3-1.

1. **La différence entre MiniMax et Alpha-Beta**

MiniMax :

- Explore toutes les branches possibles de l'arbre des jeux jusqu'à la profondeur maximale.
- Identifie la meilleure action en considérant que l'adversaire joue de manière optimale.

Alpha-Beta :

- Donne la même action optimale que MiniMax, mais en explorant moins de nœuds grâce aux coupures alpha-beta.

En somme on peut dire que les deux algorithmes ne sont pas différents car Alpha-Beta est une optimisation de Minimax et ne modifie pas le résultat final.

2. Oui, cela correspond aux attentes théoriques. Alpha-Beta explore moins de nœuds, surtout si l'ordre des coups est favorable.
3. **Observation**

   À mesure que la profondeur augmente, Alpha-Beta devient de plus en plus efficace par rapport à MiniMax.Et aussi, la distribution des nœuds visités dépend de l'ordre des coups explorés.

3-2.

4. Définition de la fonction(voir code)
5. **Expliquez comment elle peut améliorer l'élagage d'Alpha-Beta. Quels sont ses inconvénients ?**

   Amélioration de l'élagage :

   - Une fonction d'évaluation précise permet à Alpha-Beta d'évaluer les nœuds plus efficacement.
   - Les valeurs alpha et beta seront mises à jour plus rapidement, augmentant ainsi les chances de coupures (élagage) dans les sous-arbres inutiles.
   - Par exemple, si un état semble nettement favorable/défavorable, les branches suivantes peuvent être élaguées.

   Inconvénients :

   - Si la fonction d'évaluation est complexe, le temps de calcul par nœud augmente, ralentissant ainsi la recherche.
   - Une fonction mal conçue (biaisée ou inadéquate) peut mal évaluer les positions intermédiaires, entraînant de mauvais choix stratégiques.

- Elle ne garantit pas une victoire si les prédictions intermédiaires sont inexactes.

6. **Résultats attendus**

   AlphaBetaPlayer devrait surpasser l'agent de base en jouant des coups optimisés basés sur la recherche dans l'arbre.

   À mesure que la profondeur de recherche augmente, les performances de l'agent Alpha-Beta s'améliorent, mais le temps de calcul augmente également.

3-3.

7. **Limite Supérieure au Nombre de Successeurs pour un État**

Pour le jeu d'Avalam, le nombre de successeurs pour un état donné dépend de la configuration actuelle du plateau. Chaque tour peut être déplacé vers une case adjacente (haut, bas, gauche, droite, et les diagonales), à condition que le mouvement soit valide.

Pour le premier état du jeu standard, chaque tour peut potentiellement être déplacée vers 8 positions adjacentes (si aucune restriction n'est appliquée). Cependant, toutes les tours ne peuvent pas être déplacées vers toutes les directions en raison des contraintes du plateau et des règles du jeu.

8. **les successeurs ne sont pas nécessaires** :

Non, tous ces successeurs ne sont pas nécessaires pour être exhaustifs en raison des symétries du plateau. Par exemple, les mouvements symétriques autour du centre du plateau peuvent être considérés comme équivalents. En tenant compte des symétries, nous pouvons réduire le nombre de successeurs à considérer.

9. **États pouvant être ignorés** :

   Actions non critiques :

   - Déplacements sur des piles sans impact immédiat sur le contrôle ou la stratégie globale.

   Redondances :

   - États résultant de mouvements symétriques ou équivalents.

   Mouvements défensifs faibles :

- Actions qui n’apportent pas de valeur stratégique significative (par exemple, empiler une tour isolée sans potentiel futur).

Perte d’exhaustivité :

- Ignorer certains états réduit la qualité de la recherche (elle peut manquer des coups critiques).
- Cependant, cette simplification améliore considérablement les performances en limitant le nombre de successeurs à explorer.

10. Description de la Fonction de Successeurs

La fonction de successeurs doit générer tous les états possibles à partir d'un état donné en appliquant tous les mouvements valides. Nous devons également tenir compte des symétries pour réduire le nombre de successeurs à considérer.

11. Voir Code

3-4.

12. **Notion de profondeur dans alphabeta_search d’AIMA**

La **profondeur** correspond au nombre de niveaux d’exploration de l’arbre à partir de l’état initial.

13.Couper l’arbre pour d'autres raisons peut être utile pour :

- Réduire les calculs inutiles
- Limiter le temps de calcul
- Concentrer les ressources sur les branches prometteuses
- Éviter l’exploration de nœuds redondants