Rapport sur le Benchmark des modèles dans le jeu de dames

Introduction

Le développement du jeu de dames et des différents algorithmes a été entrepris avec plusieurs objectifs, dont la correction de bugs existants, l'implémentation d'algorithmes de recherche avancés tels que le minimax et Monte Carlo. De plus, l'un des objectifs est d'évaluer la performance des modèles ainsi que les résultats obtenus. Ce rapport présente les étapes franchies, les difficultés rencontrées et les résultats obtenus.

Benchmark

Suite à l'implémentation de l'algorithme minimax et random, l'un des objectif était de créer un script python pour pouvoir comparer nos premiers modèles cités précédemment.

(Voir le script python nommé benchmark.py)

Random contre Random

Concernant le modèle Random contre lui-même, comme première hypothèse, on s'attend à avoir des résultats assez similaires en termes de victoire / défaite.

```
number of games : 100
number of pieces : 30
player 1  wins 51
player 2  wins 46
draws 3
average game time: 0.02s
game time: 1.75s
```

```
number of games : 100
number of pieces : 30
player 1  wins 34
player 2  wins 60
draws 6
average game time: 0.02s
game time: 1.83s
```

```
number of games : 100
number of pieces : 30
player 1  wins 55
player 2  wins 41
draws 4
average game time: 0.02s
game time: 1.84s
```

On remarque que dans la très grande partie des tests, le modèle random est généralement ex aequo. On a 50% de chance de gagner ou de perdre si les deux modèles s'affrontent. De plus concernant le temps d'exécution il est très rapide dans tous les cas.

Minimax contre Random

Concernant le minimax, l'algorithme ne fonctionnait pas. Mais ayant fait des tests en classe, le minimax était bien plus performant que le random. Or le temps d'exécution était plus long surtout si la profondeur pour prévoir les coups était élevée.

Calcul des perfomances :
Comme on l'a vu sur les screens avec les modèles randoms, j'ai affiché le temps moyen d'une partie ainsi que le temps mis par l'algorithme en fonction du nombre de parties simulées, du nombre de pièces (et la profondeur minimax).

À propos du random / random, les parties sont très rapides. Alors que le minimax (par rapport à mes tests en classe) est plus lent pour pouvoir calculer les meilleures possibilités de coups.

(Voir le script python nommé benchmark.py)

Monte Carlo

L'algorithme de Monte Carlo représente une approche différente dans la prise de décision des coups. En introduisant cet algorithme, on compare ses performances avec celles du minimax et du modèle random.

Après l'implémentation de l'algorithme de Monte Carlo, on procède à une série de tests de performances pour évaluer à la fois la pertinence et la rapidité du modèle par rapport aux autres approches.

Monte Carlo contre Random

le monte carlo était le player 2 :
Pour ce premier test la profondeur est de 5 et on simule 10 parties d'avance.

```
number of games : 10
number of pieces : 30
player 1  wins 0
player 2  wins 10
draws 0
average game time: 3.39s
game time: 33.88s
```

Le Monte Carlo gagne haut la main avec des temps d'exécution bien plus lent que celui du random. De plus, la taille du damier et le nombre de pièces n'a pas été changé.

On réalise un second test et voici les résultats :

```
number of games : 50
number of pieces : 30
player 1  wins 0
player 2  wins 50
draws 0
average game time: 12.89s
game time: 644.54s
```

Cette fois-ci, on simule 50 parties avec une profondeur de 15 et une simulation de 20 potentielles parties.

On en conclut que contre le random le monte Carlo est imbattable mais que sa vitesse d'exécution est très lente en fonction du nombre de parties simulées et de la profondeur de l'algo choisie. Dans mon cas,