PROJET ALGORITHME AVANCEE

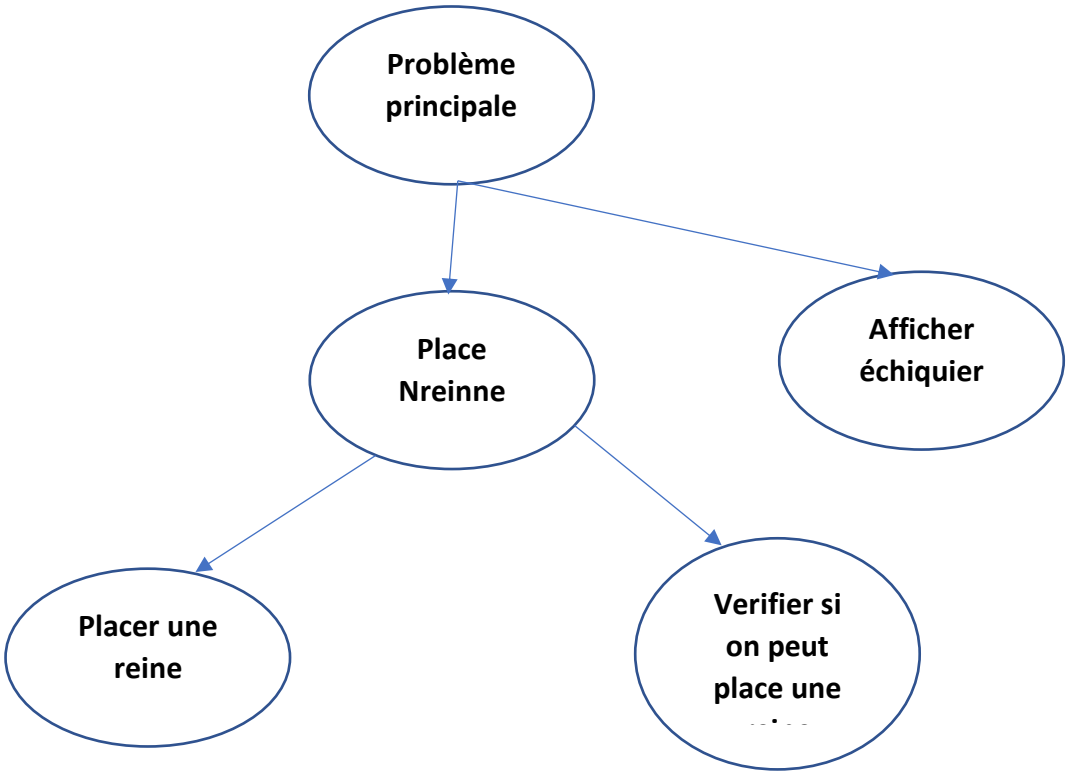
N_queen problem

Bacem Abroug
2DNI 2

Objectif :

- Création d'un algorithme récursive qui permet de placer N reines dans une échiquier de taille N

Décomposition de Problème :

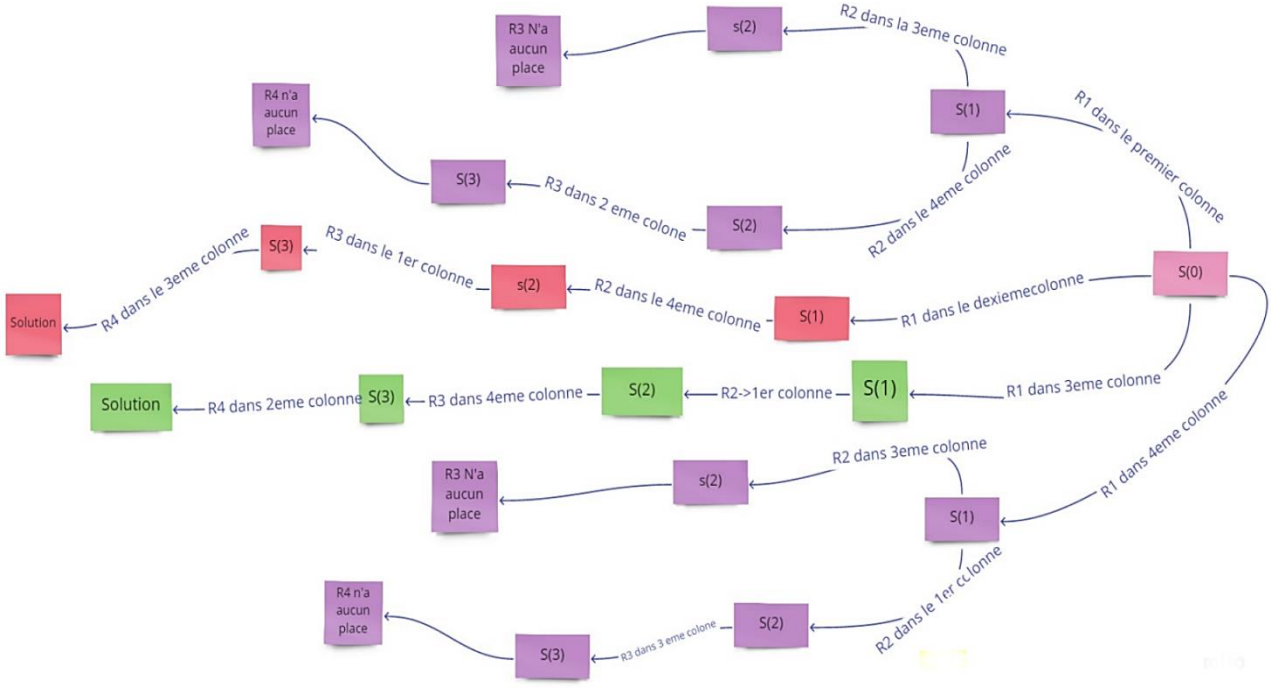


Contraintes :

- La représentation algorithmique de l'échiquier est une matrice.
- Chaque ligne de matrice contient une seul reine.
- Chaque reine placée ne peut pas être attaquer par les autres
- Les valeurs de matrice doit contient une valeur de les valeur suivantes :
  - ✓ 0 : pour indiquer une case libre.
  - ✓ 1 : pour indiquer une case interdite.
  - ✓ y : pour indiquer une case occupée par la reine numéro y (y = 1..n)
- Déterminer tous les cas possibles

Travail demander :

1. Formuler le problème selon le principe «Diviser pour régner» :
   - Pour N=4 :



R1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1

R1	1	1	1
1	1	R2	1
1	1	1	1
1	0	1	1

R1	1	1	1
1	1	R2	1
1	1	1	1
1	0	1	1

No place for R3

1	R1	1	1
1	1	1	0
0	1	0	1
0	1	0	1

1	R1	1	1
1	1	1	R2
0	1	1	1
0	1	0	1

1	R1	1	1
1	1	1	R2
R3	1	1	1
1	1	0	1

1	R1	1	1
1	1	1	R2
R3	1	1	1
1	1	R4	1

Première solution pour N=4

Algorithme pour placer une reine :

```
Def FN place_queen(E,R,i,j,k←1)
debut
remplir(E,i,j,k)
E[i][j]←R
return E
fin
```

Python

```
def place_queen(E,R,i,j,k=1):
    remplir(E,i,j,k)
    E[i][j]=R
    return E
```

```
algorithme remplir(T,i,j,k←1)
  debut

   pour l allant de 0 à n-1
      T[l][j]←k
   pour c allant de 0 à n-1
      T[i][c]←k
   ii, jj ← i+1, j+1
   l ← len(T)
   Tant que ii < l et jj < l
      T[ii][jj]←k
      ii, jj ← ii + 1, jj + 1
   row←i
   col←j
   ii, jj ← row+1, col-1
   Tant que ii < l et jj ≥ 0
      T[ii][jj]←k
      ii, jj ← ii + 1, jj - 1



   row←i
   col←j
   ii, jj ← row-1, col+1
   Tant que ii ≥ 0 et jj < l



      T[ii][jj]←k
      ii, jj ← ii - 1, jj + 1



   return T
fin
```

Python

```
def remplir(T,i,j,k=1):



   for l in range(0,n):
      T[l][j]=k
   for c in range(0,n):
      T[i][c]=k
   ii, jj = i+1, j+1
   l = len(T)
   while ii < l and jj < l:
      T[ii][jj]=k
      ii, jj = ii + 1, jj + 1
   row=i
   col=j
   ii, jj = row+1, col-1
   while ii < l and jj >= 0:
      T[ii][jj]=k
      ii, jj = ii + 1, jj - 1



   row=i
   col=j
   ii, jj = row-1, col+1
   while ii >= 0 and jj < l:



      T[ii][jj]=k
      ii, jj = ii - 1, jj + 1



   return T
```

Algorithme pour place toutes les reines :

```
algorithme place_queens(board, row←0)
debut
   global nSolutions
   si row ≥ len(board)
      afficher _board(board)
      nSolutions =nSolution+1
      return
pour col dans board

      si board[row][col]=0
         si can_place_queen(board, row, col)

            place_queen(board,'q',row,col)
            place_queens(board, row + 1)

            place_queen(board,0,row,col,0)
```

Python

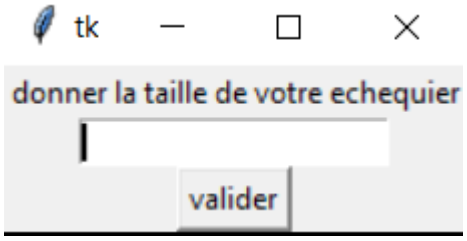
```
def place_queens(board, row=0):
   global nSolutions
   if row >= len(board):
      print_board(board)
      nSolutions += 1
      return
   for col, field in enumerate(board):

      if board[row][col]==0:

         if can_place_queen(board, row, col):

            place_queen(board,'q',row,col)
            place_queens(board, row + 1)
```

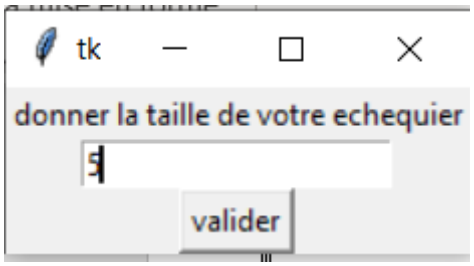
Algorithme principale :

```
nSolutions = 0
n=5
board = [[0] * n for i in range(n)]
place_queens(board)
ecrire("Found”, nSolutions,
"solutions!" )
```

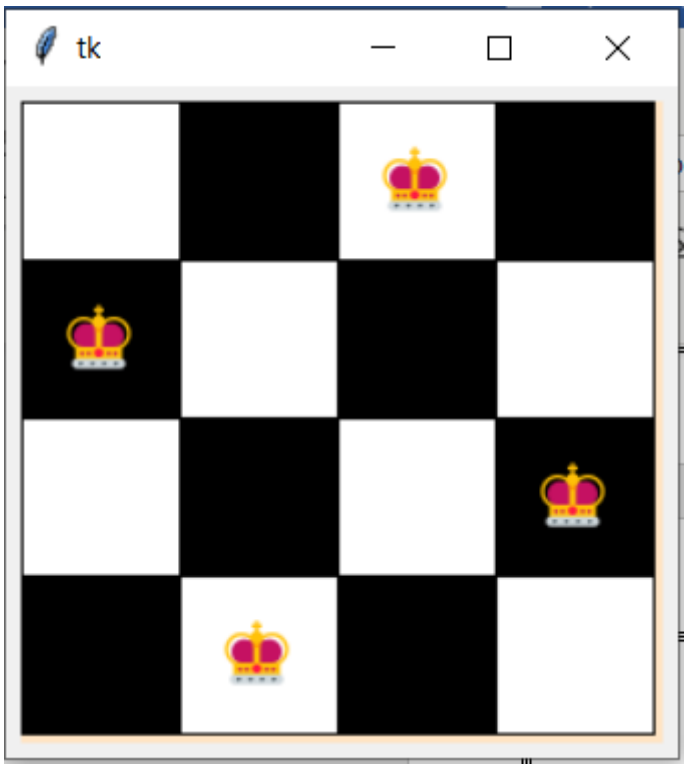
Partie d’affichage de solution en python :

La première interface nous permet de saisir la taille d’échiquier :





Le clic du bouton valider la deuxième interface sera lancé qui affichera l’échiquier et les positions des reines:



A propos les programmes en python :

Le fichier ‘recurcivite.ipynb’ est un programme en python sous l’extension jupyter notebook nous permet de placer toutes les reines et afficher toutes les solution possible

```
nSolutions = 0
n=4
board = [[0] * n for i in range(n)]
place_queens(board)
print("Found %s solutions!" % nSolutions)
```

```
1Q11
111Q
Q111
11Q1

11Q1
Q111
111Q
1Q11

Found 2 solutions!
```

Le fichier ‘recurcivite_display.py’ est un programme en python qui nous permet d’afficher les interface graphique