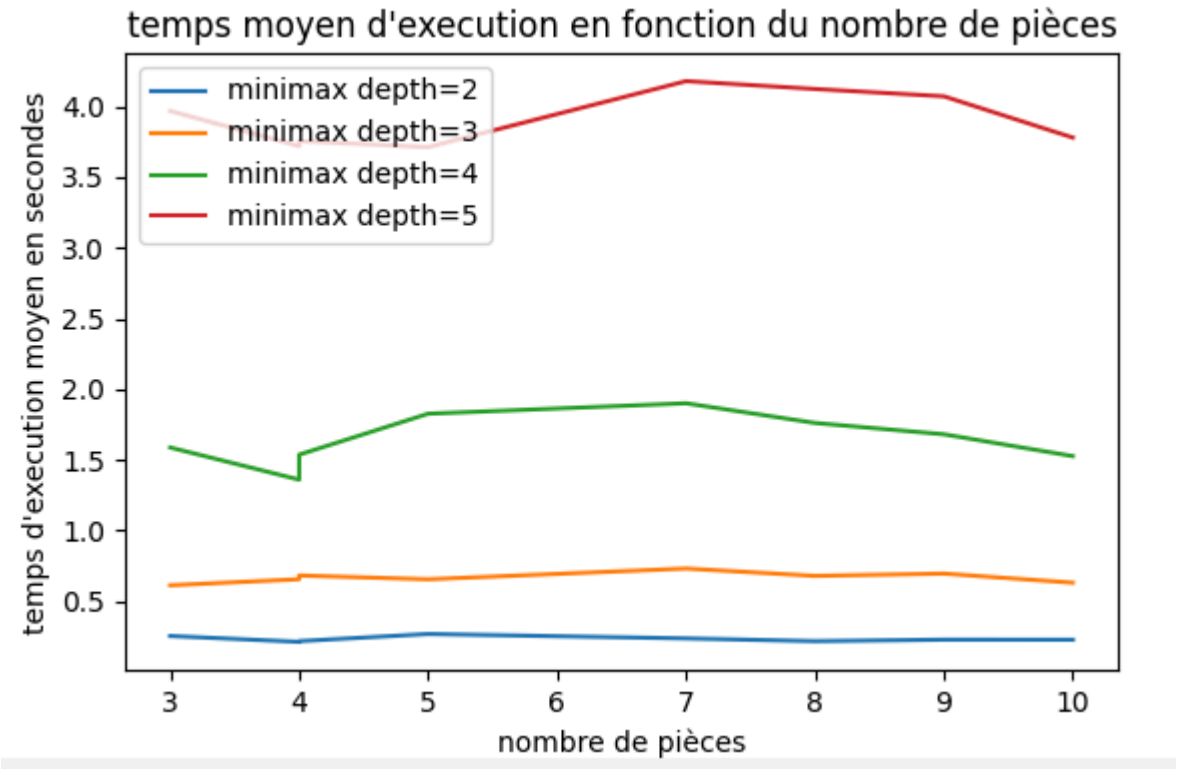
Agent intelligent pour le jeu de dames

Dans ce rapport, nous nous pencherons sur l'implémentation et la performance de deux algorithmes majeurs utilisés pour la prise de décision dans les jeux de dames : Minimax et Monte Carlo Tree Search (MCTS).

temps d'exécution moyen en fonction du nombre de pièces et de la profondeur de l'algo

Dans le fichier “execution_time_benchmark.py” on simule l’exécution de parties pour l’algorithme minimax à différent paramètres (profondeur de 3 à 5, taille de la grille de 7 à 10 , nombre de lignes remplies de pièce 1 à 2) et pour chaque combinaison de paramètres on fait 100 simulations pour déterminer le temps d'exécution moyen.

Le fichier “execution_time_plot.py” nous donne une visualisation du temps d'exécution moyen pour chaque profondeur.



Le temps d'exécution dépend principalement du niveau de profondeur de l’algorithme minimax.

Implémenter l'algorithme de Monte Carlo

```python
def monte_carlo(self,number_of_games,depth,agent_turn):
    sum_score=0
    for number_of_games_index in range(number_of_games):
        winedown=depth
        for depth_index in range(depth):
            game_state = self.game_score()
            if(game_state==0 or game_state==10.5 or game_state==-10.5):
                winedown=depth_index
                break
            selected_piece_position, move_position = self.random_model_predict()
            self.move_piece(selected_piece_position, move_position)

        sum_score+=self.game_score()
        for depth_index in range(winedown):
            self.undo_last_action()

    return sum_score
```

Tests de performances

Minimax vs Random

minimax depth=2 : 96 victoires et 2 égalités pour minimax
62.41 secondes d'exécution
minimax depth=4 : 99 victoires et 0 égalité pour minimax
130.97 secondes d'exécution
minimax depth=5: 100 victoires et 0 égalité pour minimax
294.68 secondes d'exécution

Monte carlo vs Random

monte carlo **depth=1** , number_of_games=30 : 72 victoires et 6 égalités pour monte carlo
15.08 secondes d'exécution
monte carlo **depth=5** , number_of_games=30 : 83 victoires et 3 égalités pour monte carlo
26.38 secondes d'exécution
monte carlo **depth=10** , number_of_games=30 : 89 victoires et 3 égalités pour monte carlo
41.08 secondes d'exécution
monte carlo **depth=15** , number_of_games=30 : 87 victoires et 6 égalités pour monte carlo
84.99 secondes d'exécution

monte carlo depth=10 , **number_of_games=10** : 87 victoires et 5 égalités 8 défaites
 pour monte carlo 41.08 secondes d'exécution
monte carlo depth=10 , **number_of_games=20** : 83 victoires et 3 égalités 13 défaites
 pour monte carlo 88.63 secondes d'exécution
monte carlo depth=10 , **number_of_games=30** : 89 victoires et 5 égalités 6 défaites
 pour monte carlo 133.45 secondes d'exécution

Monte carlo vs Minimax

minimax depth=2 vs monte carlo depth=5 , number_of_games=10 13 victoires et 1 égalités 6 défaites pour minimax
minimax depth=3 vs monte carlo depth=10 , number_of_games=30 1 victoires et 1 égalités 18 défaites pour minimax
minimax depth=4 vs monte carlo depth=10 , number_of_games=25 0 victoires et 0 égalités 20 défaites pour minimax
minimax depth=5 vs monte carlo depth=10 , number_of_games=25 0 victoires et 0 égalités 20 défaites pour minimax

hybride (Minimax / Monte Carlo)

```python
def mini_carlos(self, agent_turn, depth, to_maximise, alpha=-float('inf'), beta=float('inf')):
    game_state = self.check_game_state()
    if game_state == "draw_game":
        return 0

    elif game_state == 1:
        return float("inf")

    elif game_state == -1:
        return -float("inf")

    elif game_state == "game_in_progress":
        if depth == 0:
            return monte_carlo(self,10,5,agent_turn)
```

Même code que le minimax mais on appel monte_carlo au lieux de evaluate_grid.