To Do List(SOBGUI IVAN JOEL):

1. Corriger le bug qui fait planter le minimax::

   Concernant la correction du Bug, il s'agissait de déplacer la méthode **undo_last_action** avant de conditionner **beta < alpha** et la condition **to_maximize** dans la méthode minmax.

```
checker_model.move_piece(*possible_action)
score = self.minmax(checker_model=checker_model, robot_turn=not robot_turn, depth=depth
                    alpha=alpha, beta=beta, to_maximize=to_maximize)
checker_model.undo_last_action()

if to_maximize:
    best_score = max(score, best_score) if robot_turn else min(score, best_score)
else:
    best_score = min(score, best_score) if robot_turn else max(score, best_score)
```

2. Créer un script de test pour comparer le modèle minimax avec le random (différentes profondeurs)

   Pour ce faire, on créera une classe qu'on appellera BenchmarkAI.

```
1 usage
class BenchmarkAI:
    low complexity (6%)
    def __init__(self, players=None, number_of_test=50, checker_grid=None):
        if players is None:
            players = [
                ["random", {}],
                ["minmax", {"depth": 3, "to_maximize": False}]
            ]

        self.players = players
        self.number_of_test = number_of_test
```

   Faisons un test Random(Player 1) VS Minmax(Player 2), sanchant que le minimax va minimiser:

   Pour 100 parties et une profondeur de 3 pour le Minimax:

```
benchmark_ai = BenchmarkAI(
    players: [
        ["random", {}],
        ["minmax", {"depth": 3, "to_maximize": False}],

    ],
    number_of_test=100)
benchmark_ai.play_game()
```

```
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game> python .\benchmark_ai.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
100%|██████████████████████████████████████████████████████████████████████████████| 100/100 [00:48<00:00,  2.08it/s]
Player 1:random  wins 2
Player 2:minmax  wins 49
Draws: 49
```

3. Créer un script de test pour comparer le modèle minimax avec lui même (différentes profondeurs)

Avec notre classe BenchmarkAI:
Nous allons Prendre 2 minmax pour 100 parties:
Player 1 , profondeur de 3 et Minimise
Player 2, profondeur de 1 et Maximise

```
benchmark_ai = BenchmarkAI(
    players: [
        ["minmax", {"depth": 3, "to_maximize": False}],
        ["minmax", {"depth": 1, "to_maximize": True}],

    ],
    number_of_test=50)
benchmark_ai.play_game()
```

```
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game> python .\benchmark_ai.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
100%|| 50/50 [00:20<00:00,  2.48it/s]
Player 1:minmax  wins 50
Player 2:minmax  wins 0
Draws: 0
```

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

Prenons le cas pour le Random VS Minmax,
Nous avons fait un test pour 100 parties

Cas 1:

- On a comme config, 8 lignes et **1 ligne** remplie pour les pieces
- Notre MinMax avait une **profondeur de 3**

On a comme résultat: **48 secondes**

```
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game> python .\benchmark_ai.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
100%|| 100/100 [00:48<00:00,  2.08it/s]
Player 1:random  wins 2
Player 2:minmax  wins 49
Draws: 49
```

Cas 2:

- On a comme config, 8 lignes et **1 ligne** remplies pour les pieces
- Notre MinMax avait une **profondeur de 1**, qui nous a donné

On a comme résultat: **8 secondes**

```
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game> python .\benchmark_ai.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
100%|| 100/100 [00:08<00:00, 11.54it/s]
Player 1:random  wins 1
Player 2:minmax  wins 65
Draws: 34
```

Cas 3:

- On a comme config, 8 lignes et **2 lignes** remplies pour les pieces
- Notre MinMax avait une **profondeur de 3**

On a comme résultat : **3 minutes 55 secondes**

```
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game> python .\benchmark_ai.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
100%|| 100/100 [03:55<00:00,  2.36s/it]
Player 1:random  wins 0
Player 2:minmax  wins 85
Draws: 15
```

5. Implémenter l'algorithme de Monte Carlo
   L'algorithme du Monte Carlo:

1. **Initialisation** : Commencez à partir de l'état actuel du jeu.
2. **Simulation** : Répétez un grand nombre de fois :
   - Choisissez une action aléatoire parmi les actions possibles.
   - Simulez le jeu en effectuant cette action.
   - Évaluez le résultat de la simulation.
3. **Sélection de la meilleure action** : Choisissez l'action qui a produit les meilleurs résultats lors des simulations.

Pour on implémentation, nous allons ajuster notre algorithme du MinMax et ajouter en paramètre, le nombre d'itérations et nous utiliserons random.choice pour prendre un pièce aléatoire dans la liste des mouvements possibles:

```python
def get_best_move(self, checker_model, depth, to_maximize=False, nb_of_iterations=5):
    best_score = -float("inf") if to_maximize else float("inf")
    best_piece_position, best_move_position = None, None
    dict_of_moves = checker_model.dict_of_possible_moves

    # possible_actions = [( (piece) : (move) ),]
    possible_actions = []
    for selected_piece_position, moves in dict_of_moves.items():
        for move in moves:
            possible_actions.append((selected_piece_position, move.get_final_position()))

    # for possible_action in possible_actions:
    for _ in range(nb_of_iterations):
        possible_action = self.get_random_move(possible_actions)
```

6. Faire différents tests de performances (pertinence du modèle et rapidité du modèle):
   Random VS Monte Carlo pour 100 tests:

```python
benchmark_ai = BenchmarkAI(
    players: [
        ["random", {}],
        ["mtc", {"depth": 3, "to_maximize": False, "nb_of_iterations": 10}]

    ],
    number_of_test=100)
benchmark_ai.play_game()
```

```
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game> python .\benchmark_ai.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
100%|| 100/100 [02:07<00:00,  1.28s/it]
Player 1:random  wins 0
Player 2:mtc  wins 50
Draws: 50
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game>
```

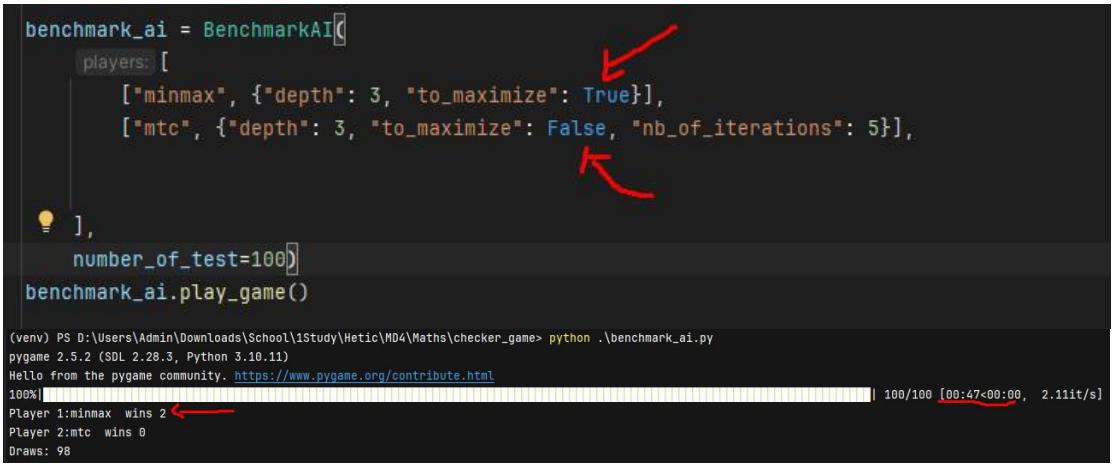
Monte Carlo VS MinMax pour 100 tests:

```
benchmark_ai = BenchmarkAI(
    players: [
        ["mtc", {"depth": 3, "to_maximize": True, "nb_of_iterations": 5}],
        ["minmax", {"depth": 3, "to_maximize": False}],

    ],
    number_of_test=100)
benchmark_ai.play_game()
```

```
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game> python .\benchmark_ai.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
100%|| 100/100 [01:39<00:00,  1.00it/s]
Player 1:mtc  wins 6
Player 2:minmax  wins 23
Draws: 71
```

MinMax VS Monte Carlo:



```
benchmark_ai = BenchmarkAI(
    players: [
        ["minmax", {"depth": 3, "to_maximize": True}],
        ["mtc", {"depth": 3, "to_maximize": False, "nb_of_iterations": 5}],


    ],
    number_of_test=100)
benchmark_ai.play_game()
```

```
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game> python .\benchmark_ai.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
100%|| 100/100 [00:47<00:00,  2.11it/s]
Player 1:minmax  wins 2
Player 2:mtc  wins 0
Draws: 98
```

Monte Carlo (Max) VS Monte Carlo(Min):

```
benchmark_ai = BenchmarkAI(
    players: [
        ["mtc", {"depth": 3, "to_maximize": True, "nb_of_iterations": 5}],
        ["mtc", {"depth": 3, "to_maximize": False, "nb_of_iterations": 5}],


    ],
    number_of_test=100)
benchmark_ai.play_game()
```

```
(venv) PS D:\Users\Admin\Downloads\School\1Study\Hetic\MD4\Maths\checker_game> python .\benchmark_ai.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
100%|| 100/100 [00:38<00:00,  2.60it/s]
Player 1:mtc  wins 3
Player 2:mtc  wins 9
Draws: 88
```

Remarques:

Dans certains test, nous avons trouvé des résultats incohérents. Ce qui signifie qu'il faudrait encore une amélioration de notre code pour certains cas(Comme le cas où si on met 2 players MinMax, et que le 1er Maximise et le 2e Minimise, le 2e Player gagnera toujours).

Sans compter qu'à un **nombre important de pièce**s, de **profondeurs** ou **nombres de lignes**, on **augmente la complexité temporelle**.