Laboratoire 2 - Recherche adversarielle

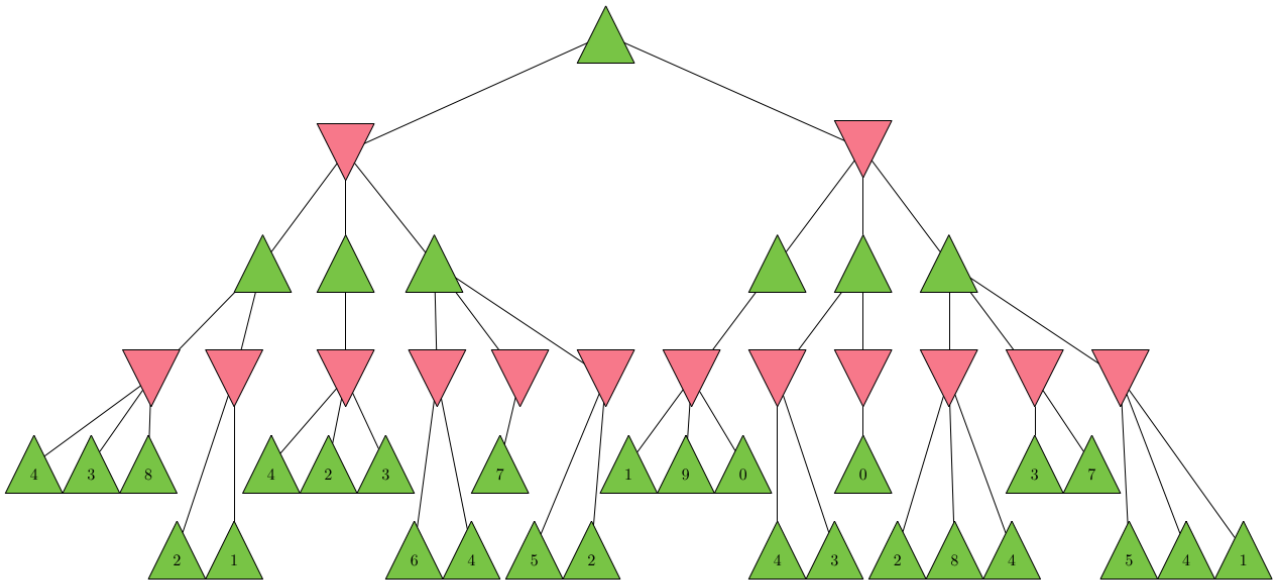
Exercice 1 - Quizz

Lesquels des énoncés suivants sont vrais et lesquels sont faux ? Expliquez brièvement vos réponses.

1. Dans un jeu totalement observable, tour par tour, à somme nulle, et entre deux joueurs parfaitement rationnels, cela n'aide pas le premier joueur à savoir quelle stratégie le deuxième joueur utilise, c'est-à-dire, quel mouvement le deuxième joueur effectuera, étant donné le mouvement du premier joueur.
2. Dans un jeu partiellement observable, à tour de rôle, à somme nulle, et entre deux joueurs parfaitement rationnels, cela n'aide pas le premier joueur à savoir quel mouvement le deuxième joueur effectuera, étant donné le mouvement du premier joueur.
3. Dans le cas d'une recherche alpha-beta, l'heuristique de sélection des actions à étendre n'a pas d'influence sur la décision finale du joueur mais sur le temps de calcul nécéssaire à cette décision.
4. Un agent pour le jeu Backgammon (présence d'aléatoire via un lancement de deux dés) parfaitement rationnel ne perd jamais.
5. Dans le cas d'une recherche de type MCTS avec selection UCT, on va toujours choisir l'action ayant le meilleur score moyen.

Exercice 2 - Algorithme Minimax et alpha-beta pruning

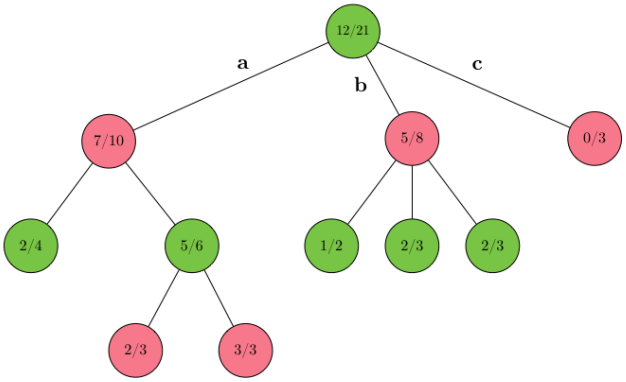
Soit l'arbre de recherche suivant :



1. Complétez les valeurs minimax pour chacun des noeuds de l'arbre.
2. Supposons deux joueurs rationnels, quel sera le score obtenu par le joueur `MAX` ?
3. En appliquant l'alpha-beta pruning sur cet arbre (avant de calculer les valeurs minimax), quelles seront les branches que vous pouvez éliminer ? Au total, combien de noeuds de recherche avez-vous évité d'explorer ?

Exercice 3 - Monte-Carlo tree search

Soit l'arbre de recherche MCTS partiellement construit suivant :



Les valeurs (x/y) de chaque noeud n indiquent respectivement le score total cumulé du noeud via les simulations $\big(U(n)\big)$ et le nombre total de simulation ayant impliqué le noeud $\big(N(n)\big)$. La règle de sélection utilisée pour la phase de sélection est la suivante : $\mathsf{UCB1}(n) = \frac{U(n)}{N(n)} + 2\sqrt{\frac{\ln N\big(\mathsf{Parent(n)}\big)}{N(n)}}$.

1. Réalisez 3 itérations complètes de l'algorithme MCTS sur cet arbre. C'est à dire que vous devez réalises les étapes de sélection, d'expansion, de simulation, et de backpropagation 3 fois. Indiquez après chaque itération l'arbre mis-à-jour. Pour la phase d'expansion, supposez que deux nouveaux noeuds sont générés, et que celui utilisé pour la simulation est celui de gauche. Pour les simulations, considérez que les scores suivants sont obtenus : 5,2, et puis 10.
2. A l'issue de ces simulations, quelle action le joueur actuel devrait jouer (a, b, ou c) ?

Exercice 4 - Morpion et Puissance 4

⚠Cet exercice a pour but de vous aider à démarrer le projet du cours. Il est fortement recommandé de le comprendre avant de commencer votre implémentation. De plus, notez que l'examen ne demandera pas de réaliser du code Python.

Dans cet exercice, il vous est demandé d'implémenter les stratégies minimax et alpha-beta pour le jeu du Morpion et le Puissance 4. Vous avez à votre disposition une implémentation python dans le sous-dossier `code-lab2`. Le dossier contient plusieurs fichiers :

— `games.py, utils4e.py` : code source des jeux, vous n'avez pas à modifier cette partie.

— `main.py` : code à exécuter pour lancer les différents scénarios développés ci-dessous. Vous n'avez pas à modifier cette partie.

— `player.py` : code source des différentes stratégies à compléter.

Il est important de prendre le temps de comprendre la structure des objets utilisés, Le caractère récursif des concepts utilisés peut paraître déroutant, n'hésitez pas à utiliser des dessins pour vous aider, ainsi qu'à vous appuyer sur les slides du cours.

Pour lancer un scénario, il vous suffit de lancer la commande suivante : *python3 main.py –scenario Si*, en remplaçant Si par l'un des scénarios (**S1, S2, S3, ...**) explicités ci-dessous.

1. Observer la fonction de score *utility(self, board, player)* sur les états terminaux. Sans l'implémenter, définissez une autre fonction de score privilégiant les victoires en un minimum d'actions.
2. Donner une borne supérieure sur le nombre d'états possibles pour le jeu du morpion. A partir de cette observation, est-il raisonnable de calculer le score de l'ensemble des états de l'arbre de recherche ?
3. On s'intéresse à la stratégie minimax :
   — Implémenter les fonctions récursives `max_value(state)` et `min_value(state)`
   — Lancer les scénarios **S1** :*Random (X) vs Minimax (O) on tic-tac-toe* et **S2** : *Random (X) vs Minimax (O) on Connect Four*. Que pouvez-vous en conclure ?
4. On s'intéresse à la stratégie alpha-beta :
   — Quel est l'intérêt de cette méthode par rapport à la version basique du minimax ?
   — Implémenter les fonctions récursives *max_value(state, alpha, beta)* et *min_value(state, alpha, beta)*
   — Lancer les scénarios **S3** :*Random (X) vs AlphaBeta (O) on tic-tac-toe* et **S4** : *Random (X) vs Alpha-Beta (O) on Connect Four*. Que pouvez-vous en conclure ?
5. Lancer le scénario **S5** :*minimax (X) vs AlphaBeta (O) on tic-tac-toe*. Que pouvez-vous prévoir avec certitude ?
6. On souhaite mettre en place une stratégie de type alpha-beta avec cutoff (stratégie de type A). h est la fonction heuristique naive sur les états intermédiaires. Cette fonction renvoi toujours 0.
   — Quel est le raisonnement logique associé à l'utilisation de la fonction heuristique h? Pourquoi avoir choisi une profondeur de cutoff de 6 ?
   — Implémenter les fonctions récursives *max_value(state, alpha, beta, depth)* et *min_value(state, alpha, beta, depth)*
   — Lancer le scénario **S6** :*Random (X) vs AlphaBeta (O) on tic-tac-toe*. Qu'observez-vous ?
   — Proposer votre propre heuristique d'évaluation sur les états intermédiaires en complétant *your_nice_heuristic(state, player)*. Confronter votre heuristique et l'heuristique naive h en lançant le scénario **S7**.

Exercice 5 - Preuve de performance (difficile)

⚠**Cette question est hors-matière pour l'examen.** Prouvez la complexité temporelle $O(b^{m/2})$ de l'algorithme Alpha-Beta dans le cas d'un ordering parfait (ie. les meilleurs actions sont toujours cherchées en premier dans l'arbre). Pour rappel, b est le facteur de branchement moyen et m la profondeur de l'arbre, et on part d'une recherche minimax avec une complexité temporelle de $O(b^m)$.