Yaniss LASBORDES

G2A

# SAÉ

# 1.02

**Compétence évaluée:**
Appréhender et construire des algorithmes.

**Apprentissages critiques :**
- Analyser un problème avec méthode
- Comparer des algorithmes pour des problèmes classiques

# Table des matières



# Rappel du sujet :

Dans cette seconde SAé, l'objectif est non seulement de réaliser (finaliser) une application complète qui réponde au sujet mais aussi de comparer plusieurs approches de résolution des différents problèmes à résoudre de façon à choisir la mieux adaptée. Des mesures de performance pourront être réalisées (après que la notion de performance aura été définie dans chaque cas).
Nous souhaitons généraliser la SAE 1.01 en faisant jouer l'ordinateur : les jeux doivent donc être jouables:

- humain contre humain (travail précédent)
- humain contre machine
- machine contre machine

On constate qu'il existe différentes stratégies pour informatiser le jeu "côté machine". Ces stratégies peuvent être très hasardeuses, très naïves ou très complètes voire complexes, allant du jeu "au hasard" à l'examen systématique de toutes les possibilités de jeu.

# Préparation :

Le premier problème avec créer des bots est de savoir comment les les plus hautes vont fonctionner donc je vais expliquer cette partie là.

## Devinette :

La logique pour la difficulté la plus basse et la difficulté la plus haute est assez simple;

La difficulté la plus basse va être complètement aléatoire et va juste enregistrer son maximum et minimum possible à chaque coup.

La difficulté la plus haute va utiliser la dichotomie pour trouver le bon chiffre.

Le problème était d'imaginer la difficulté moyenne et ayant fait des recherches je n'ai rien trouvé de mieux que de faire jouer le bot 1 fois sur 2 avec la dichotomie et 1 fois sur 2 aléatoirement. Le problème avec cette solution c'est que le bot moyen ne gagnera pas forcément plus mais sera seulement légèrement plus intelligent.

## Allumettes :

Je n'avais aucune idée de comment faire une grande difficulté pour le jeux des allumette alors j'ai fait mes recherche et j'ai trouver ce site qui m'a expliqué la stratégie pour toujours gagner en commençant sur les allumettes, celle ci étant de "viser" les chiffre 1, 5, 9, 13, 17 pour empêcher le joueur d'en face d'atteindre 1. Et s' il ne commence pas le bot a juste a essayer de faire en sorte que les allumettes soit égale à 1, 5, 9, 13, 17 avant le tour du joueur d'en face. Donc pour résumer il est imbattable si il commence et très intelligent si il ne commence pas.

Pour la difficulté moyenne j'ai utiliser la même logique que le jeu des devinette et il joue une fois sur deux aléatoirement et une fois sur deux avec la meilleur logique

Et puis pour la difficulté basse, le bots joue juste aléatoirement.

## Morpion :

J’avais une idée pour la difficulté max du morpion qui était de bloquer et de gagner quand le bot pouvait, mais il me manquait la partie quand il ne pouvait faire aucun des deux. Je me suis renseigné et il faudrait trouver tout les possibilités de gagner a chaque coup et je n’aimais pas cette solution pour deux raison; la premiere etant que ca va être BEAUCOUP plus lent d'exécuter des tests conséquent et la deuxieme etait que je pense pas etre assez a l’aise avec le Python pour pouvoir faire ca seul.

Pour la difficulté moyenne, même logique, une chance sur deux de jouer correctement ou jouer aléatoirement.

Et pour la difficulté basse, même logique, joue complètement aléatoirement.

## Puissance 4 :

Malheureusement je n’ai pas réussi à faire plusieurs difficultés pour le Puissance 4 car:

- Je manquais de temps les jeu d’avant on présenter des problème dont j’ai prit du temps à corriger
- La logique Puissance 4 est assez complexe à cause du fait que je dois prendre en compte la “gravité” des pions ce qui rend la chose extrêmement complexe.

Donc la seule difficulté est de l'aléatoire complet.

# Code :

Le premier problème présenté était d’identifier quand un joueur est un bot, j’ai décidé d’utiliser “\t{difficulté}{numéro}” car le tab ne pouvait pas être utiliser dans un pseudo et est la touche pour quitter donc elle ne pouvait pas être utilisé pour autre chose par l'utilisateur et {numéro} symbolise le fait que si deux bot on la même difficulté il faut que je les différencie donc le deuxième aura 2 a la fin de son pseudo.

## Bibliothèque externe à mon projet :

### Random :

Random est une bibliothèque qui permet de générer des nombre et autres aléatoirement, je n’ai utilisé que la fonction randint() dans mon programme qui prend en paramètre deux chiffres et génère un nombre aléatoire entre les deux (les deux compris).

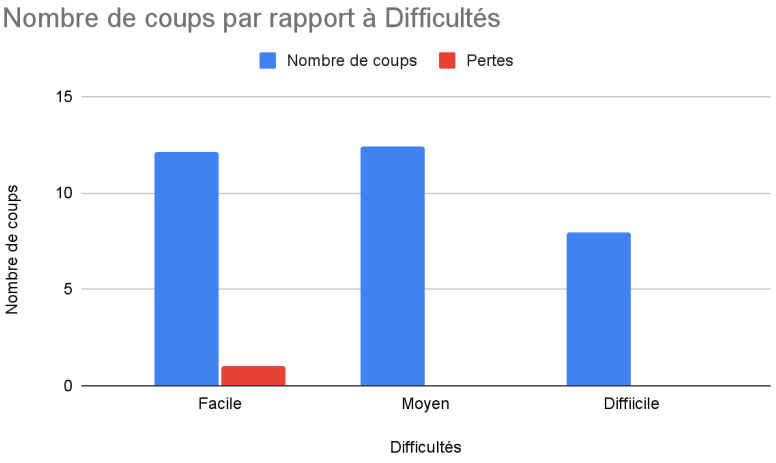
**Pour des informations plus précises il faut aller voir le code qui est entièrement commenté.**

# Jeu de tests :

## Devinette (sur 1 000 parties) :

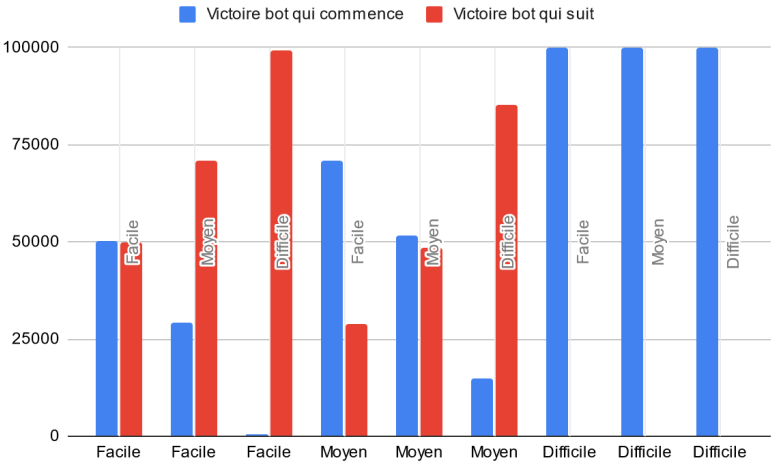
Comme vous pouvez le voir la difficulté deux fait plus de coups sur une moyenne de 1000 parties que la difficulté 1 mais elle ne peut pas perdre parce que la difficulté 1 à 1/1000 de perdre.

La difficulté 3 est bien plus optimisée parce qu'elle utilise la dichotomie.




## Allumettes (sur 100 000 parties) :

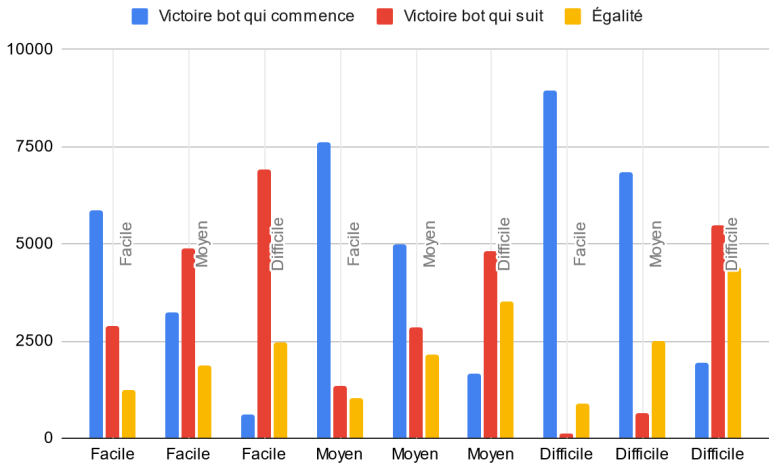
Comme vous pouvez le voir chaque difficulté est supérieur à la précédente, a noter que la difficile est imbattable quand elle commence, aussi la facile peut battre la difficile si elle joue parfaitement mais les chance que ca arrive sont d'environ 0.0041% ($(\frac{1}{3})^5$ car il faut faire 5 fois 3 allumettes pour battre difficile), en l'occurrence sur 100 000 facile ne gagne que 631 fois soit environ 0.0061% du temps.



## Morpion (sur 10 000 parties) :

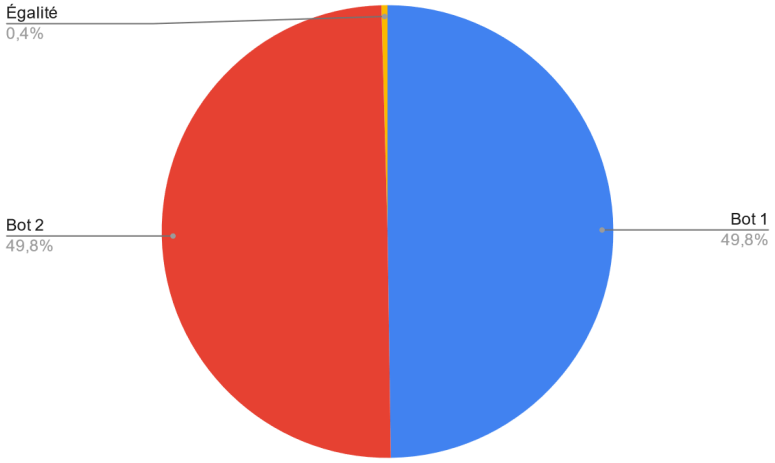
Il y a un problème avec ce graphique que je n'ai pas pu régler dans le code, qui est que difficile contre difficile, celui qui commence perd énormément sans raison apparente.

Sinon le reste est tout ce qui a de plus, chaque difficulté a du mal contre la difficulté supérieur et la difficulté 3 est la meilleure




## Puissance 4 (sur 100 000 parties) :

Avec le fait que la puissance 4 est aléatoire, les chiffres sont équivalents, on peut noter qu'il y a beaucoup moins d'égalité que sur les autres jeux, vu la difficulté et chance pour faire un puissance 4 plein sans faire 1 ligne de 4.



# Potentiel problèmes de mon programme :

Les mêmes problèmes que la SAE 1.01 car je n'ai pas forcément eu le temps de les régler ou je ne suis pas en capacité de les régler :

- Le centrage de certaines parties reste a désiré mais je n'ai pas réussi à trouver de solution dans le temps imparti.
- La gestion des points est assez basique.
- Le code est répété pas mal de fois malgré le fait que j'ai essayé d'éviter ça un maximum du temps.
- L'affichage, malgré le fait qu'il soit adaptable, fait des problèmes avec certaines tailles de terminaux (haut mais fin par exemple).
- Mon programme **N'EST PAS** compatible avec Windows, il ne fonctionne que sur des machines Unix parce que "termios" est une librairie standard d'Unix, aussi j'utilise la commande "clear" n'existe pas sous Windows, par ailleurs j'aurais pu palier à ce problème en utilisant "clear || cls" ("cls" étant l'équivalent Windows de "clear") mais comme mon programme ne démarre pas a cause de termio sous windows j'ai décidé de ne pas le faire.

Et maintenant les probleme de mon programme durant la SAE 1.02:

- Les bots du Morpion ne sont pas les meilleurs du tout.
- Le code fait à la SAE 1.01 n'était pas adapté à avoir un aussi gros ajout donc le code est vraiment moins lisible que la première SAE selon moi.
- Les difficultés moyennes restent à désirer.
- Les bots Puissance 4 inexistants.

# Conclusion :

Au début, cette SAE me paraissait extrêmement dure comparé à la première et aussi par rapport à comment j'avais coder la première.
J'ai donc appris qu'il faut que j'apprenne à faire un code modulable en prévision de changements potentiels.
J'ai aussi appris à coder des "IA", ce que je ne me pensais pas capable de faire et que je n'avais jamais essayé.