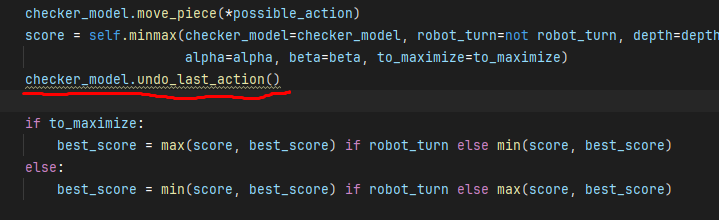
**To Do List(SOBGUI IVAN JOEL):**

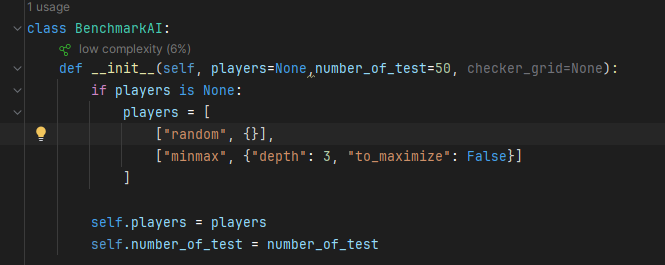
1. Corriger le bug qui fait planter le minimax::

Concernant la correction du Bug, il s’agissait de déplacer la méthode **undo\_last\_action** avant de conditioner **beta < alpha** et la condition **to\_maximize** dans la méthode minmax.



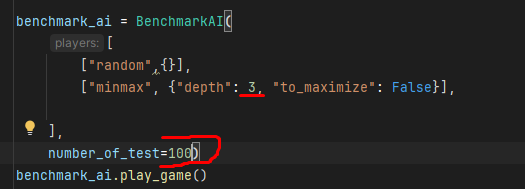
1. Créer un script de test pour comparer le modèle minimax avec le random (différentes profondeurs)

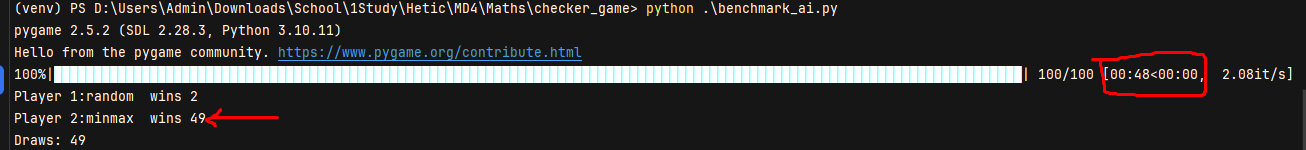
Pour ce faire, on créera une classe qu’on appellera BenchmarkAI.



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:





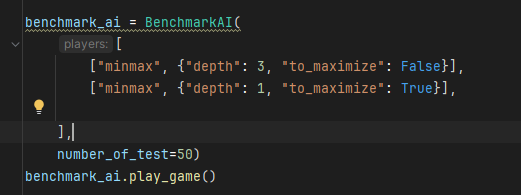
1. Créer un script de test pour comparer le modèle minimax avec lui même (différentes profondeurs)

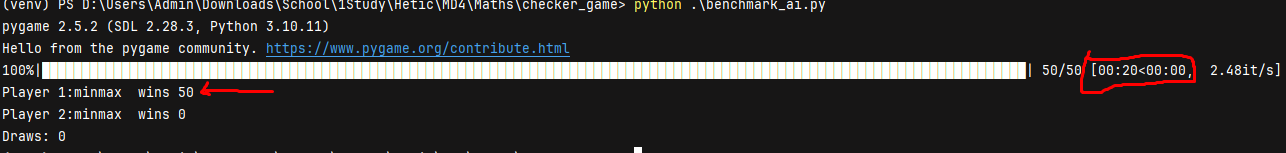
Avec notre classe BenchmarkAI:

Nous allons Prendre 2 minmax pour 100 parties:

Player 1 , profondeur de 3 et Minimise

Player 2, profondeur de 1 et Maximise





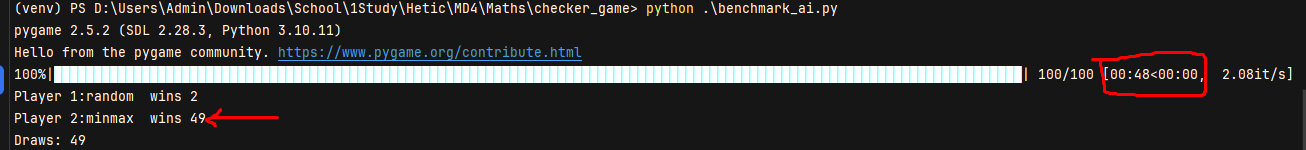
1. Calculer le temps d'exécution moyen en fonction du nombre de pièces et de la profondeur de l'algo

Prenons le cas pour le Random VS Minmax,

Nous avons fait un test pour 100 parties

Cas 1:

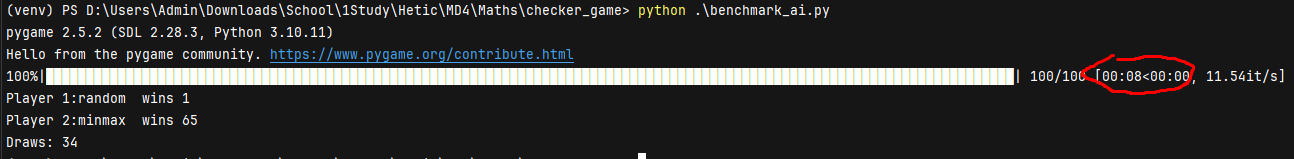
* On a comme config, 8 lignes et **1 ligne** remplie pour les pieces
* Notre MinMax avait une **profondeur de 3**

On a comme résultat: **48 secondes** 

Cas 2:

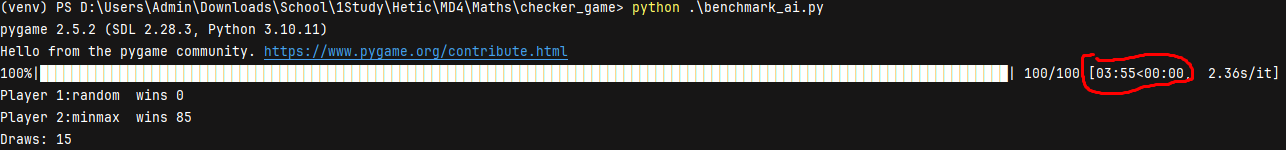
* On a comme config, 8 lignes et **1 ligne** remplies pour les pieces
* Notre MinMax avait une **profondeur de 1**, qui nous a donné

On a comme résultat: **8 secondes**



Cas 3:

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

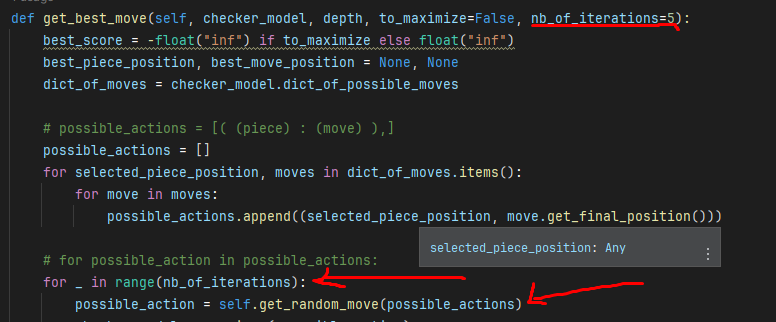
On a comme résultat : **3 minutes 55 secondes**

1. Implémenter l'algorithme de Monte Carlo

L’algorithme du Monte Carlo:

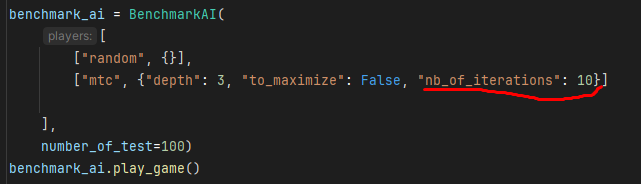
1. ****Initialisation**** : Commencez à partir de l'état actuel du jeu.
2. ****Simulation**** : Répétez un grand nombre de fois :
   * Choisissez une action aléatoire parmi les actions possibles.
   * Simulez le jeu en effectuant cette action.
   * Évaluez le résultat de la simulation.
3. ****Sélection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

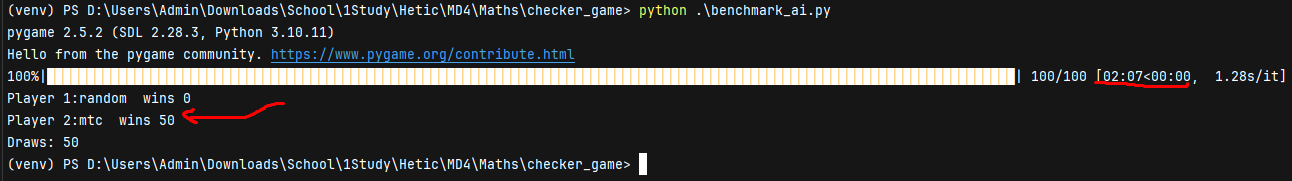
Pour on implémentation, nous allons ajuster notre algorithme du MinMax et ajouter en paramètre, le nombre d’itérations et nous utiliserons random.choice pour prendre un pièce aléatoire dans la liste des mouvements possibles:



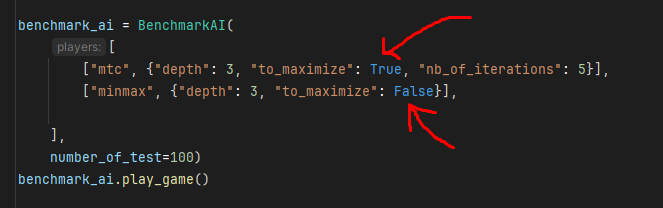
1. Faire différents tests de performances (pertinence du modèle et rapidité du modèle):

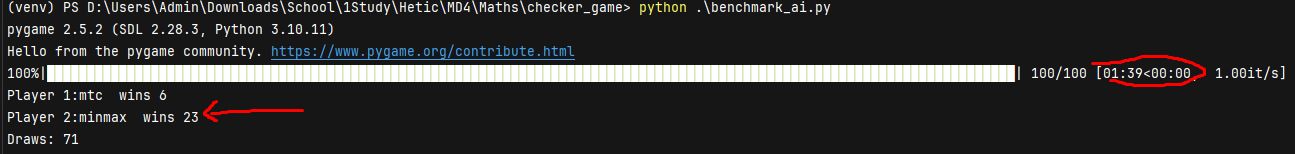
Random VS Monte Carlo pour 100 tests:



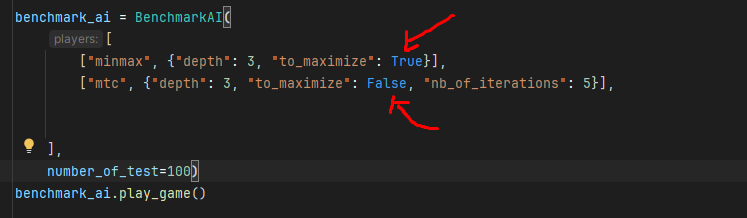


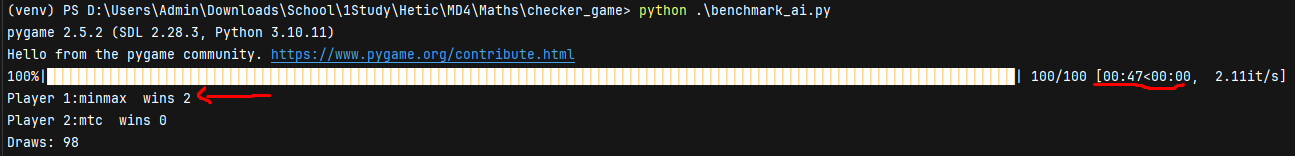
Monte Carlo VS MinMax pour 100 tests:



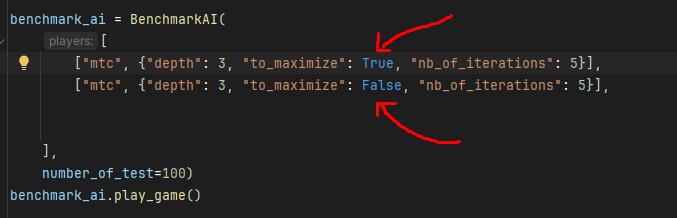


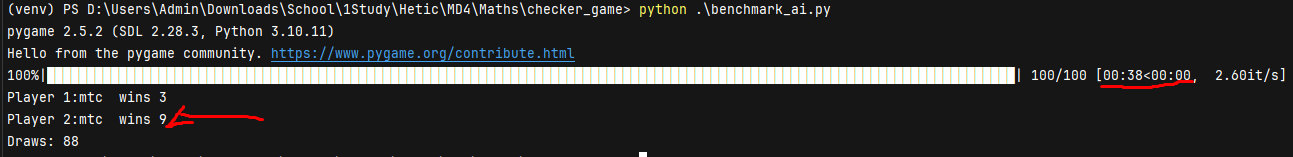
MinMax VS Monte Carlo:





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





**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**.