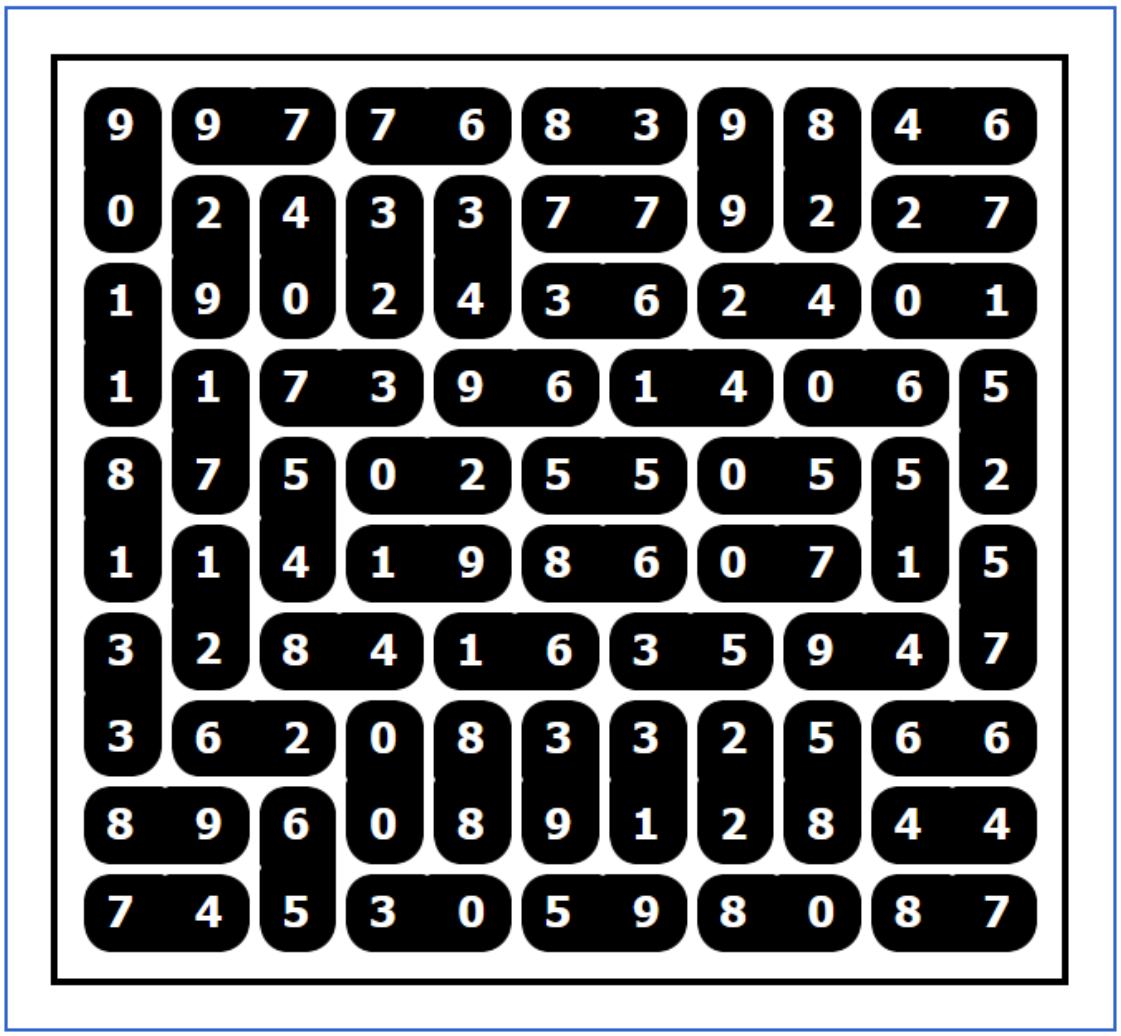
Aguesse Nathan
21001877

Projet Final - Prolog

Solveur de Dominosa

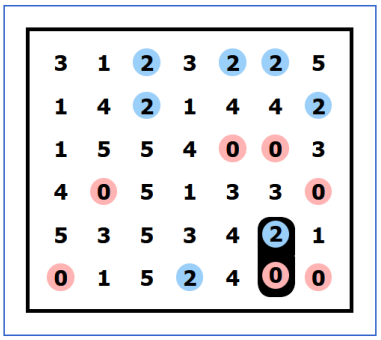
Programmation déclarative et bases de données



1 - Présentation du Projet

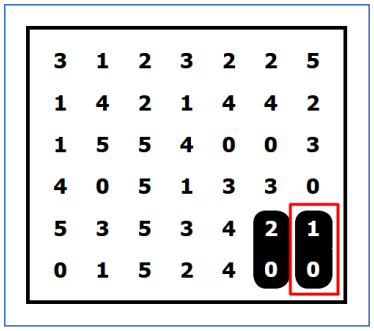
Le Dominosa, une “variante” peu connue des Dominos, consistant à retrouver tout un set de dominos dans une grille de chiffres.
Concrètement, on ne peut pas voir 2 mêmes domino dans la grille, même principe que pour le sudoku, on va devoir poser des dominos lorsqu’on est sûr qu’il soit valide.

1er cas :



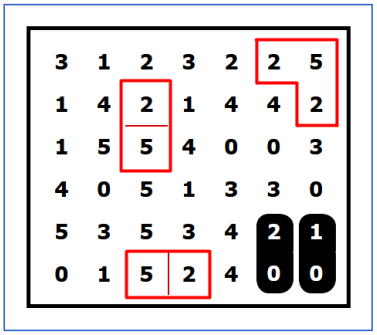
Dans cet exemple, on recherchant toutes les paires de 0 et 2, on s’aperçoit qu’il n’existe qu’une seule possibilité, on va donc poser un domino à cet endroit.

2ème cas :



Ce cas est assez simple à remarquer, lorsque vous voyez qu'un domino ne peut être posé d’une seule façon (lorsqu’elle est bloquée autour d’elle), vous pouvez donc mettre le domino étant donné qu’il n’existe pas d’autre possibilité.

3ème cas :



On passe maintenant à un certain cas ne permettant pas de poser un domino, mais de **réduire les possibilités** sur la grille, permettant peut-être de déverrouiller de nouvelles possibilités autre part, sur cette exemple, on remarque que dans le coin on a pour **seul possibilité** d’avoir un domino de 2 et 5, **donc** autre part on ne pourra pas avoir d’autre dominos avec 2 et 5.

2 - Structure du code

La première question qu'il fallait répondre fut l'apparence de la grille de départ et de fin, une matrice était indispensable, mais la grille de départ étant déjà remplie, comment faire pour la grille de fin ? Eh bien, à la fin on est censé avoir tous les **liens** entre chaque dominos, donc il faut juste afficher une autre grille indiquant la direction des liens, avec les directions cardinaux, pour avoir une solution correcte :

```
[[1,2,1,0,3],     [['E','W','S','E','W'],      [[0 ,1 ,2 ,3 ,4 ],
 [2,1,1,0,2],      ['S','S','N','E','W'],       [5 ,6 ,7 ,8 ,9 ],
 [3,3,3,3,0],      ['N','N','E','W','S'],       [10,11,12,19,14],
 [0,1,2,2,0]]      ['E','W','E','W','N']]       [15,16,17,18,19]]
```

(Matrice de début) (Matrice de fin) (Matrice des IDs)

(Effectivement, en plus de la matrice de début et de fin, un autre point de vue de la matrice avec des IDs unique pour chaque case nous permet de mieux les différencier, malgré le fait que l'on va pas avoir cette matrice en elle-même dans le code, cependant ces IDs seront très vitale et utilisé)

Ensuite, on commence à construire des prédicats qui vont chacunes servir a quelque chose dans notre code,

Que ce soit des ptits prédicats comme rows/columns pour obtenir le nombre de ligne/colonnes de la matrice :

```
rows([],0).
rows([_|B],S):-
    rows(B,S1),
    S is S1+1.
```

Ou d'un prédicat assez important pour le solveur, tout en utilisant d'autre prédicats :

```
distinct([A|_],M,A):-
    nmb(A,M,N),
    N == 1,!.
distinct([A|B],M,L):-
    nmb(A,M,N),
    N \= 1,
    distinct(B,M,L).
```

Il y en a pas mal… (environ 25)

Mais la structure principale du code va se trouver dans les deux prédicats "solution" et "dominosa" :

solution(M,S) - Va prendre la matrice de départ "M", et renvoie sa solution "S", pour ce faire, elle va appeler de nombreux prédicats, lui permettant d'obtenir de nombreuses informations sur la grille de jeu, avec, il va passer le relaie au prédicat "dominosa" qui va réussir à obtenir la solution désiré avec les informations reçu.

```
solution(M,S):-
    rows(M,R),
    columns(M,C),
    RC is R*C,
    create_list(RC,ListeID),
    check_element(M,ListeN),
    dominos(ListeN,ListeN,SetDomino),
    poss(M,LD),
    id(0,0,R,C,0,LI),
    dominosa(M,S,R,C,LD,LI,ListeID,SetDomino,SetDomino).
```

Liste des informations données en paramètres :
M - La matrice de départ
S - La solution qui sera envoyé par dominosa
R - Nombre de lignes
C - Nombres de colonnes
LD - Toutes les possibilités de dominos posables
LI - La même liste, sauf que les valeurs des dominos seront remplacé par leurs IDs
ListeID - La liste de tous les IDs des cases
SetDomino - Le set de domino du jeu, donc tout les domino que l'on devra placer

dominosa(M,S,R,C,LD,LI,ListeID,SetDomino,SetDomino) - Après avoir reçu toutes les infos requises, le principe de dominosa va être de jouer chaque coup possible un par un, jusqu'à arriver à la fin, pour ce faire, on va donc s'occuper des 3 cas possibles expliqué précédemment :

A noter que pour chaque prédicats en première position (distinct/distinct_id/check_same), en plus d'obtenir des infos que l'on veut, elle sont la condition pour que chaque cas s'exécute (donc retourne "false" dès le début, si le cas n'est pas faisable).

1er cas :

```
dominosa(M,S,R,C,LD,LI,ListeID,SetDomino,SetDominoSPE):-
    distinct(SetDomino,LD,D),
    domino_id(D,LD,L,LI),
    change_m(M,M1,L),
    sup_domino(SetDomino,NSetDomino,D),
    sup_domino(SetDominoSPE,NSetDominoSPE,D),
    sup(L,LI,LD,NLI,NLD),
    dominosa(M1,S,R,C,NLD,NLI,ListeID,NSetDomino,NSetDominoSPE),!.
```

Si l'on trouve un Domino distinct, (D)
On trouve son ID, (L)
On ajoute la solution pour ce domino dans la matrice (M > M1)
On supprime le domino du Set (SetDomino > NSetSomino)
De même pour l'autre Set (SetDominoSPE > NSetSominoSPE)
On supprime toutes les possibilités qui sont maintenant inexistante (LD > NLD, LI > NLI)
Et on rappelle dominosa, avec la nouvelle Matrice, listes de possibilités et des sets.

2ème cas :

```
dominosa(M,S,R,C,LD,LI,ListeID,SetDomino,SetDominoSPE):-
    distinct_id(ListeID,LI,L),
    change_m(M,M1,L),
    domino_id(D,LD,L,LI),
    sup_domino(SetDomino,NSetDomino,D),
    sup_domino(SetDominoSPE,NSetDominoSPE,D),
    sup(L,LI,LD,NLI,NLD),
    dominosa(M1,S,R,C,NLD,NLI,ListeID,NSetDomino,NSetDominoSPE),!.
```

Si l'on trouve un ID "distinct", (L)
On ajoute la solution pour ce domino dans la matrice (M > M1)
On trouve son Domino, (D)
On supprime le domino du Set (SetDomino > NSetSomino)
De même pour l'autre Set (SetDominoSPE > NSetSominoSPE)
On supprime toutes les possibilités qui sont maintenant inexistante (LD > NLD, LI > NLI)
Et on rappelle dominosa, avec la nouvelle Matrice, listes de possibilités et des sets.

3ème cas :

```
dominosa(M,S,R,C,LD,LI,ListeID,SetDomino,SetDominoSPE):-
    check_same(SetDominoSPE,LD,LI,N,L),
    sup_spe(N,L,LI,LD,NLI,NLD),
    sup_domino(SetDominoSPE,NSetDominoSPE,L),
    dominosa(M,S,R,C,NLD,NLI,ListeID,SetDomino,NSetDominoSPE),!.
```

On regarde si une seule possibilité n'est disponible pour chaque ID (et on retourne l'ID trouvé)
On supprime les possibilités des autres ID a part celle trouvé pour le Domino
On supprime le domino du Set spécial (utilisé seulement pour ce cas la)
Et on rappelle dominosa, avec les nouvelles listes de possibilités et du set

Puis on a le cas de fin, qui arrive lorsque la liste des possibilités/id est vide (donc la grille est pleine, plus de place pour poser de dominos, donc 0 possibilité restantes), et de même pour la liste du set de domino, après les avoir tous poser, il ne nous en reste plus.
On va donc pouvoir ajouter la touche finale :

```
dominosa(S,S,_,_,[],[],_,[],_):-!.
```

Alors que l'on appelait toujours dominosa avec (M,S,...), maintenant que M est devenue en soit, la solution après avoir été modifiées par tous les coups le que le prédicat a effectuer, on va donc pouvoir dire que M = S, donc "dominosa(S,S,...)", ce qui va permettre à la solution de remonter tous les appels que l'on avait effectué, pour pouvoir donner la solution au joueur.

3 - Temps d'exécution

On peut observer les temps d’exécutions que prend le programme en fonction de l’importance de la grille de jeu (effectué avec 10 tests pour chaque taille de grilles) :

Nmb de chiffre :	6	8	10	12
Temps Moyen (s)	0.0065 (s)	0.0354 (s)	0.116 (s)	0.331 (s)
Écart-type (s)	0.00085 (s)			

Processeur utilisé :
- Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz