Université Paris Dauphine-PSL

Projet Monte Carlo

Auteurs :
Elise Chin
Mathilde Da Cruz

6 octobre 2022

Table des matières

1 Contexte

1.1 Objectif

L'objectif de ce projet est d'appliquer un algorithme de recherche Monte Carlo à un jeu ou à un problème. Le projet consiste à faire des expériences pour résoudre le problème et comparer les algorithmes.

1.2 Choix d'un jeu

Pour trouver un jeu, nous avons choisi de nous diriger vers des jeux dits de statégie combinatoire abstrait [1] En effet, afin de réduire le champ des possibles, nous avons souhaité nous diriger vers un jeu répondant aux critères suivants :

— Jeu jouable à deux joueurs

— Les joueurs jouent à tour de rôle

— Entièrement observable

En revanche, nous avons pensé qu'un jeu pas entièrement déterministe (utilisant par exemple un lancer de dés à certains moments) pourrait également être envisageable.

Enfin, nous avons souhaité essayer de résoudre un jeu "original", sur lequel nous ne trouvions pas de résolutions existantes par des algorithmes de Monte Carlo.

Nous avons donc pensé au jeu **Marrakech**, qui est un jeu de société paru en 2008, pour 2 à 4 joueurs sur un plateau de taille 7×7. Nous nous restreignons pour le moment à la version à 2 joueurs, bien qu'une version à 3 ou 4 joueurs pourrait tout à fait être utilisée.

Branching factor. Les algorithmes de Monte Carlo sont particulièrement efficaces pour des jeux avec un branching factor assez élevé. Dans notre cas, à chaque coup, on peut choisir parmi $4 \times 49 \times 12 = 2352$ coups (4 orientations, 49 placements possibles du pion, 12 emplacements de tapis différent).

2 Description du jeu Marrakech

Résumé des règles du jeu : `https://www.jeuxavolonte.asso.fr/regles/marrakech.pdf`

FIGURE 1 – Plateau du jeu Marrakech

Idée du jeu - Extrait des règles officielles. Le marché aux tapis de la place de Marrakech est sur les dents : on va bientôt désigner le plus habile des marchands. Chacun d'entre eux va tenter d'avoir le plus grand nombre de ses tapis exposés en fin de partie tout en accumulant la plus grande fortune. Le plus fortuné (cumul des tapis visibles et de la somme détenue par chacun) a gagné.

2.1 Principe général

Marrakech est un jeu qui se joue sur un plateau de taille 7×7, avec un pion orienté commun (du nom d'Assam) pour tous les joueurs, un dé à 6 faces (avec des valeurs particulières), des tapis indiquant la conquête d'une ou deux cases du plateau, et des pièces.

Le gagnant est le joueur qui possède le plus de points à la fin de la partie, lorsqu'il n'y a plus de tapis disponibles, donc après un certain nombre de tours puisqu'un joueur pose un tapis par tour.

Le nombre de points d'un joueur est la somme de :

— Le nombre de pièces qu'il possède

— Le nombre de cases du plateau qu'il "possède", c'est-à-dire le nombre de cases du plateau sur lesquels le joueur possède un tapis non recouvert. On peut également voir cela comme le nombre de cases de sa couleur.

En cas d'égalité, c'est le joueur qui a la plus grand nombre de pièces qui gagne.

1. `https://fr.wikipedia.org/wiki/Jeu_de_strat%C3%A9gie_combinatoire_abstrait`

2.2 Déroulement d'un tour

Lors de son tour, un joueur va, dans l'ordre suivant :

— Orienter le pion

— Déplacer le pion en ligne droite du nombre de cases indiqué par le dé au tour précédent

— Si le pion termine son déplacement sur un tapis adverse, il devra donner un certain nombre de pièces à cet adversaire

— Placer un tapis de sa couleur

— Lancer le dé

Le joueur a donc 2 décisions à prendre : comment orienter le pion et où placer son tapis.

Si le joueur est le premier à jouer pour le premier tour et qu'aucun dé n'a été lancé avant, il ne bouge simplement pas.

Dans la version originale du jeu, le lancer de dé se fait par le joueur en cours après l'orientation du pion. Dans le but de gérer cette part d'aléatoire, nous avons décidé de faire la modification suivante : lancer le dé qu'à la fin du tour pour le joueur suivant. Ainsi, le mouvement d'un joueur est défini par une seule action, composée de 2 étapes (mais sans aléatoire entre les deux), c'est-à-dire un couple (orienter, placer un tapis).

2.3 Mise en place

Sur le plateau 7×7, le pion est placé au centre du plateau orienté au Nord. Dans la version à deux, chaque joueur possède 30 pièces, 12 tapis d'une couleur, et 12 tapis d'une autre. Par exemple le joueur 1 aurait 12 tapis rouges et 12 tapis orange, tandis que le joueur 2 aurait 12 tapis verts et 12 tapis bleus. Concrètement, chaque joueur possède 30 points, et le jeu va durer 24 tours, puisqu'un tapis est posé par tour et par joueur.

2.4 Précisions supplémentaires

2.4.1 Comment peut-on orienter le pion ?

Il est possible de :

— Laisser le pion dans son orientation

— Le tourner d'un quart de tour à droite

— Le tourner d'un quart de tour à gauche

— Il est interdit de faire un demi tour

2.4.2 Description du dé

Le dé à 6 faces est composé des valeurs suivantes : 1-2-2-3-3-4

2.4.3 Comment déplacer le pion lorsqu'il sort du plateau ?

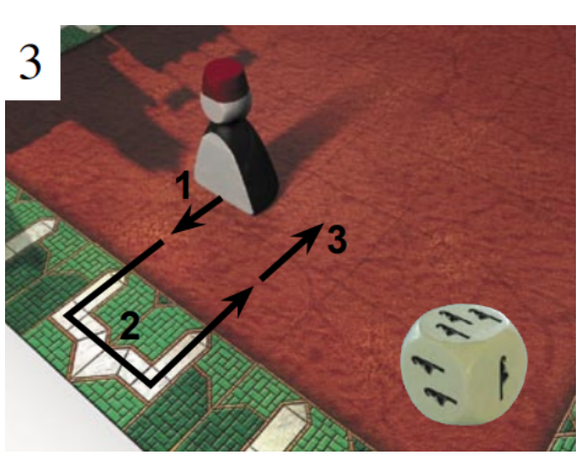


FIGURE 2 – Si le pion sort du plateau, il suit le demi-tour indiqué par les flèches (les flèches ne comptent pas comme un point de déplacement

Des règles précises indiquent sur quelle case et dans quelle orientation doit se trouver le pion après être sorti du plateau. Sur le plateau du jeu "réel", ces règles sont indiquées par des chemins non orientés. Il existe un unique chemin par case de sortie. Si une règle indique que lorsque le pion sort par la case A dans une orientation x, il rentre par la case B dans une orientation y, il y aura également une règle "contraire" indiquant que lorsque le pion sort par la case B dans l'orientation opposée à y, il devra rentrer par la case A dans l'orientation opposée à x.

Très concrètement, en considérant le plateau par coordonnées ((0,0) en bas à gauche et (6,6) en haut à droite) et en utilisant N-S-O-E pour indiquer les directions, les règles sont les suivantes (en faisant le tour du plateau en partant d'en bas à gauche dans le sens anti-horaire) :

— Sortie (0,0)S → Entrée (0,0)E
et inversement :
Sortie (0,0)O, → Entrée (0,0)N

— Sortie (1,0)S → Entrée (2,0)N
et inversement :
Sortie (2,0)S, → Entrée (1,0)N

— Sortie (3,0)S → Entrée (4,0)N
et inversement :
Sortie (4,0)S, → Entrée (3,0)N

— Sortie (5,0)S → Entrée (6,0)N
et inversement :
Sortie (6,0)S, → Entrée (5,0)N

— Sortie (6,0)E → Entrée (6,1)O
et inversement :
Sortie (6,1)E, → Entrée (6,0)O

— Sortie (6,2)E → Entrée (6,3)O
et inversement :
Sortie (6,3)E, → Entrée (6,2)O

— Sortie (6,4)E → Entrée (6,5)O
et inversement :
Sortie (6,5)E, → Entrée (6,4)O

— Sortie (6,6)E → Entrée (6,6)S
et inversement :
Sortie (6,6)N, → Entrée (6,6)O

— Sortie (5,6)N → Entrée (4,6)S
et inversement :
Sortie (4,6)N, → Entrée (5,6)S

— Sortie (2,6)N → Entrée (3,6)S
et inversement :
Sortie (3,6)N, → Entrée (2,6)S

— Sortie (0,6)N → Entrée (1,6)S
et inversement :
Sortie (1,6)N, → Entrée (0,6)S

— Sortie (0,6)O → Entrée (0,5)E
et inversement :
Sortie (0,5)O, → Entrée (0,6)E

— Sortie (0,4)O → Entrée (0,3)E
et inversement :
Sortie (0,3)O, → Entrée (0,4)E

— Sortie (0,2)O → Entrée (0,1)E
et inversement :
Sortie (0,1)O, → Entrée (0,2)E

2.4.4 Combien de pièces à payer lorsqu'on arrive sur un tapis adverse ?

Si lors de notre tour, le pion termine sur une case appartenant à un adversaire, un transfert d'argent a lieu. Le montant est égal au nombre de cases adjacentes et de la même couleur à la case sur laquelle le pion se trouve. Le joueur n'a rien à payer s'il termine sur une case vide ou sur un de ses tapis.

FIGURE 3 – Le montant à payer est équivalent au nombre de cases concernées. Les cases doivent avoir un côté commun ; les cases en diagonale ne comptent pas ; la case du pion entre dans le décompte

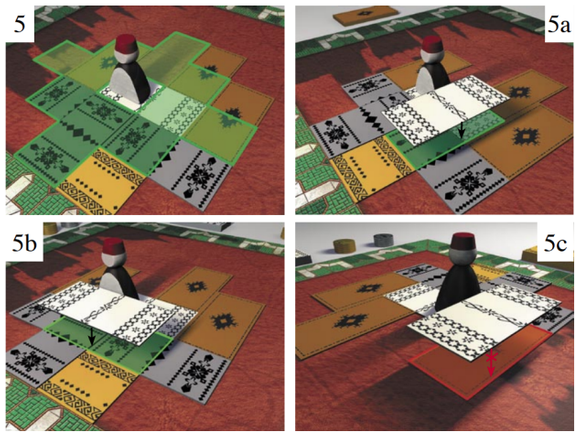


FIGURE 4 – Possibilités pour placer un tapis

2.4.5 Comment poser un tapis ?

Un tapis fait la taille de 2 cases. Le tapis doit être placé autour du pion, c'est-à-dire que une des 2 cases qui seront recouvertes par le tapis doit être directement adjacente à la case sur laquelle se trouve le pion.

Un tapis peut être posé sur :

— deux cases vides

— une case vide et une moitié de tapis (Figure 5a)

— deux moitiés de tapis différents (même s'ils sont de la même couleur) (Figure 5b)

Cependant, le tapis ne peut pas recouvrir entièrement un unique tapis (Figure 5c).

3 Adaptations et modélisation

Pour pouvoir appliquer des algorithmes de Monte Carlo à ce jeu, nous avons du procéder à plusieurs adaptations, notamment :

— Les calculs prenant trop de temps pour la version originale du jeu (c'est-à-dire avec un plateau 7×7 et 48 tours), nous avons adapté le jeu pour un plateau 5×5 et 32 tours (8 tapis par couleur).

— Afin qu'il n'y ait pas d'aléatoire dans un mouvement, nous avons choisi d'effectuer le lancer de dé AVANT le tour d'un joueur. Ainsi, le joueur n'a plus qu'à choisir, sachant le résultat du dé, l'orientation du pion et où placer son tapis.

Nous avons créé différentes classes afin de représenter tous les détails du jeu.

3.1 La classe Position

Un objet `Position` représente la position du pion ou d'un demi tapis. Cette classe a surtout pour utilité de rendre le code plus lisible.

3.2 La classe Rug

Un objet de la classe `Rug` représente un tapis. Puisqu'il est interdit de poser un tapis sur deux cases déjà recouvertes par un même tapis, il est notamment important de garder un id pour chaque tapis en plus de sa couleur afin de différencier deux tapis de même couleur.

3.3 La classe Pawn

Un objet de la classe `Pawn` représente le pion du jeu, qui est caractérisé par son orientation et sa position. Le pion peut notamment se déplacer et calculer le nombre de pièces qu'il doit à son adversaire s'il arrive sur un de ses tapis.

3.4 La classe Player

Un objet de la classe `Player` représente un joueur et permet de garder le compte de ses pièces ainsi que le nombre de tapis qui lui reste à jouer.

3.5 La classe Move

Un objet de la classe `Move` représente un mouvement du jeu. Il possède en particulier des méthodes pour savoir si un mouvement est légal ou non.

3.6 La classe Board

Un objet de la classe `Board` représente le plateau de jeu, et donc l'état du jeu à un instant t. Il possède notamment les méthodes les plus importantes, celle qui permet d'avoir la liste des mouvements légaux, celle qui permet de calculer le score, et celle qui permet de faire une partie aléatoire (`playout`).

4 Algorithmes de sélection des coups

Nous implémentons plusieurs stratégies. Nous avons également pensé à appliquer une variante sur ces stratégies.

4.1 Variantes appliquées aux algorithmes

A la fin d'une partie, le gagnant est le joueur qui a la plus de points. Pour calculer le score, nous considérons donc pour chaque joueur la somme de ses pièces et de cases du plateau recouvertes par un de ses tapis (non recouverts). Cela peut donc permettre d'évaluer une victoire : on peut supposer qu'un joueur qui a gagné la partie avec 60 points d'avance a été meilleur qu'un joueur qui a gagné une partie avec 5 points d'avance.

Le score d'un jeu est le score du premier joueur moins le score du second joueur, ce qui nous permet de dire à la fin que, pour un score s :

— Si $s > 0$: le joueur 1 a gagné avec s points d'avance

— Si $s < 0$: le joueur 2 a gagné avec -s points d'avance

— Si $s = 0$: il y a égalité.

Nous avons donc voulu essayer dans les algorithmes une variante qui, plutôt que de favoriser les mouvements qui font gagner le plus souvent, favoriser les mouvements qui font obtenir le plus de points en moyenne.

Pour cela, lorsque dans la version classique d'un algorithme nous ajouterions au compteur d'un mouvement 1 pour une victoire et zero pour une défaite, nous ajoutons à la place le score final du jeu. Autrement dit, si le joueur a gagné, on va lui ajouter une certaine quantité positive, mais s'il a perdu on va lui retirer des points.

L'idée est :

— Entre deux mouvements qui ont fait gagné la même proportion de fois, favoriser le mouvement qui a donné un meilleur score moyen

— De la même manière, on préfèrera un mouvement qui fait perdre de peu

— Il sera également possible de préférer un mouvement qui a légèrement un moins bon taux de victoire, mais qui, lors des défaites, fait perdre moins de points (plutôt qu'un meilleur taux de victoire mais des défaites larges et des victoires courtes).

Pour certains algorithmes, il est possible d'ajouter les points de manière brute (par exemple pour flat monte carlo, il suffit de faire une moyenne des scores pour évaluer un mouvement), mais pour d'autres, comme UCB et UCT, il faudra évaluer un noeud différemment, et il faudrait donc que le score d'un noeud reste entre 0 et 1.

Dans notre cas, le score maximum est 60 (le joueur 1 a récupéré toutes les pièces du joueur 2) + 5×5 (un point par case du plateau recouverte par un de ses tapis), donc 85 points. Néanmoins, les scores extrêmes sont assez rares. Nous avons donc décidé de normaliser le score de la manière suivante :

— Tout d'abord on clip le score : si le premier joueur gagne avec plus de 40 points d'avance, on ramène le score à 40 et s'il perd avec plus de 40 points de retard, on ramène le score a -40. En effet le but principal étant de différencier les victoires/défaites courtes des victoires/défaites larges, il suffit de considérer un espace plus petit et de ramener les scores extrêmes à une victoire parfaite (1) ou un défaite totale (0)

— Ensuite on ramène le score entre 0 et 1, c'est-à-dire qu'on projette le segment [-40, 40] dans [0,1].

Le score normalisé est donc

$$normalized = \frac{clip(score) - (min(score))}{max(score) - (min(score))}$$

Une hypothèse que nous faisons est que cette stratégie pourra aider à différencier plusieurs "meilleurs coups", en particulier pour un nombre de playouts faible.

4.2 Random

Nous utilisons la stratégie aléatoire comme baseline. Très simplement, à chaque coup, l'IA choisit un coup aléatoire parmi les coups légaux.

4.3 Flat Monte Carlo

Avant chaque playout, l'IA sélectionne un coup au hasard. Le meilleur coup sera celui qui a donné le plus de victoires en moyenne.

Dans un TP fait en cours, nous avions implémenté cette stratégie en faisant n playouts pour chaque coup légal (donc un total de n x nombre de coups légaux = environ 20n à chaque tour), puis en jouant le coup qui a fait remporter la partie le plus souvent. Ici, nous avons choisi de garder la version "originale", c'est-à-dire en sélectionnant le coup à explorer aléatoirement, tout simplement pour pouvoir comparer cette stratégie avec un autre algorithme qui utilisera le même nombre de simulations.

Pour cette stratégie, nous avons également implémenté la variante qui prend en compte le score moyen plutot que le taux de victoire.

4.4 UCB

UCB permet de faire un compromis entre l'exploration des branches incertaines et l'exploitation des branches les plus prometteuses.

Avant chaque playout, on sélectionne le coup qui maximise un score, comme dans le problème des bandits manchots où chaque joueur doit choisir la machine qui maximise leur récompense.

A la fin, le meilleur coup choisi par l'IA sera celui qui a été joué le plus de fois.

4.5 UCT

UCT (Upper Confidence Bound applied to Trees) est une variante de l'algorithme MCTS (Selection, Expansion, Simulation, Backpropagation) qui tire parti de la formule UCB dans la phase de sélection afin de choisir le noeud le plus prometteur. A la différence de l'algorithme précédent UCB où l'on part d'une configuration initiale et on détermine le noeud le plus prometteur, l'implémentation d'UCT applique la formule d'UCB à chaque noeud étendu de l'arbre de jeu jusqu'à atteindre une feuille.

Pour implémenter UCT, nous avons besoin de garder en mémoire pour chaque configuration du jeu, le nombre total de playouts où apparaît la configuration, une liste du nombre de playouts et une liste du nombre de parties gagnées après chaque mouvement à partir de cette configuration. En pratique, on crée un dictionnaire (`Table`) indexé par une configuration du jeu représenté par un entier. Cet entier est déterminé par le hachage de Zobrist, qui consiste à effectuer le XOR des nombres aléatoires associés à chaque case du plateau pour la configuration en question. L'état d'une case est déterminé par :

1. la présence ou l'absence du pion Assam
2. l'orientation du pion
3. le résultat du dé
4. l'abscisse de la case
5. l'ordonnée de la case
6. la couleur du tapis posée sur la case, ou l'absence de tapis
7. le joueur durant le tour
8. la couleur du tapis durant le tour

Les nombres aléatoires sont gardés dans une table de transposition, implémenté par un dictionnaire que l'on ajoute dans la classe `Board`. Chaque sous-état d'une case (de 1 à 6) est lui-même représenté par un dictionnaire. Les deux sous-états 7 et 8 sont représentés indépendemment du dictionnaire.

```
self.hashTable = defaultdict()
    for pawn in ['assam', 'no_assam']:
        self.hashTable[pawn] = defaultdict()
        for orientation in [NORTH, SOUTH, EAST, WEST]:
            self.hashTable[pawn][orientation] = defaultdict()
            for dice_result in [1, 2, 3]:
                self.hashTable[pawn][orientation][dice_result] = defaultdict()
                for x in range(5):
                    self.hashTable[pawn][orientation][dice_result][x] = defaultdict()
                    for y in range(5):
                        self.hashTable[pawn][orientation][dice_result][x][y] =
                        ↪  defaultdict()
                        for rug_color in [RED, BLUE, PINK, GREEN, EMPTY]:

                            ↪  self.hashTable[pawn][orientation][dice_result][x][y][rug_color]
                            ↪  = random.randint(0, 2**64)

self.hashTurn = random.randint(0, 2**64) # hash value for changing player
self.hashColor = random.randint(0, 2**64) # hash value for changing rug color
```

Pour calculer le code d'une configuration du plateau, il suffit alors de faire le XOR :

— du déplacement du pion (nombre aléatoire associé à l'enlèvement du pion d'une case (x, y) avec le nombre aléatoire associé à l'ajout du pion dans une nouvelle case (x', y')))

— du placement du tapis sur deux cases adjacentes au pion

— du changement de joueur pour le prochain tour

— du changement de couleur de tapis associé au prochain joueur

5 Expérimentations et résultats

Les temps de calculs étant assez longs, nous ne pouvions pas faire de test pour beaucoup de parties et beaucoup de playouts. Nous avons implémenté 7 stratégies : Random, Flat, Flat Score, UCB, UCB Score, UCT, UCT score. Nous avons effectué les expérimentations suivantes :

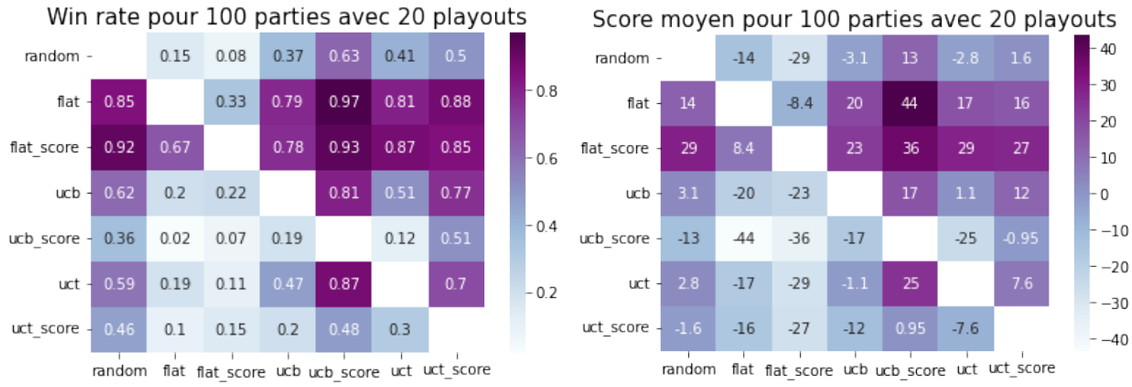
— Un tournoi faisant se rencontrer toutes nos stratégies sur 100 parties et seulement 20 playouts, soit 21 matchs au total, pour une durée d'environ 3h30.

— Un tournoi faisant se rencontrer toutes nos stratégies (sauf random et UCT Score, pour économiser du temps) sur 50 parties et 500 playouts, soit 10 matchs au total, pour une durée d'environ 22h.

— Un match de chaque stratégie (sauf le random), contre le random, sur 20 parties et pour différents nombres de playouts.

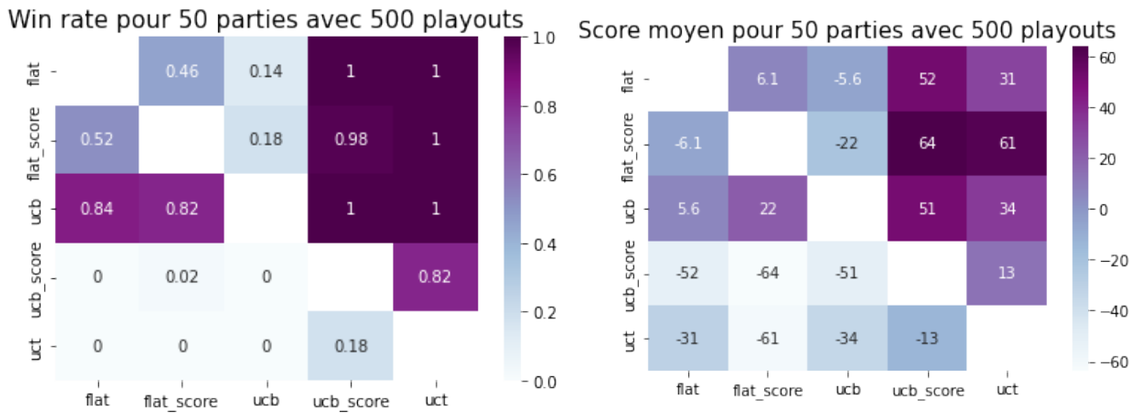
5.1 Résultats pour 100 parties avec 20 playouts

Nous représentons ci-dessous les résultats du tournoi entre chacune de nos stratégies. Les tableaux se lisent de la manière suivante : la stratégie de la ligne a gagné x% du temps contre la stratégie de la colonne (tableau de gauche) avec un score moyen de y (tableau de droite). Par exemple, la stratégie flat a gagné 85% du temps contre la stratégie random avec un score moyen de 14 points.



Avec seulement 20 playouts, les stratégies flat sont bien meilleures que les autres, et atteint même 97% contre la stratégie UCB score. Cela s'explique par le fait qu'à chaque tour, il y a environ 20 coups légaux, ainsi, UCB n'a pas le temps d'exploiter les coups prometteurs et a donc peu de chances de choisir le meilleur coup. On remarque également que flat avec la variante du score a gagné 67% du temps contre flat classique, ce qui semble confirmer que pour un petit nombre de playouts, la variante score donne un avantage. D'ailleurs, on note également que flat score réalise effectivement de meilleurs scores moyens que la version classique. En revanche, la variante score de UCB n'est pas du tout performante. Quant à la stratégie UCT, elle perd contre flat, flat score et UCB ; elle gagne très légèrement contre la stratégie random, mais largement contre la variante score d'UCT. Par contre, la stratégie d'UCT score est moins performante que les autres stratégies de manière générale : elle perd notamment contre la stratégie aléatoire et UCB score. Ces résultats sont surprenants car nous nous attendions à de meilleures performances puisque la stratégie UCT applique la formule d'UCB à chaque noeud étendu dans l'arbre de jeu, donc théoriquement par récursivité, le meilleur coup est mieux choisi que dans la stratégie UCB.

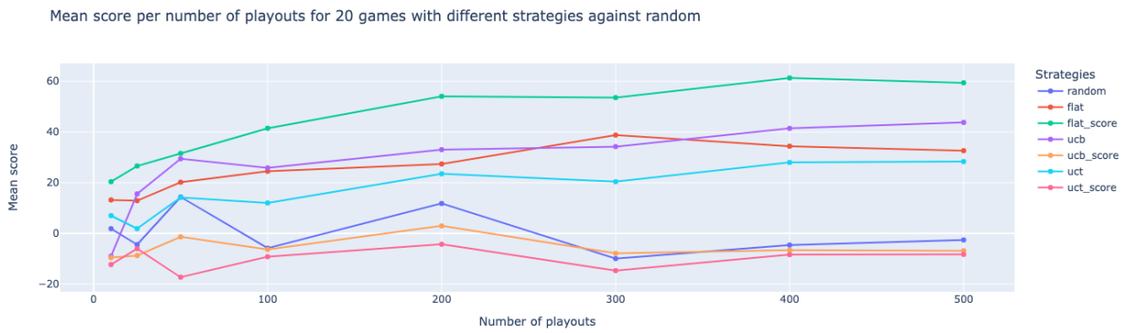
5.2 Résultats pour 50 parties avec 500 playouts



Assez logiquement, les résultats (notamment pour les winrate) sont plus tranchés pour 500 playouts que pour 20. Pour un grand nombre de playouts, la variante score de UCB est très mauvaise, et ne gagne que contre UCT (qui a perdu tous les matchs). Ce résultat est très inattendu puisque précédemment avec un nombre plus petit de playouts, UCB score gagnait des parties. En revanche, contrairement à la version à 20 playouts, la meilleure stratégie est UCB, qui a gagné tous les matchs, ce qui montre l'intérêt d'évaluer le fait qu'un noeud soit prometteur ou non. Flat score et flat classique ont fait presque match nul, ce qui semble indiquer que la variante score est bien moins utile pour un grand nombre de playouts. Cependant, pour des parties avec 100% de winrate (flat et flat score contre UCB score et UCT), la variante score réalise un meilleur score moyen (64 contre UCB score, ce qui est très haut, le meilleur score possible étant 85). Concernant la stratégie UCT, elle performe beaucoup moins bien que précédemment : elle ne gagne presque aucun match contre les autres stratégies pour un grand nombre de playouts, et perd avec des scores moyens

inférieurs à ceux observés pour la version à 20 playouts. Encore une fois, ce sont des résultats qu'on ne pourrait expliquer. Une hypothèse serait une erreur d'implémentation.

5.3 Résultats contre l'aléatoire en faisant varier le nombre de playouts



On observe que UCT score fait des scores négatifs contre l'aléatoire, donc probablement qu'il perd en moyenne, ce qui est assez étrange et indique peut-être une erreur d'implémentation. La variante score de UCB a également de très mauvaises performances, mais cela peut indiquer seulement que la variante n'est pas du tout adaptée pour UCB. En revanche, on remarque que flat score accomplit les meilleurs scores moyens, peu importe le nombre de playouts, ce qui montre bien l'utilité de cette variante. Les scores de UCB semblent s'améliorer avec le nombre de playouts, mais restent, jusqu'à 500 playouts encore loin derrière flat score. Surêment qu'avec plus de playouts, UCB aurait accompli d'encore meilleurs résultats.