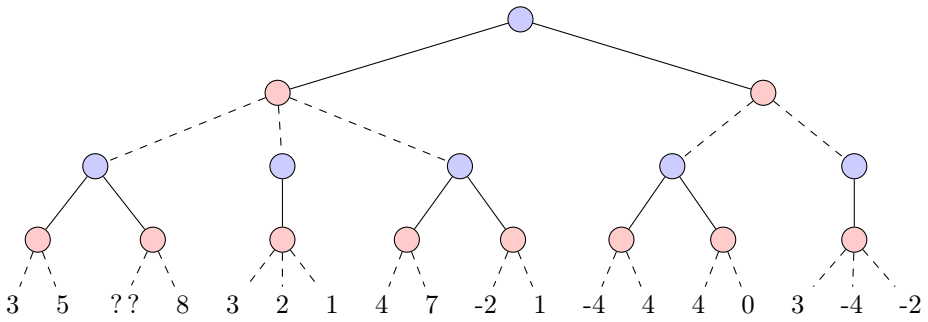
Intelligence Artificielle : feuille 2

Les jeux de plateaux à deux joueurs
Recherche avec horizon, MiniMax, Alpha-Beta

1 Exploration d'arbres... sur feuilles

Soit l'arbre de jeu représenté ci-dessous (les arcs venant de noeuds ennemis sont en pointillés, les valeurs aux feuilles sont les estimations heuristiques du point de vue ami) :



1. Quel est le facteur de branchement du jeu représenté sur l'arbre ci-dessus ? Toutes les branches d'un arbre de jeu doivent elles être de la même hauteur ?
2. Sans tenir compte de la feuille ayant une valeur de " ? ?", quelle est le meilleur plateau pour Ami ? Quel est le meilleur plateau pour Ennemi ? Expliquez le fait que deux noeuds de l'arbre aient un seul fils. Est-ce plutôt une bonne chose ou une mauvaise chose pour *ami* ?
3. Donnez la plus grande valeur possible pour la feuille ayant la valeur heuristique notée " ?" et permettant à α-β d'élaguer la feuille 8. Vous utiliserez cette valeur pour la suite du sujet ;
4. Déroulez *alpha-béta* classique sur ce même arbre de jeu : Attention, vous ferez bien figurer sur votre solution les coupes effectuées et l'évolution éventuelle des valeurs α et β au cours de la recherche ;

2 Du Morpion aux échecs

Pour jouer aux échecs, nous utiliserons une bibliothèque Python : `python-chess`, disponible à l'adresse

`https://pypi.org/project/python-chess/`

Au CREMI, la bibliothèque est installée dans un environnement virtuel. Pour pouvoir l'utiliser, vous devez exécuter la commande suivante :

```
source /net/ens/python/IA/activate
```

Si tout se passe bien, le prompt sur votre terminal doit commencer par

```
(IA)
```

Copiez également le fichier `starter-chess.py` disponible depuis le site du cours. Il vous montre comment dérouler une partie aléatoire sur le jeu d'échec.

1. Faites une recherche exhaustive de toutes les parties d'échec, mais en limitant la profondeur de la recherche par un paramètre de la recherche. Jusqu'à quelle profondeur pouvez vous aller en moins de 30s ? Combien de noeuds votre recherche explore-t-elle à profondeur 1, 2, 3, ... ?

2. Nous allons devoir fixer un horizon à nos recherches. Pour cela, il nous faut définir une heuristique pour le plateau d'échec. Codez l'heuristique proposée par Claude Shannon en 1950 et vue en cours. Vous n'utiliserez que la partie de l'heuristique donnant un poids aux pièces de jeu. Ajoutez un moyen d'exprimer qu'il est préférable d'avancer ses pions pour les mener éventuellement à la Reine (**aide** : pour parcourir le plateau de jeu, vous pourrez utiliser la méthode `board.piece_map()` tout en récupérant le caractère symbolisant la pièce grâce à la méthode `symbol()` offerte par la bibliothèque).
3. Codez un Minimax sur les échecs. Attention, les algorithmes vus en cours ne se préoccupent pas de renvoyer le coup associé au meilleur choix : ils ne renvoient que la valeur minimax de l'arbre. Il faudra donc coder une version spéciale de `MaxMin(...)` pour le niveau 1.
4. Codez un match Joueur Aléatoire contre Minimax niveau 3. Puis Minimax niveau 1 contre Minimax niveau 3. Si vous remarquez que vos joueurs joeunt en boucle, ajoutez un mécanisme qui permet de choisir le coup au hasard parmis les ex-aequao, si besoin.

3 L'alpha et l'oméga de $\alpha - \beta$

1. Faites évoluer votre code Minimax en $\alpha - \beta$. Comparez les temps de recherche et le nombre de noeuds explorés sur plusieurs plateaux de jeu entre les deux méthodes, à profondeur égale. Faites un match Minimax contre $\alpha - \beta$ à profondeur égales.
2. Encapsulez votre $\alpha - \beta$ dans un Iterative Deepening pour garantir que votre recherche ne dépassera jamais 10s de calcul.
3. Programmez un joueur humain (qui demande au clavier le coup à jouer). Battez votre IA (ou pas).