Ce TP cherche utiliser Prolog pour résoudre deux problèmes concrets. Le premier problème cherche à implémenter un jeu de devinette où l'utilisateur choisi un nom dans un liste et l'ordinateur doit trouver le nom choisi à l'aide de questions oui/non. La seconde cherche à résoudre un problème logique sur une grille.

1 Projet 1

1.1 Description

Dans le cadre de ce travail pratique, vous développerez une base de connaissances qui sera utilisée pour un jeu de devinette. Lorsque votre programme s'exécutera, il demandera de penser à une personne ou un objet pouvant se trouver dans une maison. En posant des questions qui se répondent par oui ou non, il essaiera de deviner de quelle personne ou quel objet il s'agit. En fait, plus précisément, il s'agira de deux programmes distincts, chacun ayant sa base de connaissances : un programme pour les personnes et un autre pour les objets.

1.2 Travail à réaliser

Vous développerez vos deux programmes en utilisant SWI-Prolog. Les deux bases de connaissances doivent permettre de deviner n'importe quelle des entités suivantes :

1.3 Personnes:

Michael Jackson	Julie Snyder
Stephen Harper	Moise
Wolgang Amadeus Mozart	Mario
Michel Gagnon	Lara Croft
Blanche-Neige	J.K. Rowling
Rafal Nadal	Jacques Villeneuve
Garfield	Cléopâtre VII
Michel Dagenais	Victor Hugo
John Lewis	Jésus
Pape François	Eugénie Bouchard
James Bond	Brad Pitt

1.4 Objets

Aspirateur	Ordinateur
Téléphone	Fourchette
Balai	Cactus
Assiette	Four micro-onde
Cuisinière	Cafetière
Grille-pain	Table
Casserole	Shampoing
Détergent à vaisselle	Lit
Clé	Porte-feuille
Sac à dos	Piano
Lampe	Papier

Veuillez noter qu'il est possible que d'autres entités soit ajoutées d'ici la date de remise. Vous devrez donc vous assurer que votre base de connaissances soit suffisamment bien construite pour faciliter l'ajout de nouvelles entités. Votre programme devra minimiser le nombre de questions posées. Pour vous aider à démarrer, voici un petit exemple, comprenant la description de 4 personnes :

```
ask(gouverne,Y):-
format('Gouverne ~w ? ',[Y]),
    read(Reponse),
    Reponse = 'oui'.
ask(musicien,X):-
format('~w est un musicien? ',[X]),
    read(Reponse),
    Reponse = 'oui'.
ask(chanteur,X):-
format('~w est un chanteur? ',[X]),
    read(Reponse),
    Reponse = 'oui'.
personne(X) :-
    politicien(X).
personne(X) :-
    artiste(X).
artiste(X) :-
    ask(chanteur,X),
    chanteur(X).
artiste(X) :-
    ask(musicien,X),
    musicien(X).
politicien(X) :-
    gouverne(X,Y),
    pays(Y),
```

```
    ask(gouverne,Y).
chanteur(celine_dion).
musicien(john_lewis).
gouverne(stephen_harper,canada).
gouverne(barack_obama,usa).
pays(canada).
pays(usa).
```

Pour exécuter votre programme, il suffira de faire la requête ?- personne(X) ou ?-objet(X). Aussi, si on désactive les questions posées à l'utilisateur, on devrait pouvoir déterminer si une entité appartient à une certaine classe. On pourrait imaginer, par exemple, que le programme réponde positivement à une requête comme celle-ci :

```
?-appareil_electromenager(four_micro_onde).
```

2 Projet 2

2.1 Introduction

Pour ce projet vous devez résoudre le problème logique suivant : On vous fourni une grille initialement vide. Le but du problème est de la remplir à en noircissant certaines cases pour faire apparaître un dessin. Pour savoir quelles cases noircir on vous fourni pour chaque ligne et colonne une série de chiffres. Ces chiffres indiquent la taille des blocs de cases noires de la ligne ou de la colonne sur laquelle ils se trouvent. Au moins une case blanche sépare deux blocs noirs.

On a par exemple les configurations suivantes:

Ligne 

Chaque case jaune indique le nombre de carrés noirs consécutif dans la ligne, dans l'ordre dans lequel ils se trouvent. Ainsi dans cet exemple on trouve de gauche à droite un bloc de 1 case noire, puis un de 3 puis un de 2.

Colonne 

Chaque case jaune indique le nombre de carrés noirs consécutif dans la colonne, dans l'ordre dans lequel ils se trouvent. Ainsi dans cet exemple on trouve de haut en bas un bloc de 3 cases noire, puis un de 2.

On peut remarquer que :

- au moins une case blanche sépare deux blocs noirs
- la donnée des blocs d'une seule ligne ou colonne ne permet pas, de façon générale, de compléter de façon unique la ligne ou la colonne, seule la combinaison des données lignes/colonnes le permet.

La figure 1 illustre une grille vide avec le nombre donné de blocs qui doivent être peints dans chaque rangée et colonne, et la version terminée où les blocs peints forment une image d'une tasse de café.

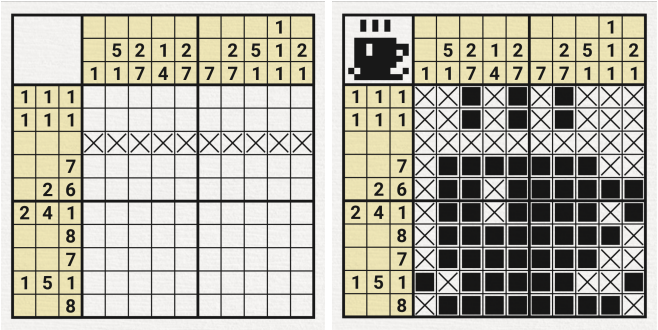

Figure 1: exemple de grille vide et complétée

2.2 Implémentation

Pour cette partie, on vous demande d'implémenter une méthode logicPrb(ColumnSpecs, LineSpecs, Lines) où, ColumnSpecs est une liste de listes, chaque liste correspondant aux blocs d'une colonne, LineSpecs est une liste de listes, chaque liste correspondant aux blocs d'une ligne, et Lines est la liste de listes initialement non connue, qui représente la grille. Lines représente la grille ligne par ligne. Ainsi pour résoudre le puzzle vous exécuterez la commande suivante :

```
logicPrb([C1,C2,C3], [L1,L2,L3], X).
```

Où C_j et L_i sont les contraintes sur les lignes i et les colonnes j respectivement.

2.3 Pistes de résolution

Tout d'abord, prenons la convention de représenter la matrice X comme une liste de listes. Nous devons alors implémenter à partir de là quelques fonction pour nous aider à résoudre le puzzle.

1. `valid_seq(CONSTRAINTS,SEQ)`
   Créez une fonction qui à partir d'une liste de contraintes (i.e. une liste de LineSpecs ou de ColumnSpecs) et une liste SEQ, qui représente soit une ligne soit une

colonne (suivant que CONSTRAINTS soit dans LineSpecs ou dans ColumnSpecs) et qui vérifie si la liste SEQ respecte les contraintes CONSTRAINTS.

2. `valid_lines(...)`
   Créez une fonstion qui parcours les lignes de X et qui vérifie pour chacun d'elles, qu'elles sont valides (par rapport à LinesSpecs). Ici il faut aussi vérifier que les éléments de la matrice ne prennent que deux valeurs (noir ou blanc). On choisira ici de représenter le noir par 1 et le blanc par 0. On peut fixer cette contrainte en utilisant la commande suivante, qui s'applique sur une liste (et pas sur une liste de listes) :

   ```
   LINE ins 0..1
   ```

3. `extract(K,X,COLUMN)`
   Comme X est une liste de lignes, il nous faut extraire les colonnes pour les tester. Cette fonction extrait donc la *K*-ème colonne de X, et la renvoie dans le champ *COLUMN*. Vous aurez besoin de la fonction (déjà dans prolog) suivante pour extraire le i-ème élément d'une liste L : `nth1(i,L,ELEMENT)`.

4. `valid_columns(...)`
   Créez ensuite la fonction analogue à `valid_lines(...)` mais pour les colonnes.

Maintenant vous pourrez résoudre le puzzle avec le code suivant :

```
logicPrb(ColumnSpecs, LineSpecs, X):-
valid_lines(...),
valid_columns(...),
print_nonogram(X).
```

Remarque : Les temps de résolution ont une complexité exponentielle en la taille de X, il se peut que certaines résolution prennent du temps. Les grilles 5x5 sont à peu près immédiates, les plus grosses prennent plus de temps. Si vous le souhaitez vous pouvez améliorer l'algorithme avec des heuristiques pour l'aider à trouver la solution plus rapidement. Vous pouvez par exemple noircir toutes les cases dont vous êtes sur qu'elles le sont.

Vous utiliserez la fonction suivante pour afficher la solution du problème.

```
print_nonogram(N) :-
    nl,write('Found nonogram:'),nl,
    print_nonogram1(N).

print_nonogram1([]).
print_nonogram1([Line | Lines]) :-
    print_line(Line),nl,
    print_nonogram1(Lines).
```

```
print_line([]).
print_line([Head | Tail]) :-
    Head = 1,
    write('# '),
    print_line(Tail).
print_line([Head | Tail]) :-
    Head = 0,
    write('. '),
    print_line(Tail).
```

2.4 Exemples de test

Voici quelques grilles que vous pouvez essayer de résoudre, à la main pour comprendre le problème puis à l'aide de Prolog.

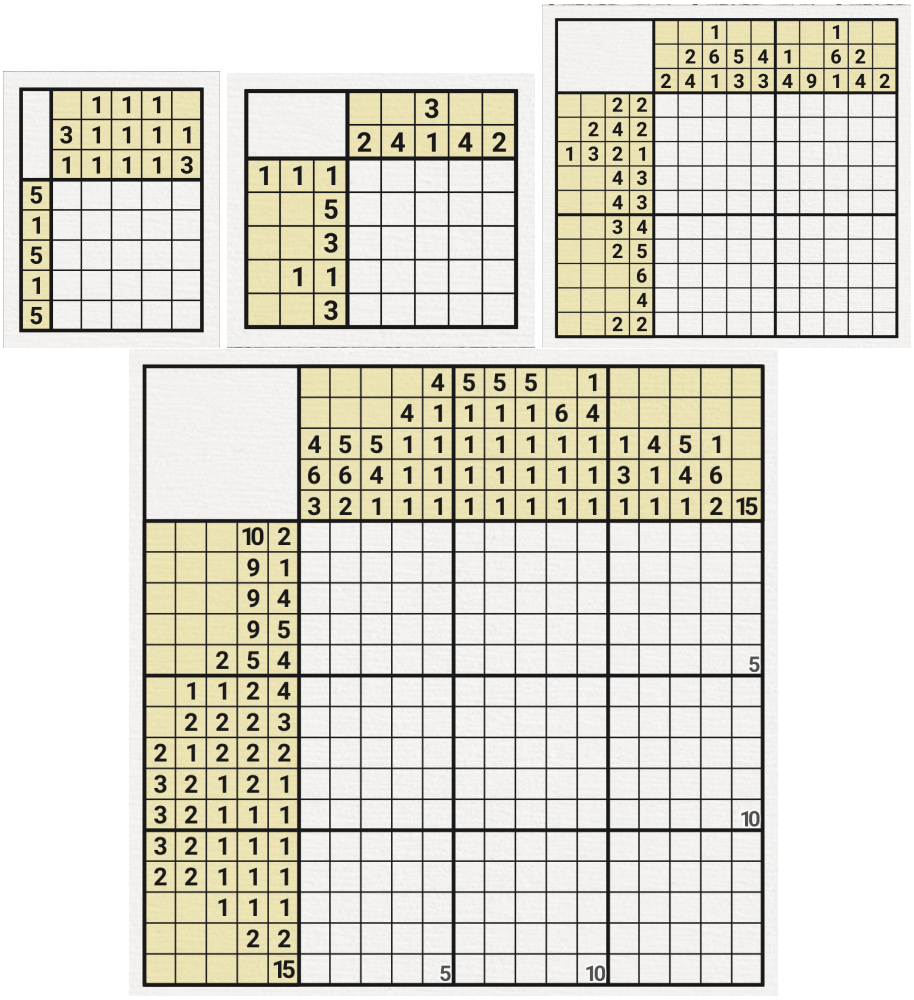

Figure 2: exemple de grille

3 Directives de remise

Le travail sera réalisé en équipe de deux ou trois. Vous remettrez un fichiers .zip par groupe, nommés TD1_nom_prenom_matricule.zip. Vous devez remettre un fichier pdf contenant une explication de vos implémentations ainsi que tous les commentaires que vous jugerez utiles et le code Prolog. La clarté et la concision du rapport seront pris en compte. Tout devra être remis avant le **25 mars à 23h55**. Tout travail en retard sera pénalisé d'une valeur de 10% par jour de retard.

Barème :

Projet 1 :	5pts
Projet 2 :	40pts
rapport :	10pts

*Ce TP est une repise du TP2 de l'hiver 2016 complété par un projet imaginé par Pierre Hulot et Rodrigo Randel