graphique du Black Jack*

# Projet INF101 S1

## BLACK JACK

Da Costa Barros Fabien | Min-I | 09/12/18

# Sommaire

## I- La réalisation

- a. Les modifications
- b. Les diagrammes avec matplotlib
- c. L'interface graphique avec tkinter

## II- Les fonctionnalités supplémentaires implémenter

- a. L'assurance
- b. Doubler
- c. Diviser

## III- Étude de différentes stratégies

- a. Les mises
- b. Toutes les IA de pioche s'affrontent avec la même mise
- c. Analyse de l'IA qui joue selon le score des autres
- d. Meilleure probabilité aléatoire de pioche

# Introduction

*Le programme Black Jack fonctionne avec les images du dossier où il est placé, le fichier licence donne la provenance et les droits sur les cartes. Le programme dit sans mise fonctionne sans ressources extérieures.*

Le début du projet d'informatique, après 2 mois de cours était quelque chose de très attendu. C'était un moment où nous étions moins dirigés et j'ai pu trouvé moi même des solutions à mes problèmes à l'aide des recherches sur internet ou des moments d'entraide entre camarades de classe. Par exemple, dans le cas de l'interface graphique que j'ai réalisé à l'aide de tkinter j'ai dû me débrouiller pour comprendre et programmer mon interface car je n'avais aucun antécédent avec tkinter.

Je me suis beaucoup renseigné sur les règles du Black jack avant de suivre toutes les fonctions à programmer au début du sujet. Je me suis rendu compte que le projet simplifiait en grande partie les règles du Black Jack, et que nous avions la liberté d'implémenter de nouvelles fonctionnalités comme Diviser, Doubler ou Assurer. Le plus dommage étant que dans cette version demandée nos gains ne dépendent pas de notre mise mais de celle des autres. L'intérêt de miser une forte somme est donc réduit car on gagne ce que les autres ont misé en récupérant notre mise. De fait, si on mise beaucoup et que tous le monde a misé peu alors on perd beaucoup. Les risques de pertes sont élevés pour les mêmes gains. J'ai donc instauré une mise minimum de 10 kopecs. Je reviendrai dans le développement sur les solutions prises pour éviter les abus en jeu.

Les premières fonctions sont très guidées car on nous impose un nombre de paramètres et les valeurs de retours. D'une part cela est très bien parce qu'il permet d'implémenter le reste des autres fonctions et de réaliser le premier programme sans mise très facilement. D'autre part, quand on arrive à la partie avec les mises tout se complique et il devient très difficile d'implémenter du code dans ce qui n'est pas pensé par nous-même.

Les différentes possibilités de mise et de pioche laissent de nombreuses perspectives pour créer et implémenter ses propres idées. J'ai donc 5 stratégies de pioche dont 2 que j'ai inventé par moi-même et 8 manières de piocher donc 4 de plus que celle demandées dans le sujet. L'avantage est que tout le monde voit les cartes des autres et les manières de jouer ne sont pas spéculatives mais basés sur des chiffres concrets, contrairement à un Poker où on doit deviner le jeu des autres pour pouvoir jouer.

Une des plus grandes difficultés a été l'interface graphique car je n'avais jamais programmé d'interface graphique avec Python auparavant. Dans les IDE que j'ai utilisé comme Lazarus (Langage : Pascal) et Unity (Langage : C#). L'interface graphique est déjà visible et on positionne les objets dedans. C'était donc pour moi la première fois que je crée une fenêtre où il faut placer les images, boutons et labels à l'intérieure. J'ai donc choisi tkinter car je pouvais demander de l'aide à des amis du groupe de TD en cas de difficultés avec ce module car certains avait déjà utilisé tkinter antérieurement.

# I. La réalisation

## A. Les modifications

Il y a plusieurs cas où j'ai choisi de donner un autre sens au projet que celui souhaité par le sujet. Il y a une seule pioche de  $n$  paquets. Le choix de la valeur des as ne se fait plus de la même manière que dans la première version du jeu. Lors de recherche, j'ai vu qu'on affichait souvent les différents scores possibles lorsqu'un joueur piochait un as. Je me suis donc dit que cela pourrait être une bonne alternative pour stocker les deux scores possibles et de choisir le meilleur score pour les IA et le croupier.

### 1. La valeur des As

J'ai donc implémenter une fonction `calculScoreAuto(scores, carte)` avec en argument le scores du joueur ou du croupier concerné et la carte tirée qui renvoie un string des deux scores possibles séparés par un « / ». Dans cette fonction, on ajoute aux anciens scores la ou les valeur(s) de la nouvelle carte et on ne laisse aucun doublon.

Par exemple, si on tire un As notre score passe de 0 à 1/11, si on en tire un deuxième toutes les possibilités de scores calculées sont 1+1, 1+11, 11+1 et 11+11 mais  $1+11=11+1=12$  et  $11+11=22 > 21$  alors la fonction renvoie « 2/12 ». Ce qui permet cela, c'est la liste des nouveaux scores (variable locale) qui stocke les scores déjà calculés. On ajoute une condition qui vérifie si le nouveau score calculé n'est pas déjà dans la liste et si il est inférieure ou égale à 21, avant de l'ajouter à la liste.

Dans tous les cas le score renvoyé est de type string car on ne peut pas stocker la valeur 1/11 dans un entier cependant on pourrait stocker la liste des scores possibles directement dans le dictionnaire « scores » à la clé correspondante au joueur mais l'affichage serait moins propre sous forme de liste.

Pour récupérer un entier et pour faciliter les comparaisons booléennes ou le choix du vainqueur, j'ai aussi implémenté une fonction `meilleurScore(scores, j)` qui prend le dictionnaire « scores » et le nom du joueur. Cette fonction récupère le score du joueur dans le dictionnaire et sépare le string au niveau du « / » pour récupérer la deuxième valeur, si le joueur a pioché un as sinon renvoie une conversion en entier du score.

### 2. Le nombre de carte initiale du croupier

Le croupier ne tire pas deux cartes mais une seule alors que généralement le croupier tire deux cartes dont une face cachée. Dans certains casino étranger, il en tire qu'une seule. J'ai donc décidé de lui en faire tirer une seule, afin de permettre l'assurance pour les joueurs joués par des utilisateurs via l'interface. Cela rajoute une nouvelle dimension au jeu et permet de minimiser les pertes si on joue bien.

## B. Les diagrammes avec matplotlib

Dans le but d'étudier les différentes stratégies implémentées, il a fallu tracer des graphiques. J'ai donc choisi de tracer la fréquence des scores obtenus sur  $n$  parties par les différents joueurs et l'évolution de la cagnotte pour voir qui est le joueur qui s'en sortait le mieux.

J'ai donc utilisé la bibliothèque matplotlib qu'on avait déjà utilisé durant certains TP. J'ai programmé des fonctions qui dans un dictionnaire associe pour chaque joueur une liste des scores selon les parties tant qu'il n'a pas perdu. Pour tous les joueurs de la table, on vérifie si la clé correspondante à son nom existe sinon on la crée et après on ajoute à la liste son score ou la valeur

actuelle de sa cagnotte pour tracer des diagrammes de fréquences pour les scores et des diagrammes de type « cagnotte en fonction de la partie ».

*voir les graphiques à la partie III. Étude des différentes stratégies*

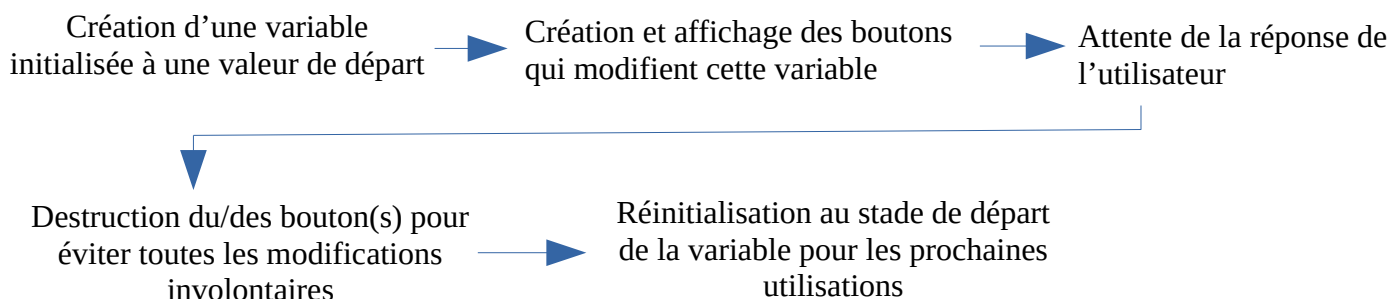
### *a) L'interface graphique avec tkinter*

L'interface graphique a été pour moi le plus gros challenge car je n'avais jamais utilisé tkinter. Je me suis quand même tourné vers ce module pour créer mon interface graphique car certains camarades auraient pu m'aider avec celui-là en cas de besoin. Je me suis finalement débrouillé avec des forums et des recherches sur internet.

La plus grande difficulté étant que l'interface graphique n'attend pas les inputs/entrées si on ne lui demande pas de les attendre. Le fait de créer des boutons oui et non à la question « Voulez-vous piocher ? », n'implique pas que le programme attende la réponse. Il a donc fallu utiliser une fonction « wait\_variable » qui est appelée depuis une variable de type bouton. La fonction « wait\_variable » demande au programme d'attendre que la variable en argument change d'état.

Une fois le problème résolu, j'ai pu remplacer tout mes inputs par des échanges via l'interface graphique. Il est intéressant de laisser les print() de la version sans interface graphique pour se servir de la fenêtre textuelle qui écrit ce qu'il se passe afin de vérifier qu'il n'y a pas de problème avec l'interface graphique.

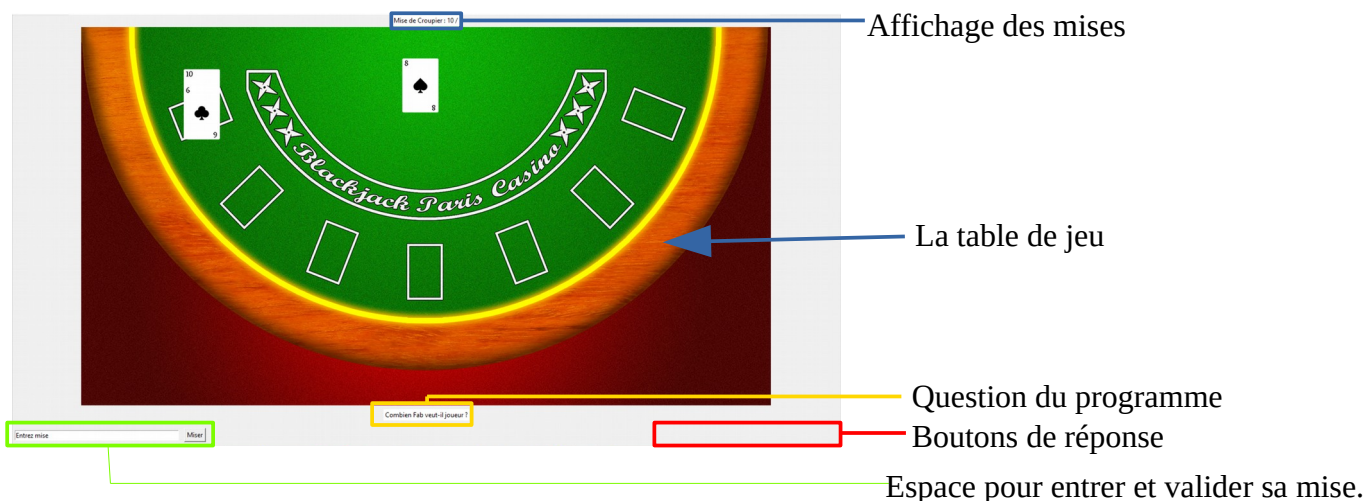
#### Principe de l'interface graphique (pour remplacer un input())



*L'initialisation/réinitialisation de la variable dépend du type de réponse attendue :*

- Si on attend que l'utilisateur rentre la mise pour continuer, on peut l'initialiser à False et mettre la variable à vrai si on clique sur le bouton, pour continuer le programme
- Si on attend une réponse de type booléenne Oui/Non, True/False on l'initialise à None et on récupère la réponse selon que le bouton Oui ou Non ait été cliqué
- Si on attend une réponse de type Piocher/Arrêter/Doubler, on modifie un string vide avec la valeur souhaitée selon le bouton

#### Disposition de l'interface graphique



## II. Les fonctionnalités supplémentaires implémenter

### A. Assurance

*Le croupier se trouve dans le dictionnaire croupier qui contient en fait toutes les IA. Il est le seul à s'appeler « Croupier » car il est initialisé en premier et qu'on ne peut pas prendre un nom déjà pris grâce à cause de la boucle while de filtrage dans initTable. Cela permet entre autres d'éviter les erreurs de clé car il ne peut pas avoir deux clés identique dans un dictionnaire sinon il peut renvoyer une valeur erroné.*

La possibilité de s'assurer contre un Black Jack du Croupier intervient au premier tour dans le cas où la première carte du croupier est un As et si le croupier à un « Black Jack », c'est à dire une bûche comme deuxième carte, alors sa mise est perdu mais son assurance rembourse les pertes. C'est donc dans la fonction premierTour et partieComplete que tout va se jouer. Il suffit de vérifier si la clé « Croupier » du dictionnaire CroupiersPart ( Croupiers Partie ) a pour score « 1/11 » qui signifie que la carte pioché au premier tour est un As. Après dans tourCroupier on vérifie si il a 21 au tour 2 et on met son score à « Black Jack » (c'est aussi pour ça qu'il confortable de manier un string pour les scores). En fin de partie, si le Croupier à un Black Jack tous les joueurs qui ont pris une assurance sont remboursés.

### B. Doubler

L'action doubler est très simple, c'est le fait de piocher une dernière carte en doublant sa mise mais elle est cependant très risqué. Lors du tour des joueurs, il suffit de rajouter une option qui soustrait une deuxième fois sa mise, de l'ajouter une deuxième fois au pot à remporter avant de doubler la valeur du joueur dans le dictionnaire « mise ». Il est important de la doubler en dernière sinon on lui retire une fois de trop sa mise.

### C. Diviser

*C'est à cause de la division qu'on filtre les espaces dans le nom des joueurs pour pouvoir formater les noms avec des espaces et récupérer les valeurs du numéro de jeu et le nom initiale du joueur.*

Le fait de diviser semble simple au casino mais deviens plus compliqué quand il faut la programmer. Le joueur a la possibilité de choisir de séparer son jeu en deux jeux si les deux cartes sont identiques à la couleur prête. Dans le cas du projet, on effectue une copie des joueurs de la table avant chaque partie, donc si le joueur décide de diviser. On peut créer un deuxième joueur au format « nom jeux n° x » avec le score de la deuxième carte et on met le score du joueur initial à la valeur de la première carte (les deux cartes ont une valeur identique donc peu importe quelle carte revient à quel jeu). On double sa mise de la même manière que pour l'action doubler. A la fin de la partie, on vérifie si il y a le mot « jeux » dans le nom du vainqueur pour retrouver le nom du joueur à qui restituer les gains.

J'utilise une fonction simple pour trouver le numéro du joueur, on cherche tous les joueurs qui commencent par le nom du joueur dont on veut diviser le jeu. On vérifie si il y a 3 mots grâce à la fonction split() enfin, on récupère le dernier caractère du nom qui est est donc le numéro x correspondant au jeu. On utilise une variable local initialisé à zéro et on assigne la valeur du numéro de jeu au max si elle est bien la maximum à ce moment de l'itération. On retourne la valeur maximale pour attribuer un numéro au prochain jeu.



### III. Étude des différentes stratégies

#### A. Les mises

Il y a différentes manières de miser cependant la meilleure façon de miser reste de miser le minimum car les gains dépendent des mises des autres et nos pertes dépendent de notre mise. On peut minimiser les pertes sans diminuer les gains en jouant toujours le plus bas possible. J'ai donc instauré une mise minimale de 10 kopecs.

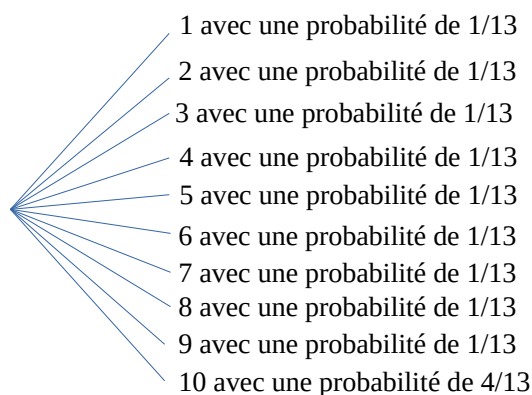
Pour la plupart des techniques, le nom est assez explicite :

1) On peut miser un pourcentage de notre cagnotte

2) On calcule une valeur de risque en fonction de la valeur du score. Si le risque est élevé alors on mise un pourcentage faible et inversement.

3) On calcule simplement la probabilité d'atteindre 21 ou moins simplement grâce à la fonction `probaXsur21(c,scores)` qui retourne cette probabilité. On doit obtenir la différence entre notre score et 21.

Exemple : Le joueur à 12 il lui faut 9 ou moins et il a donc 9 chances sur 13 d'avoir 9 au moins voir arbre de probabilité.



*On considère l'As comme une seule possibilité de score le 1 car si on pioche une As on peut toujours prendre la valeur 1 pour ne pas dépasser.*

4) La fonction `avantage` est la base des fonctions suivantes. On ajoute deux fonctions pour avoir le score maximum de la partie et le meilleur score (via `meilleurScore`) qui renvoie le score sous forme d'entier et qui gère le cas des As. On utilise la conversion Booléen vers Entier qui renvoie 1 si le joueur a l'avantage et 0 sinon. On l'utilise pour multiplier par 10 si elle a l'avantage ou sinon par 0. On mise donc  $10 + 0 \cdot 10 = 10$  ou  $10 + 10 \cdot 1 = 20$  kopecs.

5) On reprend le principe de la version `avantage` et si elle a l'avantage mise un pourcentage de sa cagnotte sinon la fonction renvoie 0. La fonction `croupier` récupère cette mise et la filtre si elle est inférieure à 10 pour la mettre à 10.

6) Le même principe mais combiné avec la probabilité. On réutilise la fonction `probaXsur21`.

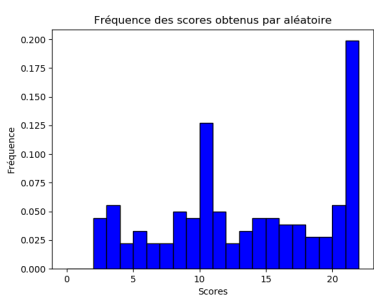
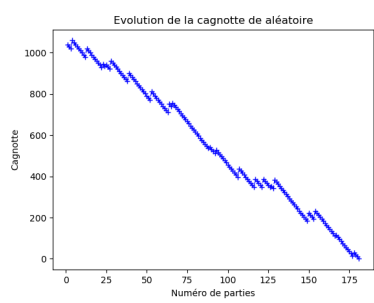
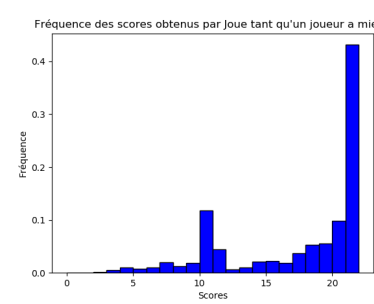
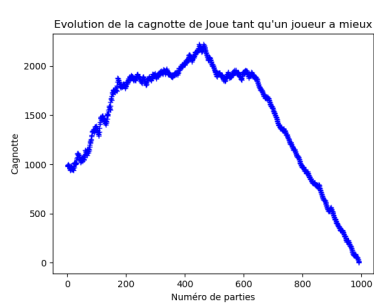
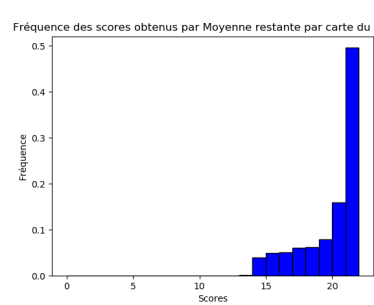
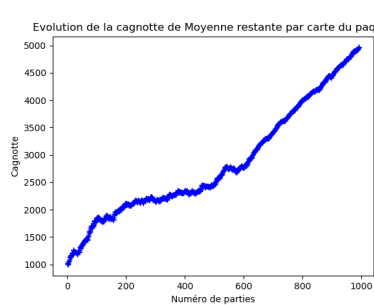
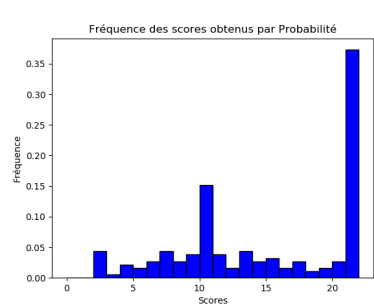
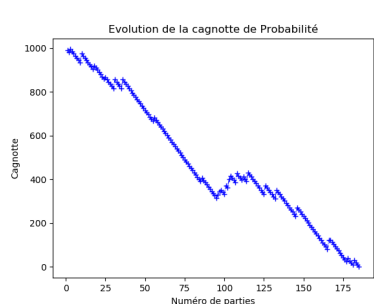
7) Un combiné des 4 premières fonctions, qui prend en compte tous les paramètres des 4 premières façons de miser.

8) Une manière de mise qui considère la valeur moyenne par carte restante dans la pioche par un calcul savant. Dans le cas où la valeur moyenne par carte est inférieure à la différence entre le score du joueur et 21 alors on mise en prenant en compte un pourcentage, la probabilité et l'avantage.

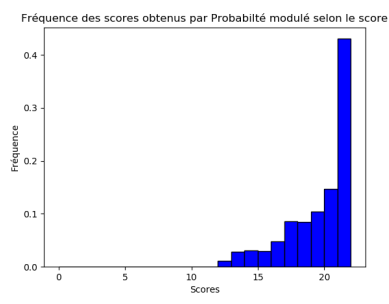
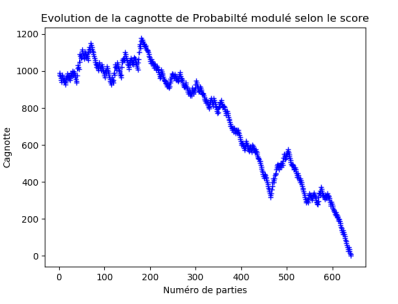
*L'avantage est utilisé dans la plupart des fonctions car à mon avis c'est une des valeurs les plus importantes car on va pas miser beaucoup si on ne mène pas au score.*

## B. Toutes les IA de pioche s'affrontent avec la même mise de 10

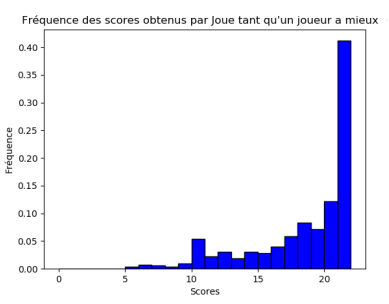
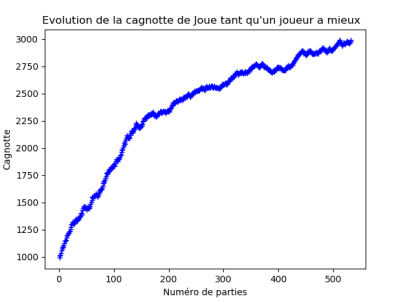
*Pour juger le vainqueur de la partie, on observe le graphe de l'évolution de sa cagnotte. Alors que pour juger de son efficacité, on regarde la fréquence de ses scores. On peut se fier à l'évolution de la cagnotte car toutes les IA mise 10 kopecs. On mise la mise minimal de 10 car miser plus augmente les risques de perte sans augmenter les risques de gains jouer le minimum est donc le meilleure choix. Il existe cependant différentes possibilités pour la mise des IA qui ne sont pas très intéressantes.*

<p>Fréquence des scores obtenus par aléatoire</p> 	<p>Evolution de la cagnotte de aléatoire</p> 	<p>Cette IA joue selon une variable aléatoire entre 1 et 2 si c'est 1 elle joue sinon elle se retire. On voit que jouer « à pile ou face » est une mauvaise stratégie. C'est évident que jouer sans prendre connaissance des données de la partie est une mauvaise idée.</p>
<p>Fréquence des scores obtenus par joue tant qu'un joueur a mieux</p> 	<p>Evolution de la cagnotte de joue tant qu'un joueur a mieux</p> 	<p>On voit que cette IA s'en sort jusqu'au moment où elle se retrouve seule contre « Moyenne restante du paquet »(partie n°500). Son jeu dépend uniquement des autres donc son « intelligence » dépend aussi des autres IA en jeu. On analysera donc une partie sans l'IA qui lui pose problème pour voir si elle s'en sort mieux.</p>
<p>Fréquence des scores obtenus par Moyenne restante par carte du paquet</p> 	<p>Evolution de la cagnotte de Moyenne restante par carte du paquet</p> 	<p>Cette IA est donc la plus performante, car on constate qu'elle a battu toute les autres sans jamais redescendre en dessous de son montant initiale. On voit aussi que 50 % de ses scores sont entre 14 et 21 ce qui est un bon résultat mais aussi qu'elle dépasse souvent 21.</p>
<p>Fréquence des scores obtenus par Probabilité</p> 	<p>Evolution de la cagnotte de Probabilité</p> 	<p>L'étude est la même que pour la version aléatoire. Choisir ce qu'on va faire sans prendre en compte aucunes données du jeu ne permet pas d'atteindre de bon résultats. Il y a 6 chances sur 10 de piocher donc ses scores son réparti mais il dépasse plus souvent 21 que aléatoire qui n'a qu'une chance sur deux de piocher une carte.</p>

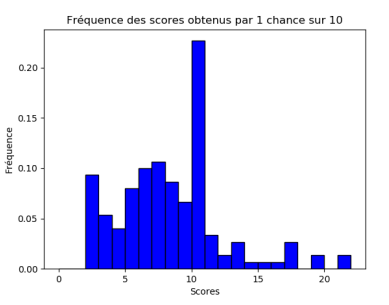
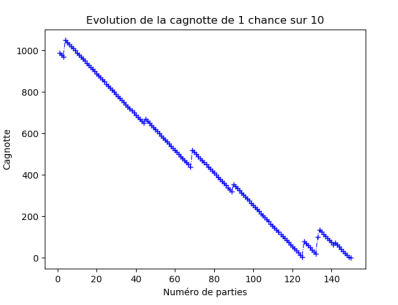
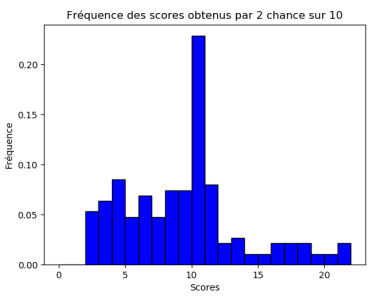
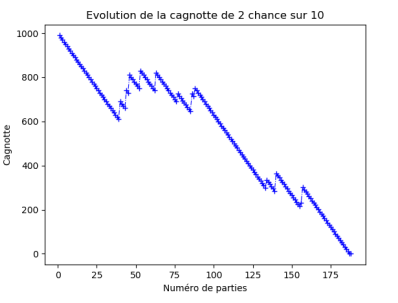


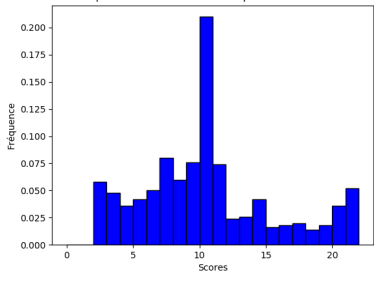
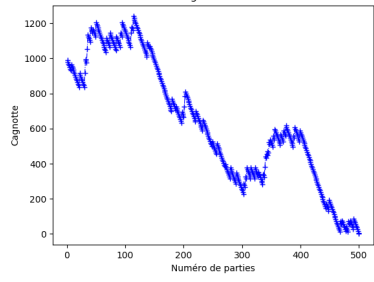
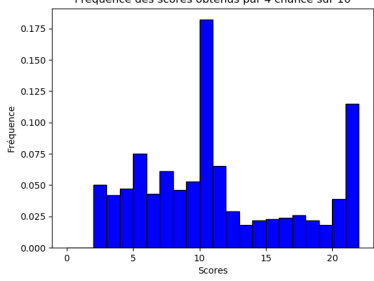
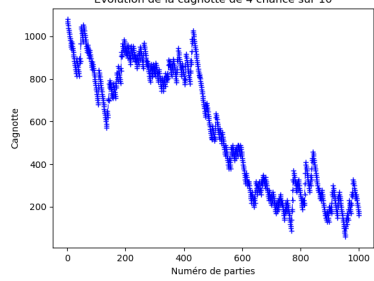
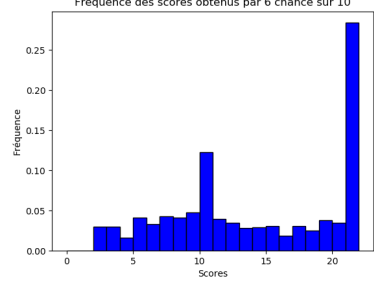
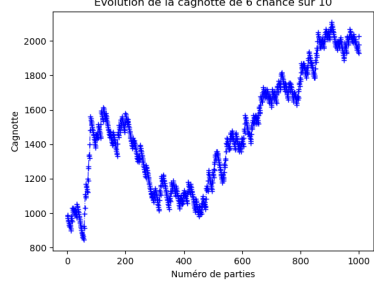
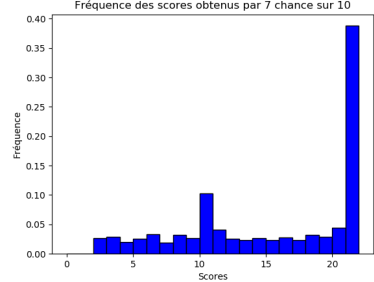
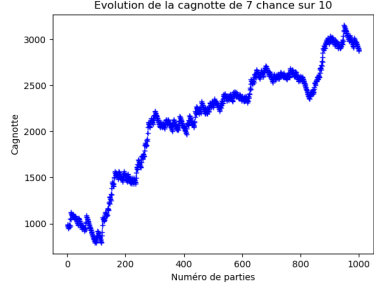
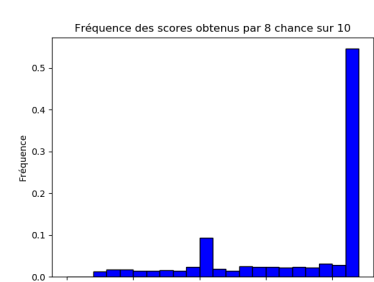
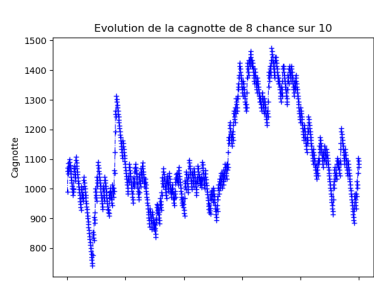
		<p>Cette IA a de bon résultat car plus son score est élevé, moins elle a de chance de piocher. Cependant elle perd quand même la partie contre l'IA qui prend en compte son score et la valeur moyenne de la pioche. C'est donc logiquement qu'elle s'incline car elle ne prend pas en compte les cartes restantes de la pioche mais seulement son score.</p>
---	---	---

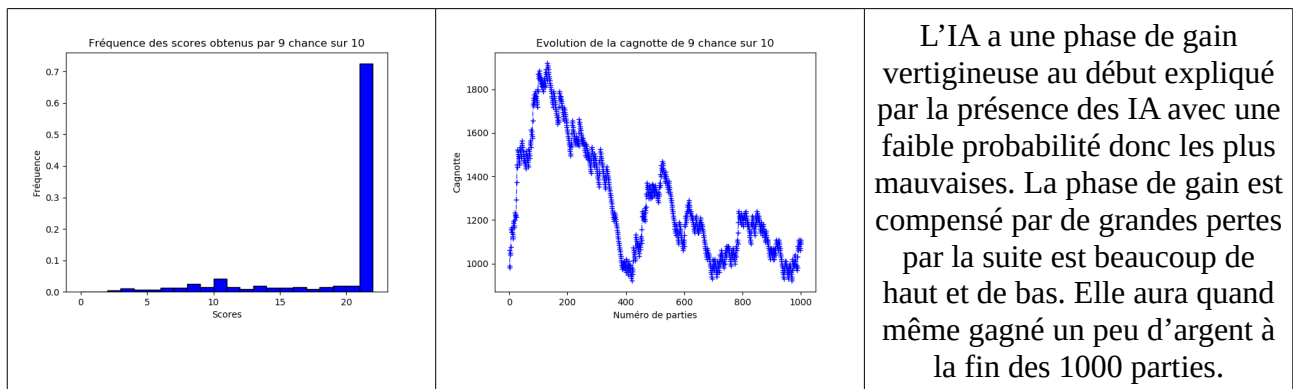
### C. Analyse de l'IA qui joue selon le score des autres

		<p>Lorsque qu'elle joue contre aléatoire et probabilité 0,6 , cette IA s'en sort mieux. Il est plus simple de gagner contre les IA aléatoire et probabilité car elle ne prennent pas de données du jeu pour savoir ce qu'elles doivent faire elles sont donc plus bêtes. On remarque bien que son jeu dépend des autres joueurs, car elle à gagné cette partie sans difficulté.</p>
---	---	---

### D. Meilleure probabilité aléatoire de pioche

		<p>La probabilité de piocher est très faible est donc son score dépend de ses premières cartes. Cela se reflètent sur une forte fréquence pour les scores faibles environ 80 % de scores inférieure ou égale à 10. Sans aucune surprise, on remarque qu'elle perd beaucoup de parties et qu'elle est très mauvaise.</p>
		<p>La probabilité de piocher est plus forte mais toujours insuffisante. On remarque que la fréquence de score supérieure à 10 est tout de même plus grande que pour une probabilité de 0,1 avec une fréquence de 25 % pour 11.</p>

<p>Fréquence des scores obtenus par 3 chance sur 10</p> 	<p>Evolution de la cagnotte de 3 chance sur 10</p> 	<p>On remarque que la fréquence des scores au dessus de 10 continue d'augmenter 50 % pour les scores supérieures ou égales à 10.</p> <p>On voit aussi une durée de vie plus longue avant de perdre tout son argent.</p>
<p>Fréquence des scores obtenus par 4 chance sur 10</p> 	<p>Evolution de la cagnotte de 4 chance sur 10</p> 	<p>Les scores sont de plus en plus équiprobable sauf pour le 10 qui est largement au dessus des autres.</p> <p>Cette IA se maintient bien tant que les IA avec une probabilité de 0,3 ou moins sont encore en jeu (partie n°500).</p>
<p>Fréquence des scores obtenus par 6 chance sur 10</p> 	<p>Evolution de la cagnotte de 6 chance sur 10</p> 	<p>On remarque trois phases :</p> <ul style="list-style-type: none"> <li>-Une phase de gains tant que les IA de probabilité 0,1 et 0,2 sont encore en jeu</li> <li>-Une phase de perte jusqu'à que l'IA de probabilité 0,3 soit sorti du jeu</li> <li>-Une nouvelle phase de gain après</li> </ul> <p>On se rapproche de la probabilité qui optimise le gain car elle fini les 1000 parties en doublant sa cagnotte initiale.</p>
<p>Fréquence des scores obtenus par 7 chance sur 10</p> 	<p>Evolution de la cagnotte de 7 chance sur 10</p> 	<p>On atteint selon moi ici le meilleur taux de gain. L'IA a des fréquences beaucoup mieux réparties, malgré beaucoup de dépassement.</p> <p>Par conséquent, il y a quand même beaucoup de scores faible compensé par les plus gros scores avec la même fréquence.</p>
<p>Fréquence des scores obtenus par 8 chance sur 10</p> 	<p>Evolution de la cagnotte de 8 chance sur 10</p> 	<p>Avec 8 chances sur 10 de piocher la probabilité de piocher est trop haute et on dépasse 21 plus d'une partie sur deux. Cette IA prend trop de risque.</p> <p>Cependant, on voit que sa cagnotte se maintient quand même.</p>



Pour moi, la meilleur probabilité pour une IA « aléatoire » est donc 0,7 car on voit que ses gains sont soumis à moins de fluctuations que les autres avec aussi une meilleure régularité dans les scores surtout sur les plus élevés. Elle arrive proche de 3000 kopecs à la fin des 1000 parties en ayant récolté la grande partie des cagnottes des joueurs éliminés. L'IA de probabilité 7 chances sur 10 de piocher reste inefficace contre toutes celles qui prennent en compte intelligemment les données du jeu.

#### IV. Conclusion

Le projet est maintenant avancé avec différentes possibilités d'intelligences artificielles. Une interface textuelle pour initialisé les IA, le croupier et les joueurs humains. La console textuelle sert aussi de « log » des parties où on voit exactement ce qui se passe. La console textuelle est plus claire que l'interface graphique, car je suis conscient qu'on ne comprend pas très bien quel jeu appartient à qu'elle personne. On pourrait améliorer cela en ajoutant un texte au dessus de l'image mais je n'ai pas trouvé comment le réaliser.

Je suis satisfait du résultat, du rendu visuel du programme et surtout d'avoir surmonté les différents problèmes que j'ai rencontré. J'ai bien aimé l'entraide présente au sein du groupe de TD qui n'a pu être que positive.