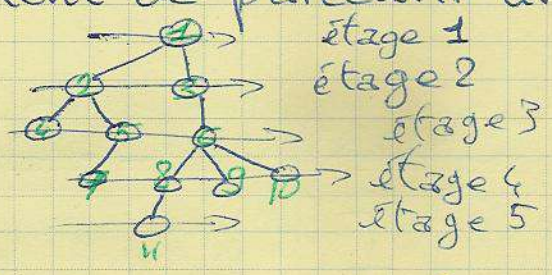
BLUTEAU
Pierre
L3 IC
29/6/17

15,5

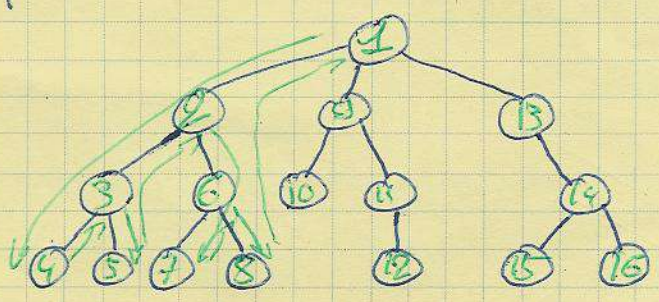
CE - Aide à la Décision

Question 1.1 1,5/2

a) Les algorithmes de type breadth-first (parcours en largeur) permettent de parcourir un arbre étage par étage :



b) Les algorithmes de type depth-first (parcours en profondeur) permettent de parcourir un arbre en descendant au maximum, et en ne remontant que lorsqu'une feuille est atteinte ou que tous les fils du noeud en cours sont traités.



(récursivité gauche, on aurait pu droite)

c) Les algorithmes "best-first search" ~~pas~~ sont plus optimisés *ils effectuent la recherche en un temps bien plus court car ils parcourent, dans l'ordre, les noeuds ~~des~~ qui sont les plus proches de ce qu'on recherche (best first). Exemple : A* prend dans OPEN l'heuristique la plus faible pour choisir le noeud à traiter).

*ils optiment breadth-first

Question 1.2 ○

Un parcours en largeur ne mémorisant pas les étapes traversées pour obtenir le résultat d'une recherche, il ne peut retourner le chemin amenant à la solution. Un parcours en profondeur est alors plus adapté.

Question 1.3 ○

Il s'agit d'un système permettant de déterminer quel choix est le plus judicieux pour obtenir, après un grand nombre de choix que nous ne maîtrisons pas tous, un résultat donné.

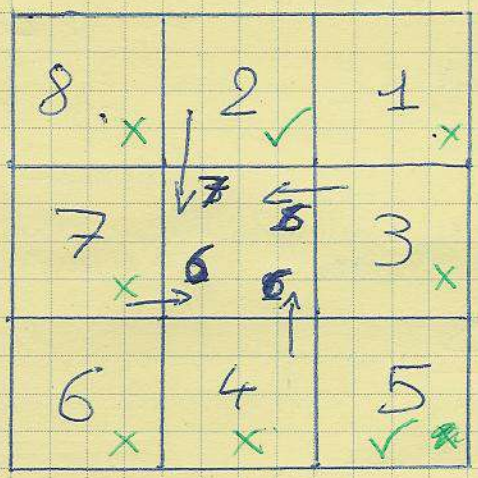
Question 2.2

2/2

* Fonction heuristique

$f(n)$ = nombre de pièces mal placées

* Étape 1 $f(n) = 6$



coût : 1

OPEN ← { 2 (7) ; 3 (6) ; 4 (8) ; 7→ (6) }
coût+1 ; coût+1 ; coût+1

CLOSED ← {  }

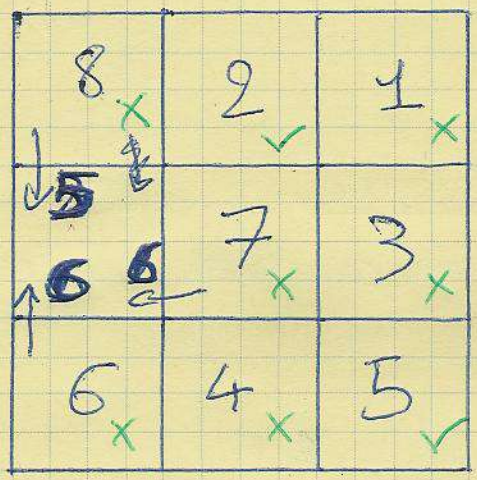
Dans OPEN et CLOSED sont stockées des configurations du damier (exemple ? est  avec heuristique = 7)

* Étape 2

Égalité : on retire de CLOSED / OPEN ∈3 (6) ou 4 (6) ou 7→ (6)

↳ Si on fait 7→ alors la case vide passe à l'indice 4, ce qui est le plus faible que pour 4 (donne 8) ou ∈3 (donne 6).

↳ On retire 7→ de OPEN pour le traiter



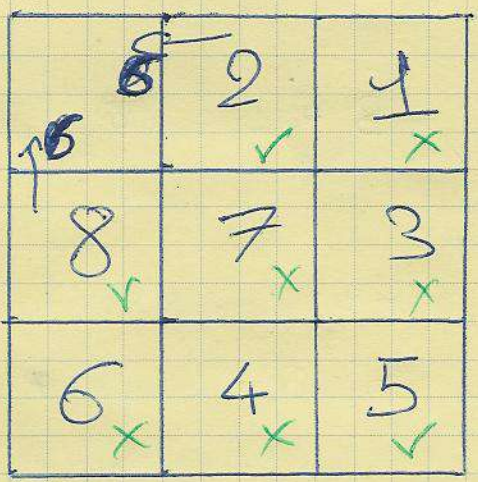
coût : 2
f(n) = 6

OPEN = { OPEN précédent ; 2 1 (5) / 8 7 3 / 6 4 5 ; 8 2 1 (6) / 6 7 3 / 4 5 }
coût 2 ; coût 2

CLOSED = { 8 2 1 / 7 3 / 6 4 5 ; 8 2 1 / 7 3 / 6 4 5 }
parent : étape 1

e 7 est inutile (configuration déjà dans OPEN avec un coût inférieur (0))

* Etape 3



coût : 3
f(n) = 5

OPEN ← { OPEN ; 2 1 / 8 7 3 / 6 4 5 } coût 3

CLOSED ← { CLOSED ; 2 1 / 8 7 3 / 6 4 5 / coût 3 / parent : étape 2 }

BLUTEAU
Pierre
L3/C
29/6/7

CE - AD (suite)

~~*~~ Étape 4

~~L'heuristique la plus forte dans OPEN est~~

Égalité entre les heuristiques 6 :

8 2 1 7 _ 3 6 4 5	8 2 1 7 4 3 6 _ 5	8 2 1 6 7 3 _ 4 5	2 _ 1 8 7 3 6 4 5
coûts : ①	①	②	③

(un état barré entre le premier et le deuxième)

Le plus faible indice du vide est ici → ③

2 ✗	→5 ←6 ↑6	1 ✗
8 ✓	7 ✗	3 ✗
6 ✗	4 ✗	5 ✓

coût : 4
$f(n) = 6$

_ 2 1
8 7 3
6 4 5

est déjà dans CLOSED avec un coût inférieur

2 1
8 7 3
6 4 5

est ajouté dans OPEN (coût 4, h = 6, parent : 2 _ 1 / 8 7 3 / 6 4 5)

2 7 1
8 _ 3
6 4 5

est ajouté dans OPEN (4 coût, 6 = h, 2 _ 1 / 8 7 3 / 6 4 5 parent)

* Étape 5

Égalité : h = 6

blanc en indice 3 est le plus faible indice

2 x	1 x	6
8 ✓	7	3
6 x	4 x	5 ✓

h = f(n) = 6
coût = 4

OPEN ← {OPEN ; 2 1 3 / 8 7 / 6 4 5 } coût 5, h = 6, parent = étape 5

Le 1→ est déjà dans OPEN avec coût inférieur

Étape 5 (2 1 / 8 7 3 / 6 4 5 coût = 4 parent = 2 1 / 8 7 3 / 6 4 5) est ajouté dans CLOSED

* Étape 6

Égalité f(n) = h = 6 dans OPEN, l'indice min. du vide est 6

8 x	2 ✓	1 x
7 x	3 x	6
6 x	4 x	5 ✓

coût 5
f(n) = 6

- 8 2 / 7 3 1 / 6 4 5 est ajouté à OPEN : h = 6, coût = 5, parent : 8 2 1 / 7 3 / 6 4 5

- 8 2 1 / 7 3 / 6 4 5 est ~~ajouté~~ déjà dans CLOSED avec un coût = 0 < 5

- 8 2 1 / 7 3 5 / 6 4 est ajouté à OPEN, h = 7, coût = 6, parent : 8 2 1 / 7 3 / 6 4 5

pertinence fct. ⇒ qualité jeu

Question 2.2 1/2

La fonction heuristique peut avoir deux conséquences fâcheuses si elle est mal choisie :

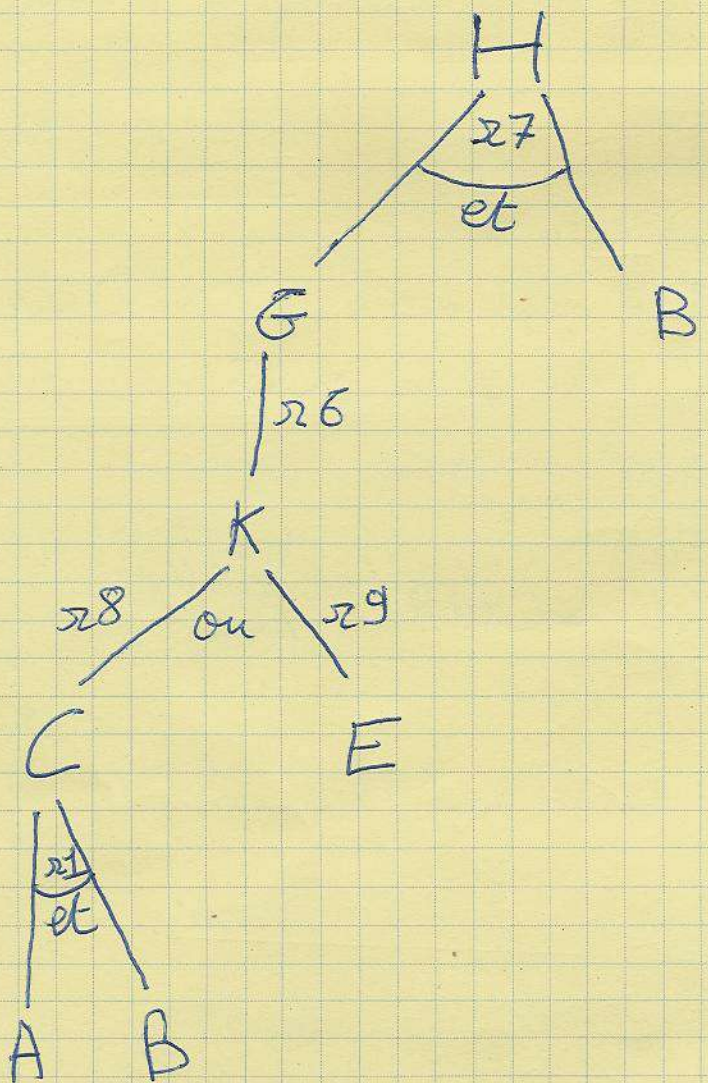
* ralentissement du programme (ici on voit que le partage en cas d'égalité n'est pas pertinent).

* ou pire, ne pas trouver la solution.

car c'est ce qui détermine chaque choix !

Question 3.1

2/2 ✓



On arrive à : ((A et B) ou E) et B

(true ou ?) et true

true et true

true

H est vrai

BLUTEAU
Pierre
L3/C
29/6/17

CE - AD (Suite et fin)

Question 3.2 1/1

- Il est important de se constituer un ensemble BFV de Faits Valides pour ne pas ~~réexécuter~~ retester (1000 fois) la validité des faits.

2 fois, c'est déjà trop !

- Il faut également ~~coder~~ marquer chaque règle utilisée, car on en a alors déduit tout ce qu'on pouvait.

- Appel récursif pour vérifier la validité d'un fait entraînant d'autres vérifications.

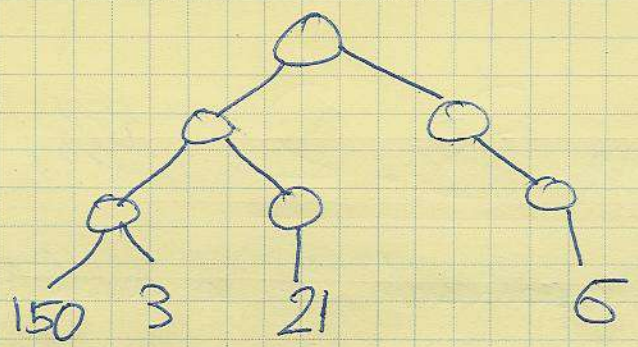
Question 3.3 0,5/1

J'ai implémenté le chaînage arrière en TP par un parcours en profondeur pour :

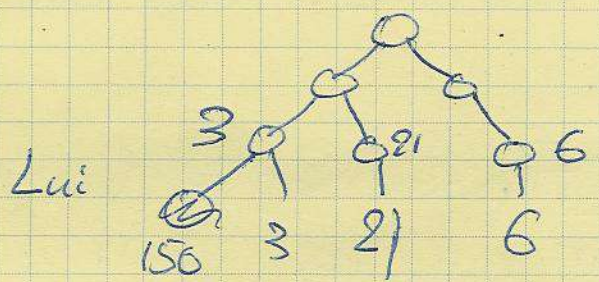
- la facilité d'implémentation
- le temps d'exécution (pas besoin de parcourir les deux branches d'un OU !)

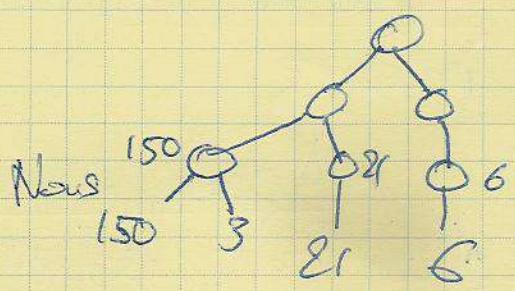
Question 4.1 2/2

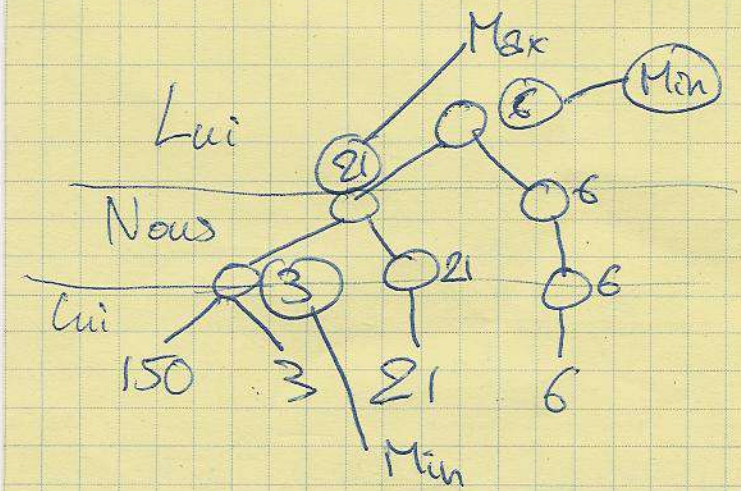
Deux adversaires jouent, B cherche à minimiser le score de A. A chaque étape, ils font un choix chacun leur tour. Les possibilités de score et l'arbre sont connus :

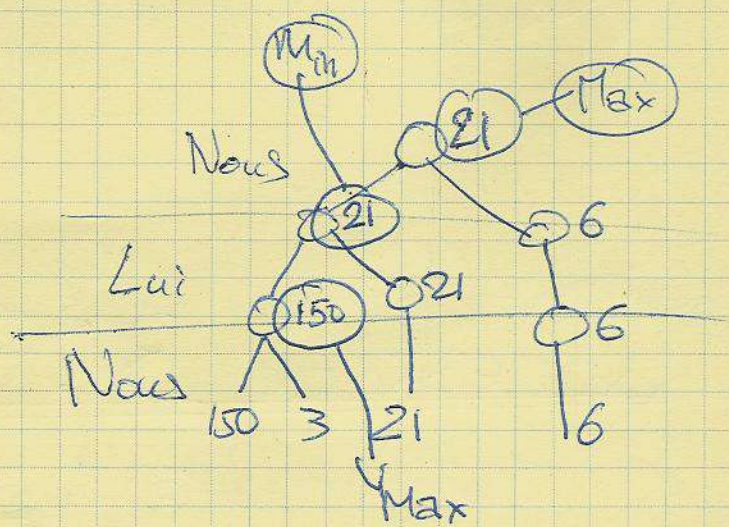


On suppose l'adversaire intelligent : il minimisera notre score. On part du dernier tour. A chaque tour, s'il joue, le noeud regait le min de ses fils, le max si c'est nous.







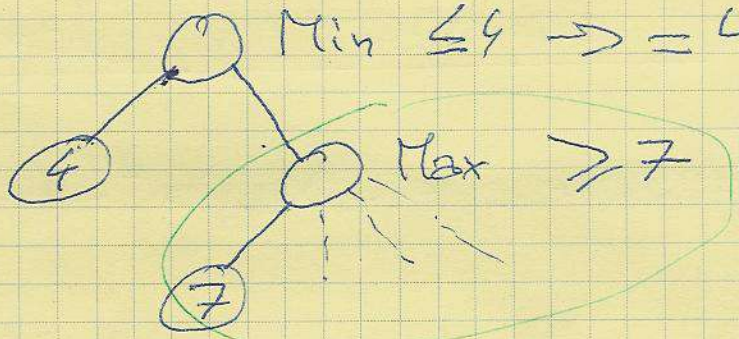


Il nous emmène à 6. On l'emmène à 21.

On obtient le chemin le plus sécurisé.

Question 4.2 2/2

L'élagage Alpha-Béta permet d'optimiser MinMax en ne calculant pas tous les noeuds. En effet, si on cherche le minimum de 4 et d'un noeud contenant 7, ce noeud contiendra au moins 7 (c'est un Max) et notre Min sera 4.

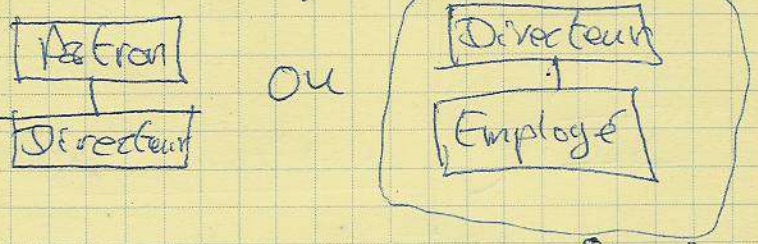


Lui

on peut arrêter le calcul, min=4

Question 5.1 3,5/4

- a_pour_responsable : on considère responsable comme "supérieur hiérarchique direct"



2 règles dans ce cas !

~~Service X~~ Direction X

SI ~~Aucune règle~~

(role(y, patron) et role(x, directeur))

ou (role(y, directeur) et direction(y, z) et direction(x, z))

Même direction

ALORS a_pour_responsable(?x, ?y)

- assure_le_support_pour
  - * x région z
  - * y région z
  - * x support
  - * y vente

SI (region (?x, ?z) et region (?y, ?z)
et direction (?x, support)
et direction (?y, support))
ALORS
assure_le_support_pour (?x, ?y)

- est_interface_support
  - * x support
  - * y client
  - * x région z
  - * y région z

SI (région (?x, ?z)
et direction (?x, support)
et client (?y, ?z))
ALORS
est_interface_support (?x, ?y)