Examen IA et Jeux

ECM 2A

25 Janvier 2019

2h - Sans documents

Question 1. Heuristique admissible

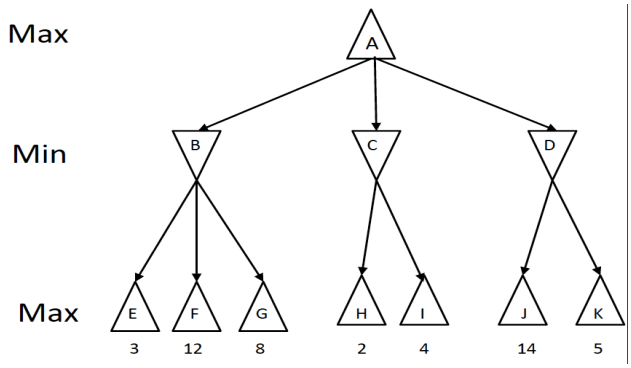
1. Rappelez ce qu'est une heuristique admissible et sa propriété essentielle.
2. Proposez, en argumentant sur leur caractère admissibile, deux heuristiques admissibles pour le problème suivant :

Sur un échiquier vide, vous souhaitez trouver comment déplacer une reine, en un nombre de coups minimal, de sa case courante, repérée par sa ligne *l* (un chiffre entre 1 et 8) et sa colonne *c* (idem), vers une case de coordonnées *(l', c')*. La reine se déplace comme une reine aux échecs (n'importe quelle distance en diagonale, horizontale ou verticale).

3. On dit qu'une heuristique domine une autre heuristique si elle est toujours meilleure. Dans vos réponses à la question précédente, déterminez s'il une heuristique en domine une autre.
4. On considère maintenant que des pièces sont présentes sur l'échiquier (mais il n'y a pas d'adversaire qui joue entre les coups de la Reine, seule la Reine se déplace) et on veut résoudre le même problème de déplacement de la reine. On considère que la Reine doit contourner les pièces présentes et ne les mange pas. Les heuristiques que vous avez trouvées en 3) sont-elles toujours valables ?

Question 2. Jeu à deux joueurs

On vous donne l'arbre des parties possibles d'un jeu à partir de la position courante (la racine de l'arbre), ainsi que les valeurs *d'utilité* ou valeurs *minmax* des feuilles.



On vous redonne le pseudo code de l'algorithme Alpha Beta vu en cours ci-dessous (on ne vous donne que le Max-Value, le Min-Value étant parfaitement symétrique).

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

En supposant que lorsque l'on itère sur les successeurs d'un état, les nœuds les plus à gauche sont en premier et ceux les plus à droite en dernier (par exemple les fils de A sont dans l'ordre B, C et D), donnez l'ordre dans lequel sont exécutés les appels aux fonctions Max-Value et Mini-Value avec leurs paramètres. Complétez la liste suivante dans laquelle vous sont donnés les deux premiers appels :

Fontion appelée	Nœud en paramètre	Alpha et Beta
Max-Value	A	infini, + infini
Min-Value	B	-infini, +infini
…		

Question 3. Minimax multi- adversaires

On vous redonne ci dessous le code d'un joueur MiniMax standard classique tel que celui que vous avez programmé pour l'Awalé et les Echecs :

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

On considère que vous vous intéressez au joueur Max. Vous devez modifier ce code pour concevoir un joueur artificiel pour le jeu Pacman dans lequel il y a plusieurs joueurs, le vôtre (le joueur Max), et plusieurs joueurs adverses (des fantômes qui poursuivent un but commun, vous manger). Vous avez donc plusieurs adversaires.

1. Donnez une version pseudo code de votre joueur Minimax en commentant l'idée principale de vos modifications.
   On considérera qu'il y a *L* joueurs, le joueur *0* est le votre, les autres joueurs, numérotés *1* à *L-1*, sont vos adversaires et que vous disposez d'une fonction *Successors(State, j)* qui prend en paramètre l'état de jeu *State* et le numéro du joueur *j*, et renvoi les couples (action ; état) correspondant à l'état courant et au joueur courant.

2. Serait-il possible d'écrire une version Alpha-Beta de ce joueur ? Si oui donnez le pseudo code correspondant.

Question 4. Jeu à deux joueurs avec hasard

Certains jeux font appel au hasard. Le tour d'un joueur se décompose en un tirage au hasard puis en une action parmi un ensemble d'actions qui dépendent de ce tirage. Par exemple au BackGammon le tour d'un joueur se décompose en deux étapes, d'abord un lancé de deux dés, puis le coup proprement dit qui dépend du lancé de dé, au Backgammon un coup est constitué du déplacement de deux pions, ou de deux déplacements successifs du même pion, d'un nombre de cases égal à chacun des résultats des dés.

1. Décrire selon vous les changements requis par ce type de jeu dans l'algorithme minimax.
2. Ecrivez le pseudo code correspondant.