[P22] Examen final IA02 + AI27

Préliminaires

- **Durée :** 2h.
- **Seul document autorisé :** deux feuilles A4 recto verso **manuscrites** (aménagements possibles pour certains 1/3 temps qui en ont préalablement fait la demande).
- **Barème indicatif**, susceptible de changement sans préavis **:** 2,5 + 4,5 + 4 + 4 + 4,5 (+ 0,5 point pour la clarté et la présentation de la copie).
- L'élégance et la qualité des solutions fournies fait partie intégrante des critères d'évaluation.
- **1 feuille par exercice**.
- Toute ambiguïté, erreur ou omission dans le sujet devra être résolue par le candidat et par lui seul.

Exercice 1 : DIMACS et résolution

On considère le fichier DIMACS suivant :

```
c pigeon-hole
p cnf 6 11
1 2 3 0
4 5 6 0
1 4 0
2 5 0
3 6 0
-1 -2 0
-1 -3 0
-2 -3 0
-5 -6 0
-4 -6 0
-4 -5 0
```

Question

1. Que représente ce fichier ?
2. Que signifient les 2 premières lignes ?
3. Montrer que l'on peut déduire la formule $x_1 \vee x_2$.
4. Quelle remarque pertinente pouvez-vous faire ?

Exercice 2 : Chou-fleur

Ce jeu de cour de récréation se joue à deux. Les joueurs se font face et sont à une distance *d* l'un de l'autre. Ils avancent chacun leur tour d'une certaine distance qui est retranchée de la distance initiale. Le premier joueur dit *chou* en avançant et le deuxième dit *fleur* et ainsi de suite. Le but du jeu est de marcher sur le pied de l'autre, ce qui arrive quand *d < 0*.

Dans le cadre de cet exercice, on considère que chaque joueur peut avancer d'uniquement 3 distances différentes d'une unité arbitraire.

Exemple : la distance initiale est *d = 5*, le joueur *chou* et le joueur *fleur* ne peuvent avancer que de 1, 2 ou 3. Le joueur *chou* avance de 3, puis le joueur *fleur* de 2, puis le joueur *chou* de 1. Le joueur *chou* gagne.

On supposera pour la suite que :

- Le joueur *chou* peut avancer des trois distances *{8, 6, 5}*.
- Le joueur *fleur* peut avancer des trois distances *{6, 5, 4}*.
- La distance initiale entre les 2 joueurs est *d = 14*.

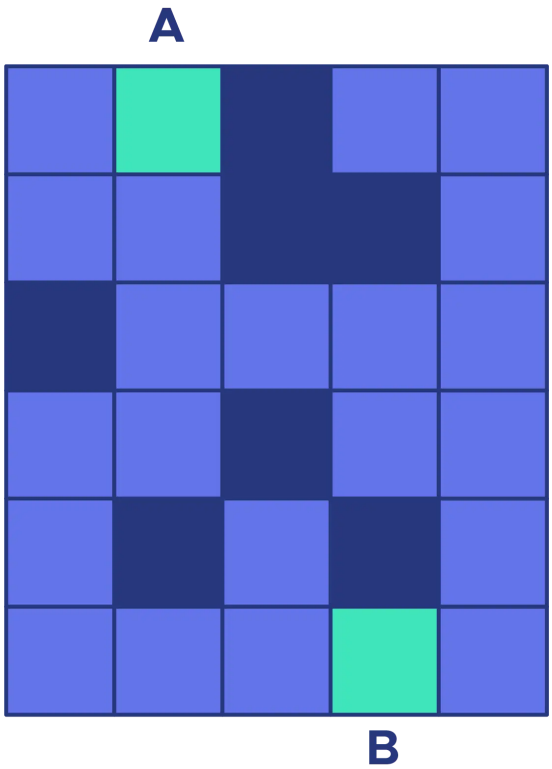
Questions

1. Modéliser le jeu chou-fleur comme un jeu à somme nulle.
2. Dans quel cas obtient-on un match nul ?
3. Résoudre ce jeu formel à l'aide de l'algorithme Min-Max. On veillera à construire et parcourir l'arbre du jeu en considérant les actions des joueurs dans l'**ordre de l'énoncé**.
4. Résoudre à l'aide de l'algorithme Alpha-Beta. *Même contrainte*.

NB : une autre façon, plus efficace, de traiter ce problème serait d'utiliser la programmation dynamique. Ce n'est pas ce qui est demandé ici.

Exercice 3 : labyrinthe et A*

Dans la grille suivante, les cases sombres sont inaccessibles. On repérera les cases en *notation matricielle* : la case initiale A est sur la 1e ligne et sur la deuxième colonne donc A(1,2) et la case but B a pour coordonnées (6,4).



Soit (i,j) la position de l'état courant et (i',j') celle du but (constant). On considère les trois heuristiques suivantes :

- $a(i,j) = 0$
- $b(i,j) = |i - i'| + |j - j'|$
- $c(i,j) = max(|i - i'|, |j - j'|)$

On suppose que les successeurs d'une case partagent un côté avec celle-ci (c.-à-d. on ne se déplace pas en diagonale) et sont parcourues dans l'ordre haut-droite-bas-gauche.

Questions

1. Calculer dans un tableau la valeur des différentes heuristiques pour chacune des positions données.
2. Quelles heuristiques sont minorantes ? Pourquoi ?
3. Que se passe-t-il si l'on utilise l'heuristique a associée à A* ?
4. Appliquer l'algorithme A* avec l'heuristique c en partant de A, en arrivant à B. Pour cela, on énumérera successivement la file à priorité des états à visiter en soulignant l'état visité à la fin de chaque étape. On considérera également qu'on ne revient pas directement à la position d'où l'on vient.

Exercice 4 : traitement de dates en Prolog (seulement pour IA02)

On souhaite représenter et traiter des dates en Prolog sous la forme d'une fonction `date/3`. Pour rappel on peut toujours unifier les éléments d'une fonction à l'intérieur d'un prédicat.

```
%%% exemple %%%
?- p(f(12,13)) = p(f(X,Y)).
X = 12,
Y = 13.
?- p(f(12,13)) = p(f(12,13)).
true
```

1. Écrire un prédicat `jour(+D,-J)` (resp. `mois(+D,-M)`, `année(+D,-A)`) qui étant donné une date `D` unifie `J` (resp. `M`, `A`) avec le jour de la date.

```
%%% exemple %%%
?- jour(date(29,12,2010), J).
J = 29.
?- mois(date(29,12,2010), M).
M = 12.
?- année(date(29,12,2010), A).
A = 2010.
```

2. Écrire un prédicat `valide(D)` qui renvoie vrai si le jour est compris entre 1 et 31, le mois entre 1 et 12 et l'année quelconque. *NB :* le prédicat prédéfini `integer(+X)` est vrai si et seulement si `X` est un entier.

```
%%% exemple %%%
?- valide(date(12,12,2012)).
true
?- valide(date(31,2,-512)).
true
?- valide(date(31,13,1024)).
false
?- valide(date(31,2,toto)).
false
```

3. Écrire un prédicat `eq(+D1, +D2)` qui est vrai si la date `D1` et la date `D2` sont identiques.

```
%%% exemple %%%
?- eq(date(12,12,2012),date(12,12,2012)).
true
```

4. Écrire un prédicat `avant(+D1, +D2)` qui est vrai si la date`D1`précède (est strictement plus ancienne que) la date `D2`.

```
%%% exemple %%%
?- avant(date(09,06,1977),date(03,04,1978)).
true
?- avant(date(09,06,1977),date(03,09,1977)).
true
?- avant(date(09,06,1977),date(10,06,1977)).
true
?- avant(date(03,04,1978),date(09,06,1977)).
false
?- avant(date(09,06,1977),date(09,06,1977)).
false
```

5. Écrire un prédicat `plus_récente(+L, -D)` qui, étant donné un liste de dates `L` unifie la date la plus récente avec `D`.

```
%%% exemple %%%
?- plus_récente([date(03,04,1978),date(22,12,2010),
                 date(09,06,1977),date(29,12,2008)], D).
D = date(22,12,2010)
```

Exercice 4' : PIC et final en logique du premier ordre (seulement pour AI27)

> Si les étudiants ont assez de temps, alors ou bien ils étudient le final d'AI27, ou bien ils vont au PIC. Si ils ne réussisent pas leur final, c'est qu'ils n'ont pas étudié et qu'en plus ils n'ont pas eu de chance. Khaled n'a pas été au PIC, mais il n'a pas réussi son final.

Questions

1. Modéliser cet énoncé en logique du premier ordre. Commencer en explicitant le vocabulaire.
2. Mettre le problème sous forme normale.
3. Montrer que cet énoncé est cohérent en exhibant un modèle.
4. Montrer, en utilisant les principes de résolution et de réfutation, que Khaled n'a pas eu de temps et qu'il n'a pas eu de chance.

Problème : M. Patate en ASP

> M. Patate est un jouet destiné aux enfants permettant de créer différentes combinaisons amusantes sur un corps en forme de pomme de terre (familièrement appelée patate).

M. Patate est constitué d'un corps. Il a aussi, **nécessairement** :

- des yeux
- une bouche
- des bras (forcément de même couleur)
- des chaussures

M. Patate peut aussi être dôté d'accessoires supplémentaires *optionnels* :

- des oreilles
- des lunettes
- un parapluie
- des cheveux ou un couvre-chef

En outre on souhaite modéliser les contraintes suivantes :

- (C1) pour porter des lunettes, il doit posséder des oreilles
- (C2) un Monsieur Patate à chapeau melon possédera forcément un parapluie
- (C3) on ne peut pas porter à la fois des lunettes de soleil et un parapluie

Dans la boîte on dispose des éléments suivants.

- 2 paires d'yeux : des grands et des petits
- 2 bouches : une rieuse, une fermée
- 2 paires de bras : des rouges et des blancs
- 2 couvre-chefs : un chapeau melon et une casquette
- 2 paires de lunettes : des lunettes bleues et des lunettes de soleil
- un parapluie noir
- 3 types de chaussures : des mocassins noirs, des sandales, des baskets (on ne peut mettre qu'un seul type de chaussures à la fois)

Question

1. Écrire un programme ASP dont l'ensemble des modèles correspond à l'ensemble des configurations possibles d'un M. Patate. On utilisera pour cela la liste de prédicats suivants :
    - pour les éléments obigatoires : `yeux/1`, `bouche/1`, `bras/1`, `chaussures/1`
    - pour les accessoires facultatifs : `couvre_chef/1`, `lunettes/1`, `parapluie/1`
    - `monsieur_patate/1` : qui permet de nommer un M. Patate (ex : `monsieur_patate(sylvain)`)
    - `poss(M, E)` : qui signifie que le M. Patate `M` possède l'élément `E`
2. Comment modifier les prédicats précédents pour prendre en compte deux M. Patate ?
