

T.I.P.E: Rubik's Cube

2x2x2

▼ I) Définitions des constantes et opérations de bases.

Les constantes sont en principe écrites tout en majuscule, à l'exception des plus usitées qui sont alors tout en minuscule.

*On charge tout d'abord le module utilisé:

with(plots);

[*animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d,* (1.1.1)
conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d,
densityplot, display, dualaxisplot, fieldplot, fieldplot3d, gradplot, gradplot3d,
graphplot3d, implicitplot, implicitplot3d, inequal, interactive, interactiveparams,
intersectplot, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot,
logplot, matrixplot, multiple, odeplot, pareto, plotcompare, pointplot, pointplot3d,
polarplot, polygonplot, polygonplot3d, polyhedra_supported, polyhedraplot,
rootlocus, semilogplot, setcolors, setoptions, setoptions3d, spacecurve,
sparsematrixplot, surfdata, textplot, textplot3d, tubeplot]

▼ A) Les couleurs

*Tableau (C) des 6 couleurs utilisées dans le cube (version officielle):

Ce tableau sert essentiellement à obtenir les couleurs de chaque face pour les options des plots, la manipulation pour les mouvements du Rubik's Cube se faisant alors au niveau des indices, on a donc Chiffre⇒Couleur(en Maple)

```
C[1] := red;
C[2] := "DarkGreen";
C[3] := "OrangeRed";
C[4] := blue;
C[5] := yellow;
C[6] := white;
```

red

"DarkGreen"
 "OrangeRed"
blue
yellow
white

(1.2.1.1)

*Attributions des couleurs à chaque indice de C (initiales anglaises pour éviter les confusions bleu/blanc):

Ces notations permettent exclusivement de faciliter l'entrée des couleurs de chaque face d'un Rubik's Cube donné, en pratique on s'en sert de la manière suivante: Couleur(en "français")⇒ Chiffre⇒ Indice⇒ Couleur(en Maple)

r := 1;
g := 2;
o := 3;
b := 4;
y := 5;
w := 6;

1
2
3
4
5
6

(1.2.2.1)

B) Le Rubik's Cube

Le Rubik's Cube est un tableau composé par chacun de ses 8 petits cubes.

Les petits cubes sont eux même des tableaux composé de leurs coordonnées et des couleurs de leurs 3 faces.

Les coordonnées sont eux aussi des tableaux donnée par 3 chiffres (d'après les notations adopté). Les couleurs sont finalement toujours des tableaux donnée par les 3 couleurs des faces du petit cube concerné.

*Definition d'un entier max (pour la fréquence max de BougerRubix plus loin):

Nécessaire car $+\infty$ n'est pas un entier...

top := 100000;

100000

(1.3.1.1)

*Définitions des 8 cubes d'un Rubik's Cube fini:

Le numéro de chaque cube n'a aucune importance, ils sont pris dans l'ordre de la notation choisie.

- CUBE.FINI[1] :=* $\left[\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} C[w] \\ C[r] \\ C[b] \end{bmatrix} \right];$
- $\left[\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} \text{white} \\ \text{red} \\ \text{blue} \end{bmatrix} \right]$
- CUBE.FINI[2] :=* $\left[\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} C[g] \\ C[r] \\ C[w] \end{bmatrix} \right];$
- $\left[\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \text{"DarkGreen"} \\ \text{red} \\ \text{white} \end{bmatrix} \right]$ (1.3.2.2)
- CUBE.FINI[3] :=* $\left[\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} C[b] \\ C[r] \\ C[y] \end{bmatrix} \right];$
- $\left[\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} \text{blue} \\ \text{red} \\ \text{yellow} \end{bmatrix} \right]$ (1.3.2.3)
- CUBE.FINI[4] :=* $\left[\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} C[y] \\ C[r] \\ C[g] \end{bmatrix} \right];$
- $\left[\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} \text{yellow} \\ \text{red} \\ \text{"DarkGreen"} \end{bmatrix} \right]$ (1.3.2.4)
- CUBE.FINI[5] :=* $\left[\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} C[b] \\ C[o] \\ C[w] \end{bmatrix} \right];$

$$\left[\begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right], \left[\begin{array}{c} blue \\ "OrangeRed" \\ white \end{array} \right] \quad (1.3.2.5)$$

$$CUBE.FINI[6] := \left[\begin{array}{c} 1 \\ 0 \\ 1 \end{array} \right], \left[\begin{array}{c} C[w] \\ C[o] \\ C[g] \end{array} \right];$$

$$\left[\begin{array}{c} 1 \\ 0 \\ 1 \end{array} \right], \left[\begin{array}{c} white \\ "OrangeRed" \\ "DarkGreen" \end{array} \right] \quad (1.3.2.6)$$

$$CUBE.FINI[7] := \left[\begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right], \left[\begin{array}{c} C[y] \\ C[o] \\ C[b] \end{array} \right];$$

$$\left[\begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right], \left[\begin{array}{c} yellow \\ "OrangeRed" \\ blue \end{array} \right] \quad (1.3.2.7)$$

$$CUBE.FINI[8] := \left[\begin{array}{c} 1 \\ 1 \\ 1 \end{array} \right], \left[\begin{array}{c} C[g] \\ C[o] \\ C[y] \end{array} \right];$$

$$\left[\begin{array}{c} 1 \\ 1 \\ 1 \end{array} \right], \left[\begin{array}{c} "DarkGreen" \\ "OrangeRed" \\ yellow \end{array} \right] \quad (1.3.2.8)$$

*Notations du premier tableau d'un cube (les coordonnées) et du second (les couleurs):

coor := 1;
coul := 2;

$$\begin{matrix} 1 \\ 2 \end{matrix} \quad (1.3.3.1)$$

*Définitions du Rubik's Cube fini:

RUBIX.FINI := [*CUBE.FINI*[1], *CUBE.FINI*[2], *CUBE.FINI*[3], *CUBE.FINI*[4], *CUBE.FINI*[5], *CUBE.FINI*[6], *CUBE.FINI*[7], *CUBE.FINI*[8]];

$$\left[\begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right], \left[\begin{array}{c} white \\ red \\ blue \end{array} \right], \left[\begin{array}{c} 1 \\ 0 \\ 0 \end{array} \right], \left[\begin{array}{c} "DarkGreen" \\ red \\ white \end{array} \right], \left[\begin{array}{c} 0 \\ 1 \\ 1 \end{array} \right], \left[\begin{array}{c} blue \\ red \\ yellow \end{array} \right], \left[\begin{array}{c} 1 \\ 1 \\ 0 \end{array} \right], \quad (1.3.4.1)$$

$$\begin{aligned}
 & \left[\begin{array}{c} yellow \\ red \\ "DarkGreen" \end{array} \right], \left[\begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right], \left[\begin{array}{c} blue \\ "OrangeRed" \\ white \end{array} \right], \left[\begin{array}{c} 1 \\ 0 \\ 1 \end{array} \right], \left[\begin{array}{c} white \\ "OrangeRed" \\ "DarkGreen" \end{array} \right], \left[\begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right], \\
 & \left[\begin{array}{c} yellow \\ "OrangeRed" \\ blue \end{array} \right], \left[\begin{array}{c} 1 \\ 1 \\ 1 \end{array} \right], \left[\begin{array}{c} "DarkGreen" \\ "OrangeRed" \\ yellow \end{array} \right]
 \end{aligned}$$

Compte tenu de l'utilisation ultra fréquente de ce dernier dans les arguments des autres fonctions, on prendra la notation *rubix* := *RUBIX.FINI*;

$$\begin{aligned}
 & \left[\begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right], \left[\begin{array}{c} white \\ red \\ blue \end{array} \right], \left[\begin{array}{c} 1 \\ 0 \\ 0 \end{array} \right], \left[\begin{array}{c} "DarkGreen" \\ red \\ white \end{array} \right], \left[\begin{array}{c} 0 \\ 1 \\ 1 \end{array} \right], \left[\begin{array}{c} blue \\ red \\ yellow \end{array} \right], \left[\begin{array}{c} 1 \\ 1 \\ 0 \end{array} \right], \quad (1.3.4.2) \\
 & \left[\begin{array}{c} yellow \\ red \\ "DarkGreen" \end{array} \right], \left[\begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right], \left[\begin{array}{c} blue \\ "OrangeRed" \\ white \end{array} \right], \left[\begin{array}{c} 1 \\ 0 \\ 1 \end{array} \right], \left[\begin{array}{c} white \\ "OrangeRed" \\ "DarkGreen" \end{array} \right], \left[\begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right], \\
 & \left[\begin{array}{c} yellow \\ "OrangeRed" \\ blue \end{array} \right], \left[\begin{array}{c} 1 \\ 1 \\ 1 \end{array} \right], \left[\begin{array}{c} "DarkGreen" \\ "OrangeRed" \\ yellow \end{array} \right]
 \end{aligned}$$

*Fonction de correspondance n° Petit Cube \Rightarrow Coordonnée:

C'est une fonction qui donne les coordonnées correspondantes à un petit cube désigné par son numéro dans la notation choisie.

```

NumeroVersCoordonnees :=proc (num,$)
  local x, y, z;
  #La première coordonnée se récupère très facilement
  x := 0num mod 2;
  #Les deux autres nécessite une boucle conditionnelle
  if num = 1 then y := 0; z := 1;
  elif num = 2 then y := 0; z := 0;
  elif num = 3 then y := 1; z := 1;
  elif num = 4 then y := 1; z := 0;
  elif num = 5 then y := 0; z := 0;
  elif num = 6 then y := 0; z := 1;
  elif num = 7 then y := 1; z := 0;
  elif num = 8 then y := 1; z := 1;
  fi;
  #On renvoie finnalement le résultat
  [x, y, z];
end proc;

proc(num, $) (1.3.5.1)

```

```

local x, y, z;
x := 0^(mod(num, 2));
if num = 1 then
    y := 0; z := 1
elif num = 2 then
    y := 0; z := 0
elif num = 3 then
    y := 1; z := 1
elif num = 4 then
    y := 1; z := 0
elif num = 5 then
    y := 0; z := 0
elif num = 6 then
    y := 0; z := 1
elif num = 7 then
    y := 1; z := 0
elif num = 8 then
    y := 1; z := 1
end if;
[x, y, z]
end proc

```

*Fonction de correspondance Coordonnée \Rightarrow n° Petit Cube:

C'est une fonction qui donne le numéro d'un petit cube dans la notation choisie en fonction de ses coordonnées .

CoordonneesVersNumero :=proc (x, y, z,\$)

#Une boucle conditionnelle est la façon la plus claire pour avoir ce qu'on veut

```

if (x, y, z) = (0, 0, 1) then 1
elif (x, y, z) = (1, 0, 0) then 2
elif (x, y, z) = (0, 1, 1) then 3
elif (x, y, z) = (1, 1, 0) then 4
elif (x, y, z) = (0, 0, 0) then 5
elif (x, y, z) = (1, 0, 1) then 6
elif (x, y, z) = (0, 1, 0) then 7
elif (x, y, z) = (1, 1, 1) then 8
fi

```

end proc;

proc(x, y, z, \$)

(1.3.6.1)

```

if (x, y, z) = (0, 0, 1) then
    1
elif (x, y, z) = (1, 0, 0) then
    2
elif (x, y, z) = (0, 1, 1) then

```

```

    3
elif (x, y, z) = (1, 1, 0) then
    4
elif (x, y, z) = (0, 0, 0) then
    5
elif (x, y, z) = (1, 0, 1) then
    6
elif (x, y, z) = (0, 1, 0) then
    7
elif (x, y, z) = (1, 1, 1) then
    8
end if
end proc

```

***Fonction de correspondance Couleurs \Rightarrow n° Petit Cube:**

C'est une fonction qui donne le numéro d'un petit cube dans la notation choisit en fonction de ses coordonnées .

CouleursVersNumero :=proc (c1, c2, c3,\$)

#Une boucle conditionnelle est la façon la plus claire pour avoir ce qu'on veut

```

        if {c1, c2, c3} = {C[w], C[r], C[b]} then 1
        elif {c1, c2, c3} = {C[g], C[r], C[w]} then 2
        elif {c1, c2, c3} = {C[b], C[r], C[y]} then 3
        elif {c1, c2, c3} = {C[y], C[r], C[g]} then 4
        elif {c1, c2, c3} = {C[b], C[o], C[w]} then 5
        elif {c1, c2, c3} = {C[w], C[o], C[g]} then 6
        elif {c1, c2, c3} = {C[y], C[o], C[b]} then 7
        elif {c1, c2, c3} = {C[g], C[o], C[y]} then 8
        fi

```

end proc;

proc(c1, c2, c3, \$)

(1.3.7.1)

```

        if {c1, c2, c3} = {C[b], C[r], C[w]} then
            1
        elif {c1, c2, c3} = {C[g], C[r], C[w]} then
            2
        elif {c1, c2, c3} = {C[b], C[r], C[y]} then
            3
        elif {c1, c2, c3} = {C[g], C[r], C[y]} then
            4
        elif {c1, c2, c3} = {C[b], C[o], C[w]} then
            5
        elif {c1, c2, c3} = {C[g], C[o], C[w]} then
            6
        elif {c1, c2, c3} = {C[b], C[o], C[y]} then

```

```

    7
  elif {c1, c2, c3} = {C[g], C[o], C[y]} then
    8
  end if
end proc

```

***Fonction de rafraîchissement du Rubik's cube, qui le place dans sa notation standard:**
 Cette fonction est strictement nécessaire pour afficher correctement le cube en 3D

```

Rafraichir :=proc(Rubix1,$)
  local k, Rubix2;
#On replace les positions de chaque petit cube
  for k to 8 do
    Rubix2[CoordonneesVersNumero(Rubix1[k][coor][1], Rubix1[k][coor][2],
    Rubix1[k][coor][3])] := Rubix1[k]
  end do;
#On renvoie le nouveau Rubik's Cube
  Rubix2;
  end proc;
proc(Rubix1, $)
  local k, Rubix2;
  for k to 8 do
    Rubix2[CoordonneesVersNumero(Rubix1[k][coor][1], Rubix1[k][coor][2
    ], Rubix1[k][coor][3])] := Rubix1[k]
  end do;
  Rubix2
end proc

```

(1.3.8.1)

C) Les etats du Rubik's Cube

a) La finitude

***Fonction testant si le Rubik's Cube considéré est fini:**

On vérifie simplement que les 4 petites faces de chacune des 6 grandes faces sont de la même couleur.

```

EstFini :=proc(RubixTeste,$)
  local k, x, y, z, Rubix;
#On renumérote les petits cube du Rubik's Cube pour le placer dans la position de
références
  for k to 8 do
    #On récupère les coordonées de chaque petit cube
    x := RubixTeste[k][coor][1];
    y := RubixTeste[k][coot][2];
    z := RubixTeste[k][coor][3];
    #Une formule barbare renumérote chacun des petits cubes du Rubik's Cube selon

```

leurs coordonnées

#pour les faire correspondre avec la notation choisit.

```
Rubix[0x·(0y·(5 - 4 ·z) + y·(7 - 4 ·z)) + x·(0y·(2 + 4 ·z) + y·(4 + 4 ·z))] := RubixTeste[k];  
end do;
```

#Un gros booléens donne alors directement la réponse

```
Rubix[1][coul][2] = Rubix[2][coul][2] = Rubix[3][coul][2] = Rubix[4][coul][2]
```

and Rubix[5][coul][2] = Rubix[6][coul][2] = Rubix[7][coul][2] = Rubix[8][coul][2]

and Rubix[2][coul][1] = Rubix[4][coul][3] = Rubix[6][coul][3] = Rubix[8][coul][1]

and Rubix[1][coul][3] = Rubix[3][coul][1] = Rubix[5][coul][1] = Rubix[7][coul][3]

and Rubix[1][coul][1] = Rubix[2][coul][3] = Rubix[5][coul][3]

= Rubix[6][coul][1]

and Rubix[3][coul][3] = Rubix[4][coul][1] = Rubix[7][coul][1] = Rubix[8][coul][3]

end proc;

proc(RubixTeste, \$)

local k, x, y, z, Rubix;

for k to 8 do

x := RubixTeste[k][coor][1];

y := RubixTeste[k][coot][2];

z := RubixTeste[k][coor][3];

Rubix[0^x·(0^y·(5 - 4 ·z) + y·(7 - 4 ·z)) + x·(0^y·(2 + 4 ·z) + y·(4 + 4 ·z))] := RubixTeste[k]

end do;

Rubix[1][coul][2] = Rubix[2][coul][2] and Rubix[2][coul][2] = Rubix[3]

[coul][2] and Rubix[3][coul][2] = Rubix[4][coul][2] and Rubix[5]

[coul][2] = Rubix[6][coul][2] and Rubix[6][coul][2] = Rubix[7][coul]

[2] and Rubix[7][coul][2] = Rubix[8][coul][2] and Rubix[2][coul][1]

= Rubix[4][coul][3] and Rubix[4][coul][3] = Rubix[6][coul][3] and

Rubix[6][coul][3] = Rubix[8][coul][1] and Rubix[1][coul][3] = Rubix[3]

[coul][1] and Rubix[3][coul][1] = Rubix[5][coul][1] and Rubix[5]

[coul][1] = Rubix[7][coul][3] and Rubix[1][coul][1] = Rubix[2][coul]

[3] and Rubix[2][coul][3] = Rubix[5][coul][3] and Rubix[5][coul][3]

= Rubix[6][coul][1] and Rubix[3][coul][3] = Rubix[4][coul][1] and

Rubix[4][coul][1] = Rubix[7][coul][1] and Rubix[7][coul][1] = Rubix[8]

[coul][3]

end proc

(1.4.1.1.1)

B) La faisabilité

Comme il existe une suite de mouvement permettant de interchanger 2 petits cubes quelconques sans changer leur orientation, on ne considère que les orientations des petits cubes.

Il est alors nécessaire et suffisant d'avoir:

- $\forall \text{Cube1}, \text{Cube2} \in \text{Rubix}, \{\text{Cube1}[coul]\} \neq \{\text{Cube2}[coul]\}$
- $\forall \text{Cube} \in \text{Rubix}, \exists \text{CUBE.FINI} \in \text{RUBIX.FINI} \mid \exists n \in \mathbb{N} \mid \text{Cube}[coul] =$

$c^n(CUBE.FINI[coul])$: où $c(E)$ est un cycle de l'ensemble E

• $\sum n \equiv 0 [3]$

*Fonction qui vérifie l'axiome 1:

Chaque petits cubes d'un Rubik's Cube faisable doivent nécessairement avoir leur triplets de couleur différent entre eux.

Axiome1 :=proc(Rubix,\$)

#On vérifie qu'il y ait bien 8 triplets différents de couleurs

evalb(nops({

*{Rubix[1][coul][1], Rubix[1][coul][2], Rubix[1][coul][3]},
{Rubix[2][coul][1], Rubix[2][coul][2], Rubix[2][coul][3]},
{Rubix[3][coul][1], Rubix[3][coul][2], Rubix[3][coul][3]},
{Rubix[4][coul][1], Rubix[4][coul][2], Rubix[4][coul][3]},
{Rubix[5][coul][1], Rubix[5][coul][2], Rubix[5][coul][3]},
{Rubix[6][coul][1], Rubix[6][coul][2], Rubix[6][coul][3]},
{Rubix[7][coul][1], Rubix[7][coul][2], Rubix[7][coul][3]},
{Rubix[8][coul][1], Rubix[8][coul][2], Rubix[8][coul][3]},*

}) = 8);

end proc;

proc(Rubix, \$)

evalb(nops({ {Rubix[1][coul][1], Rubix[1][coul][2], Rubix[1][coul][3]}, {Rubix[2][coul][1], Rubix[2][coul][2], Rubix[2][coul][3]}, {Rubix[3][coul][1], Rubix[3][coul][2], Rubix[3][coul][3]}, {Rubix[4][coul][1], Rubix[4][coul][2], Rubix[4][coul][3]}, {Rubix[5][coul][1], Rubix[5][coul][2], Rubix[5][coul][3]}, {Rubix[6][coul][1], Rubix[6][coul][2], Rubix[6][coul][3]}, {Rubix[7][coul][1], Rubix[7][coul][2], Rubix[7][coul][3]}, {Rubix[8][coul][1], Rubix[8][coul][2], Rubix[8][coul][3]} }) = 8)

end proc

(1.4.2.1.1)

*Fonction qui vérifie si un triplet de couleur est cycle du rubix fini:

Elle renvoie ∞ si le triplet n'appartient à aucun cycle.

EstCycle :=proc(Couleur,\$)

local k, n;

n := infinity;

#On vérifie que le vecteur de couleur donnée est cycle du rubix fini

for k to 8 do

if Couleur[1] = (RUBIX.FINI)[k][coul][1] and Couleur[2] = (RUBIX.FINI)[k][coul][2] and Couleur[3] = (RUBIX.FINI)[k][coul][3] then n := 0; break;

elif Couleur[1] = (RUBIX.FINI)[k][coul][2] and Couleur[2] = (RUBIX.FINI)[k][coul][3] and Couleur[3] = (RUBIX.FINI)[k][coul][1] then n := 1; break;

elif Couleur[1] = (RUBIX.FINI)[k][coul][3] and Couleur[2] = (RUBIX.FINI)[k][coul][1] and Couleur[3] = (RUBIX.FINI)[k][coul][2] then n := 2; break;

end if; end do;

```

#On renvoie l'indice du cycle
    n;
    end proc;
proc(Couleur, $)                                (1.4.2.2.1)
    local k, n;
    n := ∞;
    for k to 8 do
        if Couleur[1] = Typesetting:-delayDotProduct(RUBIX, FINI)[k][coul]
        [1] and Couleur[2] = Typesetting:-delayDotProduct(RUBIX, FINI)[k]
        [coul][2] and Couleur[3] = Typesetting:-delayDotProduct(RUBIX,
        FINI)[k][coul][3] then
            n := 0; break
        elif Couleur[1] = Typesetting:-delayDotProduct(RUBIX, FINI)[k][coul]
        ][2] and Couleur[2] = Typesetting:-delayDotProduct(RUBIX, FINI)[k
        ][coul][3] and Couleur[3] = Typesetting:-delayDotProduct(RUBIX,
        FINI)[k][coul][1] then
            n := 1; break
        elif Couleur[1] = Typesetting:-delayDotProduct(RUBIX, FINI)[k][coul]
        ][3] and Couleur[2] = Typesetting:-delayDotProduct(RUBIX, FINI)[k
        ][coul][1] and Couleur[3] = Typesetting:-delayDotProduct(RUBIX,
        FINI)[k][coul][2] then
            n := 2; break
        end if
    end do;
    n
end proc

```

*Fonction qui vérifie l'axiome 2 et 3:

L'orientation de chaque petits cubes d'un Rubik's Cube faisable doit nécessairement être un cycle d'un des petits petits cube du cube fini.

```

Axiome2et3 :=proc(Rubix,$)
    local n, k;
    n := 0;
#On récupère l'indice des cycle de chaque cube
    for k to 8 do
        n := n + EstCycle(Rubix[k][coul]);
    end do;
#On vérifie d'un coup les deux axiomes
    (not n = ∞) and (n mod 3 = 0);
    end proc;
proc(Rubix, $)                                    (1.4.2.3.1)
    local n, k;
    n := 0;

```

```

for k to 8 do n := n + EstCycle(Rubix[k][coul]) end do;
not n =  $\infty$  and mod(n, 3) = 0
end proc

```

*Fonction qui vérifie si un cube est faisable:

```

EstFaisable :=proc(Rubix,$)
    (Axiome1(Rubix) and Axiome2et3(Rubix))
    end proc;
proc(Rubix, $) Axiome1(Rubix) and Axiome2et3(Rubix) end proc      (1.4.2.4.1)

```

II)Les 12 mouvements possibles

A)Selon la face gauche

*Fonction qui change les données d'un petit cube ayant subi une rotation du bloc L(left)
(dans le sens direct):

```

RotationCubeL := proc(Cube,$)
    local x, y, z, C, a, b, c, d, e, f;
    #On récupère les coordonnées du petit cube donné
    x := Cube[coor][1];
    y := Cube[coor][2];
    z := Cube[coor][3];
    #On récupère l'orientation des couleurs du petit cube donné
    C[0] := Cube[coul][1];
    C[1] := Cube[coul][2];
    C[2] := Cube[coul][3];
    #On détermine les nouvelles coordonnées du petit cube
    a := x;
    b := x·y + (1 - x) · (0z+y+1 mod 2 + y) mod 2;
    c := x·z + (1 - x) · (0(z+y) mod 2 + z) mod 2;

```

#On détermine les nouvelles orientations des couleurs du petit cube

```

    d := (x·0 + (1 - x) · (y + z mod 2) · 2 + (1 - x) · 0y+z mod 2 · 1) mod 3;
    e := (x·1 + (1 - x) · (y + z mod 2) · 0 + (1 - x) · 0y+z mod 2 · 2) mod 3;
    f := (x·2 + (1 - x) · (y + z mod 2) · 1 + (1 - x) · 0y+z mod 2 · 0) mod 3;

```

#On renvoie les nouvelles données du cube

```

    [[ a ], [[ C[d] ]],
    [[ b ], [[ C[e] ]],
    [[ c ], [[ C[f] ]]];
end proc;

```

```

proc(Cube, $)                                (2.1.1.1)
    local x, y, z, C, a, b, c, d, e, f;
    x := Cube[coor][1];
    y := Cube[coor][2];

```

```

z := Cube[coor][3];
C[0] := Cube[coul][1];
C[1] := Cube[coul][2];
C[2] := Cube[coul][3];
a := x;
b := mod(x * y + (1 - x) * (0^(mod(z + y + 1, 2)) + y), 2);
c := mod(x * z + (1 - x) * (0^(mod(z + y, 2)) + z), 2);
d := mod(2 * (1 - x) * (mod(z + y, 2)) + (1 - x) * 0^(mod(z + y, 2)), 3);
e := mod(x + 2 * (1 - x) * 0^(mod(z + y, 2)), 3);
f := mod(2 * x + (1 - x) * (mod(z + y, 2)), 3);
[Vector(3, {1 = a, 2 = b, 3 = c}), Vector(3, {1 = C[d], 2 = C[e], 3 = C[f]})]
end proc

```

***Fonction qui applique celle qui précède à tous les petit cube du rubik's cube et renvoie le nouveau:**

```

RotationBlocL := proc(Rubix1,$)
local k, Rubix2;
#On change tous les petits cube du Rubik's considéré
for k to 8 do
Rubix2[k] := RotationCubeL(Rubix1[k]);
end do;
Rubix2;
end proc;

proc(Rubix1, $) (2.1.2.1)
local k, Rubix2;
for k to 8 do Rubix2[k] := RotationCubeL(Rubix1[k]) end do; Rubix2
end proc

```

***Fonction qui fait cette fois ci l'opération dans le sens indirect:**

```

RotationBlocL" := proc(Rubix1,$)
local k, Rubix2;
#On change tous les petits cube du Rubik's considéré
for k to 8 do
Rubix2[k]
:= RotationCubeL(RotationCubeL(RotationCubeL(Rubix1[k])));
end do;
Rubix2;
end proc;

proc(Rubix1, $) (2.1.3.1)
local k, Rubix2;
for k to 8 do
Rubix2[k] := RotationCubeL(RotationCubeL(RotationCubeL(Rubix1[k])) )
end do;
Rubix2
end proc

```

*Notation:

Compte tenu de l'utilisation plus que récurrente de ces deux fonctions pour la saisie des mouvements, on les nomme également, jusqu'à la fin du projet, de manière moins explicite mais plus simple par:

$$\begin{aligned} L &:= \text{RotationBlocL}; \\ L'' &:= \text{RotationBlocL}''; \\ &\quad \text{RotationBlocL} \\ &\quad \text{RotationBlocL}'' \end{aligned} \tag{2.1.4.1}$$

B) Selon la face droite

*Fonction qui change les données d'un petit cube ayant subi une rotation du bloc R (right) (dans le sens direct):

```
RotationCubeR := proc(Cube,$)
    local x, y, z, C, a, b, c, d, e, f;
    #On récupère les coordonnées du petit cube donné
    x := Cube[coor][1];
    y := Cube[coor][2];
    z := Cube[coor][3];
    #On récupère l'orientation des couleurs du petit cube donné
    C[0] := Cube[coul][1];
    C[1] := Cube[coul][2];
    C[2] := Cube[coul][3];
    #On détermine les nouvelles coordonnées du petit cube
    a := x;
    b := (1 - x) · y + x · (0z+y+1 mod 2 + y) mod 2;
    c := (1 - x) · z + x · (0(z+y) mod 2 + z) mod 2;
    #On détermine les nouvelles orientations des couleurs du petit cube
    d := ((1 - x) · 0 + x · (y + z mod 2) · 2 + x · 0y+z mod 2 · 1) mod 3;
    e := ((1 - x) · 1 + x · (y + z mod 2) · 0 + x · 0y+z mod 2 · 2) mod 3;
    f := ((1 - x) · 2 + x · (y + z mod 2) · 1 + x · 0y+z mod 2 · 0) mod 3;
```

#On renvoie les nouvelles données du cube

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}, \begin{bmatrix} C[d] \\ C[e] \\ C[f] \end{bmatrix};$$

end proc;

```
proc(Cube, $)
    local x, y, z, C, a, b, c, d, e, f;
    x := Cube[coor][1];
    y := Cube[coor][2];
    z := Cube[coor][3];
    C[0] := Cube[coul][1];
```

```

C[1]:=Cube[coul][2];
C[2]:=Cube[coul][3];
a:=x;
b:=mod((1-x)*y+x*(0^(mod(z+y+1,2))+y),2);
c:=mod((1-x)*z+x*(0^(mod(z+y,2))+z),2);
d:=mod(2*x*(mod(z+y,2))+x*0^(mod(z+y,2)),3);
e:=mod(1-x+2*x*0^(mod(z+y,2)),3);
f:=mod(2-2*x+x*(mod(z+y,2)),3);
[Vector(3, {1=a, 2=b, 3=c}), Vector(3, {1=C[d], 2=C[e], 3=C[f]})]
end proc

```

***Fonction qui applique celle qui précède à tous les petit cube du rubik's cube et renvoie le nouveau:**

```

RotationBlocR :=proc(Rubix1,$)
  local k, Rubix2;
  #On change tous les petits cube du Rubik's considéré
  for k to 8 do
    Rubix2[k]:=RotationCubeR(Rubix1[k]);
  end do;
  Rubix2;
  end proc;

proc(Rubix1, $) (2.2.2.1)
  local k, Rubix2;
  for k to 8 do Rubix2[k]:=RotationCubeR(Rubix1[k]) end do; Rubix2
end proc

```

***Fonction qui fait cette fois ci l'opération dans le sens indirect:**

```

RotationBlocR" :=proc(Rubix1,$)
  local k, Rubix2;
  #On change tous les petits cube du Rubik's considéré
  for k to 8 do
    Rubix2[k]
    :=RotationCubeR(RotationCubeR(RotationCubeR(Rubix1[k])));
  end do;
  Rubix2;
  end proc;

proc(Rubix1, $) (2.2.3.1)
  local k, Rubix2;
  for k to 8 do
    Rubix2[k]:=RotationCubeR(RotationCubeR(RotationCubeR(Rubix1[k])) )
  end do;
  Rubix2
end proc

```

***Notation:**

Compte tenu de l'utilisation plus que récurrente de ces deux fonctions pour la saisie des mouvements, on les nomme également, jusqu'à la fin du projet, de manière moins explicite mais plus simple par:

```
R := RotationBlocR;
R' := RotationBlocR';
RotationBlocR
RotationBlocR'
```

(2.2.4.1)

C) Selon la face basse

***Fonction qui change les données d'un petit cube ayant subi une rotation du bloc B (basse) (dans le sens direct):**

```
RotationCubeB := proc(Cube,$)
local x, y, z, C, a, b , c, d, e,f;
#On récupère les coordonnées du petit cube donné
x := Cube[coor][1];
y := Cube[coor][2];
z := Cube[coor][3];
#On récupère l'orientation des couleurs du petit cube donné
C[0] := Cube[coul][1];
C[1] := Cube[coul][2];
C[2] := Cube[coul][3];
#On détermine les nouvelles coordonnées du petit cube
a := y·x + (1 - y) · (x + 0z) mod 2;
b := y;
c := y·z + (1 - y) · (x + z + 0x) mod 2;
#On détermine les nouvelles orientations des couleurs du petit cube
d := (y·0 + (1 - y) · z·1 + (1 - y) · (1 - z) · 2) mod 3;
e := (y·1 + (1 - y) · z·2 + (1 - y) · (1 - z) · 0) mod 3;
f := (y·2 + (1 - y) · z·0 + (1 - y) · (1 - z) · 1) mod 3;
#On renvoie les nouvelles données du cube
[ [ a ] [ C[d] ]
      [ b ], [ C[e] ]
      [ c ] [ C[f] ] ];
end proc;
```

```
proc(Cube, $)
local x, y, z, C, a, b, c, d, e, f,
x := Cube[coor][1];
y := Cube[coor][2];
z := Cube[coor][3];
C[0] := Cube[coul][1];
C[1] := Cube[coul][2];
C[2] := Cube[coul][3];
a := mod(x * y + (1 - y) * (x + 0z), 2);
b := y;
```

(2.3.1.1)

```

c := mod(y * z + (1 - y) * (x + z + 0^x), 2);
d := mod((1 - y) * z + 2 * (1 - y) * (1 - z), 3);
e := mod(y + 2 * (1 - y) * z, 3);
f := mod(2 * y + (1 - y) * (1 - z), 3);
[Vector(3, {1 = a, 2 = b, 3 = c}), Vector(3, {1 = C[d], 2 = C[e], 3 = C[f]})]
end proc

```

***Fonction qui applique celle qui précède à tous les petit cube du rubik's cube et renvoie le nouveau:**

```

RotationBlocB :=proc(Rubix1,$)
    local k, Rubix2;
    #On change tous les petits cube du Rubik's considéré
    for k to 8 do
        Rubix2[k] := RotationCubeB(Rubix1[k]);
    end do;
    Rubix2;
end proc;

proc(Rubix1, $) (2.3.2.1)
    local k, Rubix2;
    for k to 8 do Rubix2[k] := RotationCubeB(Rubix1[k]) end do; Rubix2
end proc

```

***Fonction qui fait cette fois ci l'opération dans le sens indirect:**

```

RotationBlocB" :=proc(Rubix1,$)
    local k, Rubix2;
    #On change tous les petits cube du Rubik's considéré
    for k to 8 do
        Rubix2[k]
        := RotationCubeB(RotationCubeB(RotationCubeB(Rubix1[k])));
    end do;
    Rubix2;
end proc;

proc(Rubix1, $) (2.3.3.1)
    local k, Rubix2;
    for k to 8 do
        Rubix2[k] := RotationCubeB(RotationCubeB(RotationCubeB(Rubix1[k])))
    end do;
    Rubix2
end proc

```

***Notation:**

Compte tennue de l'utilisation plus que récurrente de ces deux fonctions pour la saisi des mouvement, on les nommes également, jusqu'à la fin du projet, de manière moins explicite mais plus simple par:

$B := \text{RotationBloc}B;$
 $B'' := \text{RotationBloc}B'',$
 $\text{RotationBloc}B$
 $\text{RotationBloc}B''$
(2.3.4.1)

D) Selon la face haut

*Fonction qui change les données d'un petit cube ayant subi une rotation du bloc H(haut)
(dans le sens direct):

```

RotationCubeH := proc(Cube,$)
    local x, y, z, C, a, b, c, d, e, f;
    #On récupère les coordonnées du petit cube donné
    x := Cube[coor][1];
    y := Cube[coor][2];
    z := Cube[coor][3];
    #On récupère l'orientation des couleurs du petit cube donné
    C[0] := Cube[coul][1];
    C[1] := Cube[coul][2];
    C[2] := Cube[coul][3];
    #On détermine les nouvelles coordonnées du petit cube
    a := (1 - y) · x + y · (x + 0z) mod 2;
    b := y;
    c := (1 - y) · z + y · (x + z + 0x) mod 2;
    #On détermine les nouvelles orientations des couleurs du petit cube
    d := ((1 - y) · 0 + y · z · 2 + y · (1 - z) · 1) mod 3;
    e := ((1 - y) · 1 + y · z · 0 + y · (1 - z) · 2) mod 3;
    f := ((1 - y) · 2 + y · z · 1 + y · (1 - z) · 0) mod 3;
    #On renvoie les nouvelles données du cube
    [[ a, C[d], ], [ b, C[e], ], [ c, C[f], ]];
end proc;

```

*Fonction qui applique celle qui précède à tous les petit cube du rubik's cube et renvoie le nouveau:

```

RotationBlocH := proc(Rubix1,$)
    local k, Rubix2;
    #On change tous les petits cube du Rubik's considéré
    for k to 8 do
        Rubix2[k]
        := RotationCubeH(RotationCubeH(RotationCubeH(Rubix1[k])));
    end do;
    Rubix2;
end proc;

proc(Rubix1, $)
    local k, Rubix2;
(2.4.2.1)

```

```

for k to 8 do
    Rubix2[k] := RotationCubeH(RotationCubeH(RotationCubeH(Rubix1[k
    ])))
end do;
    Rubix2
end proc

```

***Fonction qui fait cette fois ci l'opération dans le sens indirect:**

RotationBlocH" :=proc(Rubix1,\$)
local k, Rubix2;

#On change tous les petits cube du Rubik's considéré

```

for k to 8 do
    Rubix2[k] := RotationCubeH(Rubix1[k]);
end do;
    Rubix2;
end proc;

```

proc(Rubix1, \$) (2.4.3.1)
local k, Rubix2;
***for** k **to** 8 **do** Rubix2[k] := RotationCubeH(Rubix1[k]) **end do;** Rubix2*
end proc

***Notation:**

Compte tenu de l'utilisation plus que récurrente de ces deux fonctions pour la saisie des mouvements, on les nomme également, jusqu'à la fin du projet, de manière moins explicite mais plus simple par:

H := RotationBlocH;
H" := RotationBlocH";
RotationBlocH
RotationBlocH" (2.4.4.1)

E) Selon la face postérieur

***Fonction qui change les données d'un petit cube ayant subi une rotation du bloc P (postérieur) (dans le sens direct):**

RotationCubeP := proc(Cube,\$)
local x, y, z, C, a, b, c;

#On récupère les coordonnées du petit cube donné

x := Cube[coor][1];
y := Cube[coor][2];
z := Cube[coor][3];

#On récupère l'orientation des couleurs du petit cube donné

C[0] := Cube[coul][1];
C[1] := Cube[coul][2];
C[2] := Cube[coul][3];

#On détermine les nouvelles coordonnées du petit cube

$$a := (x + z \bmod 2) \cdot x + (1 - (x + z \bmod 2)) \cdot (x + z + 0^{x+y+z \bmod 2} \bmod 2);$$

$$b := (x + z \bmod 2) \cdot y + (1 - (x + z \bmod 2)) \cdot (y + x + 1 + 0^{x+y+z \bmod 2} \bmod 2);$$

$$c := (x + z \bmod 2) \cdot z + (1 - (x + z \bmod 2)) \cdot (x + z + 0^{x+y+z \bmod 2} \bmod 2);$$

#On renvoie les nouvelles données du cube

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}, \begin{bmatrix} C[0] \\ C[1] \\ C[2] \end{bmatrix};$$

end proc;

proc(Cube, \$) (2.5.1.1)

local x, y, z, C, a, b, c;

 x := Cube[coor][1];

 y := Cube[coor][2];

 z := Cube[coor][3];

 C[0] := Cube[coul][1];

 C[1] := Cube[coul][2];

 C[2] := Cube[coul][3];

 a := (mod(x + z, 2)) * x + (1 - (mod(x + z, 2))) * (mod(x + z + 0^(mod(x + y + z, 2)), 2));

 b := (mod(x + z, 2)) * y + (1 - (mod(x + z, 2))) * (mod(y + x + 1 + 0^(mod(x + y + z, 2)), 2));

 c := (mod(x + z, 2)) * z + (1 - (mod(x + z, 2))) * (mod(x + z + 0^(mod(x + y + z, 2)), 2));

 [Vector(3, {1 = a, 2 = b, 3 = c}), Vector(3, {1 = C[0], 2 = C[1], 3 = C[2]})]

end proc

***Fonction qui applique celle qui précède à tous les petit cube du rubik's cube et renvoie le nouveau:**

RotationBlocP :=proc(Rubix1,\$)

local k, Rubix2;

#On change tous les petits cube du Rubik's considéré

for k **to** 8 **do**

 Rubix2[k]

 := RotationCubeP(RotationCubeP(RotationCubeP(Rubix1[k]))));

end do;

 Rubix2;

end proc;

proc(Rubix1, \$) (2.5.2.1)

local k, Rubix2;

for k **to** 8 **do**

```

    Rubix2[k] := RotationCubeP(RotationCubeP(RotationCubeP(Rubix1[k])) )
end do;
    Rubix2
end proc

```

***Fonction qui fait cette fois ci l'opération dans le sens indirect:**

```

 $\text{RotationBlocP}'' := \text{proc}(Rubix1, \$)$ 
    local k, Rubix2;
#On change tous les petits cube du Rubik's considéré
    for k to 8 do
        Rubix2[k] := RotationCubeP(Rubix1[k]);
    end do;
    Rubix2;
end proc;

proc(Rubix1, \$) (2.5.3.1)
    local k, Rubix2;
    for k to 8 do Rubix2[k] := RotationCubeP(Rubix1[k]) end do; Rubix2
end proc

```

***Notation:**

Compte tennue de l'utilisation plus que récurrente de ces deux fonctions pour la saisie des mouvement, on les nommes également, jusqu'à la fin du projet, de manière moins explicite mais plus simple par:

```

 $P := \text{RotationBlocP};$ 
 $P'' := \text{RotationBlocP}'';$ 
    RotationBlocP
    RotationBlocP'' (2.5.4.1)

```

F) Selon la face avant

***Fonction qui change les données d'un petit cube ayant subi une rotation du bloc A (avant) (dans le sens direct):**

```

 $\text{RotationCubeA} := \text{proc}(Cube, \$)$ 
    local x, y, z, C, a, b, c;
#On récupère les coordonnées du petit cube donné
    x := Cube[coor][1];
    y := Cube[coor][2];
    z := Cube[coor][3];
#On récupère l'orientation des couleurs du petit cube donné
    C[0] := Cube[coul][1];
    C[1] := Cube[coul][2];
    C[2] := Cube[coul][3];
#On détermine les nouvelles coordonnées du petit cube
    a := (1 - (x + z mod 2)) · x + (x + z mod 2) · (x + z + 0x+y+z mod 2)

```

```

mod 2);
 $b := (1 - (x + z \bmod 2)) \cdot y + (x + z \bmod 2) \cdot (x + y + 0^{x+y+z \bmod 2}$ 
mod 2);
 $c := (1 - (x + z \bmod 2)) \cdot z + (x + z \bmod 2) \cdot (x + z + 1 + 0^{x+y+z \bmod 2}$ 
mod 2);
#On renvoie les nouvelles données du cube

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}, \begin{bmatrix} C[0] \\ C[1] \\ C[2] \end{bmatrix};$$

end proc;
proc(Cube, $) (2.6.1.1)
  local x, y, z, C, a, b, c;
  x := Cube[coor][1];
  y := Cube[coor][2];
  z := Cube[coor][3];
  C[0] := Cube[coul][1];
  C[1] := Cube[coul][2];
  C[2] := Cube[coul][3];
  a := (1 - (mod(x + z, 2))) * x + (mod(x + z, 2)) * (mod(x + z + 0^(mod(x
    + y + z, 2)), 2));
  b := (1 - (mod(x + z, 2))) * y + (mod(x + z, 2)) * (mod(x + y + 0^(mod(x
    + y + z, 2)), 2));
  c := (1 - (mod(x + z, 2))) * z + (mod(x + z, 2)) * (mod(x + z + 1 + 0
    ^ (mod(x + y + z, 2)), 2));
  [Vector(3, {1 = a, 2 = b, 3 = c}), Vector(3, {1 = C[0], 2 = C[1], 3 = C[2]})]
end proc

```

***Fonction qui applique celle qui précède à tous les petit cube du rubik's cube et renvoie le nouveau:**

```

RotationBlocA :=proc(Rubix1,$)
  local k, Rubix2;
#On change tous les petits cube du Rubik's considéré
  for k to 8 do
    Rubix2[k] := RotationCubeA(Rubix1[k]);
  end do;
  Rubix2;
end proc;
proc(Rubix1, $) (2.6.2.1)
  local k, Rubix2;
  for k to 8 do Rubix2[k] := RotationCubeA(Rubix1[k]) end do; Rubix2
end proc

```

***Fonction qui fait cette fois ci l'opération dans le sens indirect:**

```

RotationBlocA" :=proc(Rubix1,$)
    local k, Rubix2;
#On change tous les petits cube du Rubik's considéré
    for k to 8 do
        Rubix2[k]
    := RotationCubeA(RotationCubeA(RotationCubeA(RotationCubeA(Rubix1[k]))));
    end do;
    Rubix2;
end proc;

proc(Rubix1, $) (2.6.3.1)
    local k, Rubix2;
    for k to 8 do
        Rubix2[k] := RotationCubeA(RotationCubeA(RotationCubeA(RotationCubeA(Rubix1[k]))))
    end do;
    Rubix2
end proc

```

*Notation:

Compte tenu de l'utilisation plus que récurrente de ces deux fonctions pour la saisie des mouvements, on les nomme également, jusqu'à la fin du projet, de manière moins explicite mais plus simple par:

```

A := RotationBlocA;
A" := RotationBlocA";
RotationBlocA
RotationBlocA" (2.6.4.1)

```

III) Les outils

A) Le patron

*Fonction qui dessine un petit carré:

```

DessinePetitCube :=proc(Cube,$)
    local S, x, y, t, C;
#On récupère les coordonnées
    t := 01 - Cube[coor][3]. (1 + 2·Cube[coor][2]) + 0Cube[coor][3]. (2 + 2
    ·Cube[coor][2]);
    x := 4 · (02 - Cube[coor][1] - Cube[coor][3] - 0Cube[coor][1] + Cube[coor][3]);
    y := 0;
    C := Cube[coul];
#On dessine les 3 petits carrés
    if t ≤ 2 then
        S := polygonplot([ [x,y], [x + (-1)t,y], [x + (-1)t,y - 1], [x,y - 1], [x,y] ], color

```

```

= C[2]);  

S := S, polygonplot( [[x - 0^{t-1}, y - 0^{2-t}], [x - 0^{t-1} + (-1)^t, y - 0^{2-t}], [x - 0^{t-1} + (-1)^t, y - 0^{2-t} - 1], [x - 0^{t-1}, y - 0^{2-t} - 1], [x - 0^{t-1}, y - 0^{2-t}]], color = C[3]);  

S := S, polygonplot( [[x + 0^{2-t}, y - 0^{t-1}], [x + 0^{2-t} + (-1)^t, y - 0^{t-1}], [x + 0^{2-t} + (-1)^t, y - 0^{t-1} - 1], [x + 0^{2-t}, y - 0^{t-1} - 1], [x + 0^{2-t}, y - 0^{t-1}]], color = C[1]);  

else  

S := polygonplot( [[x, y], [x + (-1)^t, y], [x + (-1)^t, y + 1], [x, y + 1], [x, y]], color  

= C[2]);  

S := S, polygonplot( [[x + 0^{4-t}, y + 0^{t-3}], [x + 0^{4-t} + (-1)^t, y + 0^{t-3}], [x + 0^{4-t} + (-1)^t, y + 0^{t-3} + 1], [x + 0^{4-t}, y + 0^{t-3} + 1], [x + 0^{4-t}, y + 0^{t-3}]], color = C[3]);  

S := S, polygonplot( [[x - 0^{t-3}, y + 0^{4-t}], [x - 0^{t-3} + (-1)^t, y + 0^{4-t}], [x - 0^{t-3} + (-1)^t, y + 0^{4-t} + 1], [x - 0^{t-3}, y + 0^{4-t} + 1], [x - 0^{t-3}, y + 0^{4-t}]], color = C[1]);  

end if;  

S;  

end proc;  

proc(Cube, $) (3.1.1.1)  

local S, x, y, t, C;  

t := 0^(1 - Cube[coor][3]) * (1 + 2 * Cube[coor][2]) + 0^Cube[coor][3]  

* (2 + 2 * Cube[coor][2]);  

x := 4 * 0^(2 - Cube[coor][1] - Cube[coor][3]) - 4 * 0^(Cube[coor][1]  

+ Cube[coor][3]);  

y := 0;  

C := Cube[coul];  

if t <= 2 then  

S := plots:-polygonplot( [[x, y], [x + (-1)^t, y], [x + (-1)^t, y - 1],  

[x, y - 1], [x, y]], color = C[2]);  

S := S, plots:-polygonplot( [[x - 0^(t-1), y - 0^(2-t)], [x - 0  

^(t-1) + (-1)^t, y - 0^(2-t)], [x - 0^(t-1) + (-1)^t, y - 0  

^(2-t) - 1], [x - 0^(t-1), y - 0^(2-t) - 1], [x - 0^(t-1),  

y - 0^(2-t)]], color = C[3]);  

S := S, plots:-polygonplot( [[x + 0^(2-t), y - 0^(t-1)], [x + 0  

^(2-t) + (-1)^t, y - 0^(t-1)], [x + 0^(2-t) + (-1)^t, y - 0  

^(t-1) - 1], [x + 0^(2-t), y - 0^(t-1) - 1], [x + 0^(2-t),  

y - 0^(t-1)]], color = C[1])  

else  

S := plots:-polygonplot( [[x, y], [x + (-1)^t, y], [x + (-1)^t, y + 1],  

[x, y + 1], [x, y]], color = C[2]);  

S := S, plots:-polygonplot( [[x + 0^(4-t), y + 0^(t-3)], [x + 0  

^(4-t) + (-1)^t, y + 0^(t-3)], [x + 0^(4-t) + (-1)^t, y + 0  

^(t-3) + 1], [x + 0^(4-t), y + 0^(t-3) + 1], [x + 0^(4-t), y + 0  

^(t-3)]], color = C[3]);  

S := S, plots:-polygonplot( [[x - 0^(t-3), y + 0^(4-t)], [x - 0  

^(t-3) + (-1)^t, y + 0^(4-t)], [x - 0^(t-3) + (-1)^t, y + 0

```

```

 $^{\wedge}(4 - t) + 1], [x - 0^{\wedge}(t - 3), y + 0^{\wedge}(4 - t) + 1], [x - 0^{\wedge}(t - 3), y$ 
 $+ 0^{\wedge}(4 - t)]], color = C[1])$ 
end if;
S
end proc
```

***Fonction qui trace le patron d'un Rubix:**

```

DessinePatron :=proc(Rubix)
display(DessinePetitCube(Rubix[1]), DessinePetitCube(Rubix[2]),
DessinePetitCube(Rubix[3]), DessinePetitCube(Rubix[4]),
DessinePetitCube(Rubix[5]), DessinePetitCube(Rubix[6]),
DessinePetitCube(Rubix[7]), DessinePetitCube(Rubix[8]), view = [-4 ..4, -4 ..4],
title = "Patron du cube", axes = none);
end proc;

proc(Rubix)
plots:-display(DessinePetitCube(Rubix[1]), DessinePetitCube(Rubix[2]),
DessinePetitCube(Rubix[3]), DessinePetitCube(Rubix[4]),
DessinePetitCube(Rubix[5]), DessinePetitCube(Rubix[6]),
DessinePetitCube(Rubix[7]), DessinePetitCube(Rubix[8]), view = [-4 ..4,
-4 ..4], title = "Patron du cube", axes = none)
end proc
```

(3.1.2.1)

B)Le volume

***Fonction qui renvoie les plot d'un petit cube donnée:**

```

DessineCube :=proc(x, y, z, c1, c2, c3, c4, c5, c6,$)
local S;
S := polygonplot3d([ [x, y, z], [x + 1, y, z], [x + 1, y, z + 1], [x, y, z + 1]], color = c1);
S := S, polygonplot3d([ [x + 1, y, z], [x + 1, y + 1, z], [x + 1, y + 1, z + 1], [x + 1, y, z
+ 1]], color = c2);
S := S, polygonplot3d([ [x, y + 1, z], [x + 1, y + 1, z], [x + 1, y + 1, z + 1], [x, y + 1, z
+ 1]], color = c3);
S := S, polygonplot3d([ [x, y, z], [x, y + 1, z], [x, y + 1, z + 1], [x, y, z + 1]], color = c4);
S := S, polygonplot3d([ [x, y, z], [x + 1, y, z], [x + 1, y + 1, z], [x, y + 1, z]], color = c5);
S := S, polygonplot3d([ [x, y, z + 1], [x + 1, y, z + 1], [x + 1, y + 1, z + 1], [x, y + 1, z
+ 1]], color = c6);
end proc;

proc(x, y, z, c1, c2, c3, c4, c5, c6, $) (3.2.1.1)
local S;
S := plots:-polygonplot3d([ [x, y, z], [x + 1, y, z], [x + 1, y, z + 1], [x, y, z
+ 1]], color = c1);
S := S, plots:-polygonplot3d([ [x + 1, y, z], [x + 1, y + 1, z], [x + 1, y + 1, z + 1], [x + 1, y, z + 1]], color = c2);
S := S, plots:-polygonplot3d([ [x, y + 1, z], [x + 1, y + 1, z], [x + 1, y + 1, z + 1], [x, y + 1, z + 1]], color = c3);
```

```

+ 1], [x, y + 1, z + 1]], color = c3);
S := S, plots:-polygonplot3d([[x, y, z], [x, y + 1, z], [x, y + 1, z + 1], [x, y, z
+ 1]], color = c4);
S := S, plots:-polygonplot3d([[x, y, z], [x + 1, y, z], [x + 1, y + 1, z], [x, y + 1,
z]], color = c5);
S := S, plots:-polygonplot3d([[x, y, z + 1], [x + 1, y, z + 1], [x + 1, y + 1, z
+ 1], [x, y + 1, z + 1]], color = c6)
end proc

```

***Fonction qui renvoie les plot d'un petit cube donnée:**

```

DessineVolume :=proc(Rubix,$)
display( DessineCube( -1, -1, -1, Rubix[ 1 ][coul][ 2 ], black, black, Rubix[ 1 ][coul][ 3 ],
Rubix[ 1 ][coul][ 1 ], black),
DessineCube( 0, -1, -1, Rubix[ 2 ][coul][ 2 ], Rubix[ 2 ][coul][ 1 ], black, black,
Rubix[ 2 ][coul][ 3 ], black),
DessineCube( -1, -1, 0, Rubix[ 3 ][coul][ 2 ], black, black, Rubix[ 3 ][coul][ 1 ], black,
Rubix[ 3 ][coul][ 3 ]),
DessineCube( 0, -1, 0, Rubix[ 4 ][coul][ 2 ], Rubix[ 4 ][coul][ 3 ], black, black, black,
Rubix[ 4 ][coul][ 1 ]),
DessineCube( -1, 0, -1, black, black, Rubix[ 5 ][coul][ 2 ], Rubix[ 5 ][coul][ 1 ],
Rubix[ 5 ][coul][ 3 ], black),
DessineCube( 0, 0, -1, black, Rubix[ 6 ][coul][ 3 ], Rubix[ 6 ][coul][ 2 ], black,
Rubix[ 6 ][coul][ 1 ], black),
DessineCube( -1, 0, 0, black, black, Rubix[ 7 ][coul][ 2 ], Rubix[ 7 ][coul][ 3 ], black,
Rubix[ 7 ][coul][ 1 ]),
DessineCube( 0, 0, 0, black, Rubix[ 8 ][coul][ 1 ], Rubix[ 8 ][coul][ 2 ], black, black,
Rubix[ 8 ][coul][ 3 ]),
projection = 0.6, orientation = [ -52, 60 ], title = "Représentation 3D du cube");
end proc;
proc(Rubix, $) (3.2.2.1)
plots:-display(DessineCube( -1, -1, -1, Rubix[ 1 ][coul][ 2 ], black, black,
Rubix[ 1 ][coul][ 3 ], Rubix[ 1 ][coul][ 1 ], black), DessineCube( 0, -1, -1,
Rubix[ 2 ][coul][ 2 ], Rubix[ 2 ][coul][ 1 ], black, black, Rubix[ 2 ][coul][ 3 ], black),
DessineCube( -1, -1, 0, Rubix[ 3 ][coul][ 2 ], black, black, Rubix[ 3 ][coul][ 1 ],
black, Rubix[ 3 ][coul][ 3 ]), DessineCube( 0, -1, 0, Rubix[ 4 ][coul][ 2 ], Rubix[ 4
][coul][ 3 ], black, black, Rubix[ 4 ][coul][ 1 ]), DessineCube( -1, 0, -1,
black, black, Rubix[ 5 ][coul][ 2 ], Rubix[ 5 ][coul][ 1 ], Rubix[ 5 ][coul][ 3 ], black),
DessineCube( 0, 0, -1, black, Rubix[ 6 ][coul][ 3 ], Rubix[ 6 ][coul][ 2 ], black,
Rubix[ 6 ][coul][ 1 ], black), DessineCube( -1, 0, 0, black, black, Rubix[ 7 ][coul]
[ 2 ], Rubix[ 7 ][coul][ 3 ], black, Rubix[ 7 ][coul][ 1 ]), DessineCube( 0, 0, 0, black,
Rubix[ 8 ][coul][ 1 ], Rubix[ 8 ][coul][ 2 ], black, black, Rubix[ 8 ][coul][ 3 ]),
projection = 0.6, orientation = [ -52, 60 ], title = "Représentation 3D du cube")
end proc

```

C)La création d'un rubix cube

*Fonction qui creer un rubik's cube à partir de ses couleurs donnée dans l'ordre de référence:

CreerRubix :=proc(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24,\$)

```
[[[ [ 0 ], [ C[c19] ] ], [[ 1 ], [ C[c5] ]], [[ 0 ], [ C[c16] ]], [[ 1 ], [ C[c22] ]],  
[[ 0 ], [ C[c1] ]], [[ 0 ], [ C[c2] ]], [[ 1 ], [ C[c3] ]], [[ 1 ], [ C[c4] ]],  
[[ 1 ], [ C[c14] ]], [[ 0 ], [ C[c20] ]], [[ 1 ], [ C[c21] ]], [[ 0 ], [ C[c7] ]],  
[[ 0 ], [ C[c13] ]], [[ 1 ], [ C[c18] ]], [[ 0 ], [ C[c23] ]], [[ 1 ], [ C[c8] ]],  
[[ 0 ], [ C[c10] ]], [[ 0 ], [ C[c9] ]], [[ 1 ], [ C[c12] ]], [[ 1 ], [ C[c11] ]],  
[[ 0 ], [ C[c17] ]], [[ 1 ], [ C[c6] ]], [[ 0 ], [ C[c15] ]], [[ 1 ], [ C[c24] ]]]];  
end proc;
```

proc(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, (3.3.1.1)

c20, c21, c22, c23, c24, \$)

```
[[Vector(3, {1 = 0, 2 = 0, 3 = 1}), Vector(3, {1 = C[c19], 2 = C[c1], 3 = C[c14]}), [Vector(3, {1 = 1, 2 = 0, 3 = 0}), Vector(3, {1 = C[c5], 2 = C[c2], 3 = C[c20]}), [Vector(3, {1 = 0, 2 = 1, 3 = 1}), Vector(3, {1 = C[c16], 2 = C[c3], 3 = C[c21]}), [Vector(3, {1 = 1, 2 = 1, 3 = 0}), Vector(3, {1 = C[c22], 2 = C[c4], 3 = C[c7]}), [Vector(3, {1 = 0, 2 = 0, 3 = 0}), Vector(3, {1 = C[c13], 2 = C[c10], 3 = C[c17]}), [Vector(3, {1 = 1, 2 = 0, 3 = 1}), Vector(3, {1 = C[c18], 2 = C[c9], 3 = C[c6]}), [Vector(3, {1 = 0, 2 = 1, 3 = 0}), Vector(3, {1 = C[c23], 2 = C[c12], 3 = C[c15]}), [Vector(3, {1 = 1, 2 = 1, 3 = 1}), Vector(3, {1 = C[c8], 2 = C[c11], 3 = C[c24]}))]];
```

end proc

*Fonction qui creer aléatoirement un rubik's cube:

CreerRubixAleatoire :=proc({n :: integer := 30})

```
local M, Rubix, k;  
Rubix := RUBIX.FINI;
```

#On initialise les différents mouvement possible

```
M[1] := A;  
M[2] := A";  
M[3] := P;  
M[4] := P";  
M[5] := B;  
M[6] := B";  
M[7] := H;  
M[8] := H";  
M[9] := B;  
M[10] := B";  
M[11] := R;  
M[12] := R";
```

```

 $M[13] := L;$ 
 $M[14] := L'';$ 
#On effectue n fois des opération élémentaire sur le rubix de référence
for k to n do
    Rubix := Rafraichir(BougerRubix(Rubix, M[rand(1..14)( )], patron
    =false, cube=true, volume=false));
    end do;
    Rubix;
end proc;

proc( {n::integer := 30}) (3.3.2.1)
    local M, Rubix, k;
    Rubix := Typesetting:-delayDotProduct(RUBIX, FINI);
    M[1]:=A;
    M[2]:=A";
    M[3]:=P;
    M[4]:=P";
    M[5]:=B;
    M[6]:=B";
    M[7]:=H;
    M[8]:=H";
    M[9]:=B;
    M[10]:=B";
    M[11]:=R;
    M[12]:=R";
    M[13]:=L;
    M[14]:=L";
    for k to n do
        Rubix := Rafraichir(BougerRubix(Rubix, M[rand(1..14)( )], patron =false,
        cube =true, volume =false))
    end do;
    Rubix
end proc

```

D)Changement d'origine

*Fonction qui change le petit cube n°1 avec celui donné comme origine:

```

ChangerDeCoin :=proc(a, b, n,$)
    local k, x, y, z, s;
    s := NULL;
    x := NumeroVersCoordonnees(a)[1];
    y := NumeroVersCoordonnees(a)[2];
    z := NumeroVersCoordonnees(a)[3];
    for k to x·(x + z) do s := s, y0",y1" end do;
    for k to y do s := s, x0, x1 end do;

```

```

for k to  $0^x + z$  do s := s, z0, z1 end do;
end proc;

proc(a, b, n, §) (3.4.1.1)
  local k, x, y, z, s;
  s := NULL;
  x := NumeroVersCoordonnees(a)[1];
  y := NumeroVersCoordonnees(a)[2];
  z := NumeroVersCoordonnees(a)[3];
  for k to  $x * (x + z)$  do s := s, y0", y1" end do;
  for k to y do s := s, x0, x1 end do;
  for k to  $0^z(x + z)$  do
    s := s, z0, z1
  end do
end proc

```

▼ E) Deplacement du Rubick's Cube

***Fonction qui execute une série de mouvement sur un rubick's cube:**

```

BougerRubix :=proc( { patron :: boolean := true, volume :: boolean := true, cube
  :: boolean := false, frequence :: posint := top } )
  local k, Rubix, seq, i, p, n;

```

#On initialise les variables

```

#Le premier argument est le rubick's cube considéré
Rubix := op(1, [args]);
#Indice pour stocker les différents états du rubix cube
i := 2;
#Indice pour compter le nombre d'option entré
p := 0;
#Sequence de tous les états du cube
seq[1] := Rafraichir(Rubix);

```

#On effectue toutes les opérations demandées

```

for k in [op(2 ..nops([args]), [args])] do
  #On fait rien si c'est une option
  if type(k, boolean) or type(k, posint) then p := p + 1 else
    #Sinon on stock le rubix et son graphe
    Rubix := k(Rubix);
    seq[i] := Rafraichir(Rubix);
    i := i + 1;
  end if;
end do;

```

#On renvoie ce que l'on a demander

```

if patron then print(animate( DessinePatron, [seq[frequence·n]], n
= [ $\frac{1}{frequence}$ , `\$` $\left(1 \dots \text{floor}\left(\frac{\text{nops}([\text{args}])-p}{frequence}\right)\right)$ ,  $\frac{i-1}{frequence}$  ]) ) end if;

```

```

        if volume then print(animate(DessineVolume, [seq[frequence*n]], n
= [  $\frac{1}{frequence}$ , `\$`(1 .. floor(  $\frac{nops([args])-p}{frequence}$  ),  $\frac{i-1}{frequence}$  )] ) end if;
        if cube then Rubix; end if;
        end proc;

proc( {cube:boolean := false, frequence:posint := top, patron:boolean := true,      (3.5.1.1)
       volume:boolean := true})
local k, Rubix, seq, i, p, n;
Rubix := op(1, [args]);
i := 2;
p := 0;
seq[1] := Rafraichir(Rubix);
for k in [op(2 .. nops([args]), [args]) ] do
  if type(k, boolean) or type(k, posint) then
    p := p + 1
  else
    Rubix := k(Rubix); seq[i] := Rafraichir(Rubix); i := i + 1
  end if
end do;
if patron then
  print(plots:-animate(DessinePatron, [seq[frequence*n]], n = [1
  /frequence, `\$`(1 .. floor((nops([args]) - p)/frequence)), (i - 1)
  /frequence]));
end if;
if volume then
  print(plots:-animate(DessineVolume, [seq[frequence*n]], n = [1
  /frequence, `\$`(1 .. floor((nops([args]) - p)/frequence)), (i - 1)
  /frequence]));
end if;
if cube then Rubix end if
end proc

```

IV) La résolution

A) Les mouvements clefs

*Mouvement de base:

Qui échanger la position du bloc 1 et 2 et l'orientation des blocs 1, 2 et 3

MouvementDeBase := L", H", L", H", R, H, L, H, L, B, B;

RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH", RotationBlocR, (4.1.1.1)

RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B, B

On l'abrège:

mb := MouvementDeBase;

RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH", RotationBlocR, (4.1.1.2)
RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B, B

*Mouvement d'échange:

Mouvement qui échange le petit cube 1 et 2 sans en changer l'orientation

MouvementDEchange := mb, mb, mb;

RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH", RotationBlocR, (4.1.2.1)
RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B, B,
RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH",
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B, RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH",
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B

On l'abrège:

me := MouvementDEchange;

RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH", RotationBlocR, (4.1.2.2)
RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B, B,
RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH",
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B, RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH",
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B

*Mouvement d'orientation:

Mouvement qui change l'orientation du petit cube 1 et 2.

MouvementDOrientation := B, mb, mb, B, mb, B", mb, mb, B, mb, B, B;

B, RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH", (4.1.3.1)
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B, RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH",
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B, B, RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH",
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B, B", RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH",
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B, RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH",
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B, B, RotationBlocL", RotationBlocH", RotationBlocL", RotationBlocH",
RotationBlocR, RotationBlocH, RotationBlocL, RotationBlocH, RotationBlocL, B,
B, B, B

On l'abrège:

$mo := \text{MouvementDOrientation};$
 $B, \text{RotationBlocL}^", \text{RotationBlocH}^", \text{RotationBlocL}^", \text{RotationBlocH}^",$ (4.1.3.2)
 $\text{RotationBlocR}, \text{RotationBlocH}, \text{RotationBlocL}, \text{RotationBlocH}, \text{RotationBlocL}, B,$
 $B, \text{RotationBlocL}^", \text{RotationBlocH}^", \text{RotationBlocL}^", \text{RotationBlocH}^",$
 $\text{RotationBlocR}, \text{RotationBlocH}, \text{RotationBlocL}, \text{RotationBlocH}, \text{RotationBlocL}, B,$
 $B, B, \text{RotationBlocL}^", \text{RotationBlocH}^", \text{RotationBlocL}^", \text{RotationBlocH}^",$
 $\text{RotationBlocR}, \text{RotationBlocH}, \text{RotationBlocL}, \text{RotationBlocH}, \text{RotationBlocL}, B,$
 $B, B^", \text{RotationBlocL}^", \text{RotationBlocH}^", \text{RotationBlocL}^", \text{RotationBlocH}^",$
 $\text{RotationBlocR}, \text{RotationBlocH}, \text{RotationBlocL}, \text{RotationBlocH}, \text{RotationBlocL}, B,$
 $B, \text{RotationBlocL}^", \text{RotationBlocH}^", \text{RotationBlocL}^", \text{RotationBlocH}^",$
 $\text{RotationBlocR}, \text{RotationBlocH}, \text{RotationBlocL}, \text{RotationBlocH}, \text{RotationBlocL}, B,$
 $B, B, \text{RotationBlocL}^", \text{RotationBlocH}^", \text{RotationBlocL}^", \text{RotationBlocH}^",$
 $\text{RotationBlocR}, \text{RotationBlocH}, \text{RotationBlocL}, \text{RotationBlocH}, \text{RotationBlocL}, B,$
 B, B, B

B) Echange de 2 petits cubes

*Mouvement pour échanger un seul petit cube avec le n°1:

MouvementPourEchanger1 := proc(a,\$)
local coord, s;
coord := NumeroVersCoordonnees(a);
#On traite chaque cas différent
if coord = [1, 0, 1] then s := R", me, R;
elif coord = [0, 1, 0] then s := P", B", me, B, P;
elif coord = [0, 1, 1] then s := A, me, A";
elif coord = [1, 1, 0] then s := R, me, R";
elif coord = [1, 1, 1] then s := R, R, me, R, R;
elif coord = [0, 0, 0] then s := B", me, B;
elif coord = [1, 0, 0] then s := me;
elif coord = [0, 0, 1] then s := NULL : end if;

#On renvoie la liste des mouvement à faire

*s;
end proc;*

proc(a, \$) (4.2.1.1)
local coord, s;
coord := NumeroVersCoordonnees(a);
if coord = [1, 0, 1] **then**
s := R", me, R
elif coord = [0, 1, 0] **then**
s := P", B", me, B, P
elif coord = [0, 1, 1] **then**
s := A, me, A"
elif coord = [1, 1, 0] **then**
s := R, me, R"
elif coord = [1, 1, 1] **then**

```

    s := R, R, me, R, R
elif coord = [0, 0, 0] then
    s := B", me, B
elif coord = [1, 0, 0] then
    s := me
elif coord = [0, 0, 1] then
    s := NULL
end if;
    s
end proc

```

***Mouvement pour échanger n'importe quel cube entre eux:**

```

MouvementPourEchanger :=proc(a, b,$)
    MouvementPourEchanger1(a), MouvementPourEchanger1(b),
    MouvementPourEchanger1(a);
    end proc;

proc(a, b, $)
    MouvementPourEchanger1(a), MouvementPourEchanger1(b),
    MouvementPourEchanger1(a)
end proc

```

(4.2.2.1)

On l'abrège:

```

mpe := MouvementPourEchanger;
    MouvementPourEchanger

```

(4.2.2.2)

C)Changement de l'orientation de 2 petits cubes

***Mouvement pour changer l'orientation d'un petit cube et du n°1:**

```

MouvementPourOrienter1 :=proc(a,$)
    local coord, s;
    coord := NumeroVersCoordonnees(a);
#On traite chaque cas différent
    if coord = [1, 0, 1] then s := R", mo, R;
    elif coord = [0, 1, 0] then s := P", B", mo, B, P;
    elif coord = [0, 1, 1] then s := A, mo, A";
    elif coord = [1, 1, 0] then s := R, mo, R";
    elif coord = [1, 1, 1] then s := R, R, mo, R, R;
    elif coord = [0, 0, 0] then s := B", mo, B;
    elif coord = [1, 0, 0] then s := mo;
    elif coord = [0, 0, 1] then s := NULL : end if;

```

#On renvoie la liste des mouvement à faire

```

        s
end proc;

proc(a, $)
    local coord, s;

```

(4.3.1.1)

```

coord := NumeroVersCoordonnees(a);
if coord = [1, 0, 1] then
    s := R'', mo, R
elif coord = [0, 1, 0] then
    s := P'', B'', mo, B, P
elif coord = [0, 1, 1] then
    s := A, mo, A''
elif coord = [1, 1, 0] then
    s := R, mo, R''
elif coord = [1, 1, 1] then
    s := R, R, mo, R, R
elif coord = [0, 0, 0] then
    s := B'', mo, B
elif coord = [1, 0, 0] then
    s := mo
elif coord = [0, 0, 1] then
    s := NULL
end if;
    s
end proc

```

***Mouvement pour échanger n'importe quel cube entre eux:**

```

MouvementPourOrienter :=proc(a, b,$)
    MouvementPourOrienter1(a), MouvementPourOrienter1(b);
end proc;

proc(a, b, $)
    MouvementPourOrienter1(a), MouvementPourOrienter1(b)
end proc

```

(4.3.2.1)

On l'abrège:

```

mpo := MouvementPourOrienter;
MouvementPourOrienter

```

(4.3.2.2)

D)Résolution

***Fonction qui place les petits cubes au bon endroit selon leur couleur:**

```

ResoudreCoordonnees :=proc(Rubix1,$)
    local Rubix2, s, k, temp, a, b;
    s := NULL;
    Rubix2 := Rubix1;
#On règle tout ça petit cube par petit cube
    for k to 8 do
        #On vérifie s'il y a raison de bouger
        a := CouleursVersNumero(      Rubix2[k][coul][1],

```

```

Rubix2[k][coul][2], Rubix2[k][coul][3]);
    b := CoordonneesVersNumero(Rubix2[k][coor][1],
    Rubix2[k][coor][2], Rubix2[k][coor][3]);
    if a ≠ b then temp := mpe(a, b);
        s := s, temp;
        Rubix2 := BougerRubix(Rubix2, temp, patron = false,
        volume = false, cube = true);
    end if; end do;
    s;
end proc;

proc(Rubix1, $) (4.4.1.1)
    local Rubix2, s, k, temp, a, b;
    s := NULL;
    Rubix2 := Rubix1;
    for k to 8 do
        a := CouleursVersNumero(Rubix2[k][coul][1], Rubix2[k][coul][2], Rubix2
        [k][coul][3]);
        b := CoordonneesVersNumero(Rubix2[k][coor][1], Rubix2[k][coor][2],
        Rubix2[k][coor][3]);
        if a <> b then
            temp := mpe(a, b);
            s := s, temp;
            Rubix2 := BougerRubix(Rubix2, temp, patron = false, volume = false, cube
            = true)
        end if
    end do;
    s
end proc

```

***Fonction qui oriente les petits cubes correctement:**

```

ResoudreOrientations :=proc(Rubix1,$)
    local Rubix2, s, temp, i, j, k, a, b, c;
    s := NULL;
    Rubix2 := Rafraichir(Rubix1);
#On règle tout ça petit cube par petit cube
    for i from 1 to 8 do
        a := EstCycle(Rafraichir(Rubix2)[i][coul]);
        for j from i + 1 to 8 do
            b := EstCycle(Rafraichir(Rubix2)[j][coul]);
            if b ≠ 0 then
                if a = 1 then temp := mpo(i, j); s := s, temp; i := 0; j := 9;
                    Rubix2 := Rafraichir(BougerRubix(Rubix2, temp,
                    patron = false, volume = false, cube = true) );
                end if;
                if a = 2 then temp := mpo(j, i); s := s, temp; i := 0; j := 9;
                    Rubix2 := Rafraichir(BougerRubix(Rubix2, temp,
                    patron = false, volume = false, cube = true) );
            
```

```

end if;
end if; end do; end do;
s;
end proc;

proc(RubixI, s) (4.4.2.1)
  local Rubix2, s, temp, i, j, k, a, b, c;
  s := NULL;
  Rubix2 := Rafraichir(RubixI);
  for i to 8 do
    a := EstCycle(Rafraichir(Rubix2)[i][coul]);
    for j from i+1 to 8 do
      b := EstCycle(Rafraichir(Rubix2)[j][coul]);
      if b <> 0 then
        if a = 1 then
          temp := mpo(i,j);
          s := s, temp;
          i := 0;
          j := 9;
          Rubix2 := Rafraichir(BougerRubix(Rubix2, temp, patron = false,
          volume = false, cube = true))
        end if;
        if a = 2 then
          temp := mpo(j,i);
          s := s, temp;
          i := 0;
          j := 9;
          Rubix2 := Rafraichir(BougerRubix(Rubix2, temp, patron = false,
          volume = false, cube = true))
        end if
      end if
    end do
  end do;
  s
end proc

```

***Fonction tant attendu de résolution du rubick's cube:**

```

ResoudreRubix :=proc(RubixI,$)
  local Rubix2, s, temp, k;
  Rubix2 := RubixI;
#On vérifie que le cube est bien résoluble
  if not (Axiome1(Rubix2) and Axiome2et3(Rubix2) )
  then print("Cube non résoluble") else
#On résoud les coordonnées
  s := ResoudreCoordonnees(Rubix2);

```

```

    Rubix2 := Rafraichir(BougerRubix(Rubix2, s, patron =false, volume
    =false, cube =true));
#On résoud les orientations
    s := s, ResoudreOrientations(Rubix2);
    end if; end proc;
proc(Rubix1, $) (4.4.3.1)
    local Rubix2, s, temp, k;
    Rubix2 := Rubix1;
    if not (Axiome1(Rubix2) and Axiome2et3(Rubix2)) then
        print("Cube non résoluble")
    else
        s := ResoudreCoordonnees(Rubix2);
        Rubix2 := Rafraichir(BougerRubix(Rubix2, s, patron =false, volume =false,
        cube =true));
        s := s, ResoudreOrientations(Rubix2)
    end if
end proc

```

*Fonction finale de résolution d'un rubix créée:

```

Resoudre :=proc(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18,
c19, c20, c21, c22, c23, c24,
{freq :: posint := 1, vol :: boolean := true, pat :: boolean := false, mouv
:: boolean := false, coup :: boolean := true}, $)
    local Rubix, Mouvement:
    Rubix := CreerRubix(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14,
c15, c16, c17, c18, c19, c20, c21, c22, c23, c24);
    Mouvement := ResoudreRubix(Rubix);
    if coup then printf("Cube résolu en %g coups", nops([Mouvement])) end if;
    BougerRubix(Rubix, Mouvement, frequence = freq, volume = vol, patron = pat);
    if mouv then Mouvement end if;
end proc;

proc(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, (4.4.4.1)
c20, c21, c22, c23, c24, {coup:boolean := true, freq:posint := 1, mouv:
boolean := false, pat:boolean := false, vol:boolean := true}, $)
    local Rubix, Mouvement,
    Rubix := CreerRubix(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14,
c15, c16, c17, c18, c19, c20, c21, c22, c23, c24);
    Mouvement := ResoudreRubix(Rubix);
    if coup then printf("Cube résolu en %g coups", nops([Mouvement])) end if;
    BougerRubix(Rubix, Mouvement, frequence = freq, volume = vol, patron = pat);
    if mouv then Mouvement end if
end proc

```

E) Tentative (veine) de simplification

Vu le nombre d'opération nécessaire pour résoudre le rubick's cube avec la méthode précédent, on tente de faire un programme qui les simplifirait.

*Fonction de teste de l'égalité de deux rubick's cube:

```
EstEgale :=proc(Rub1, Rub2)
    local Rubix1, Rubix2, Bool, k, c;
    Bool := true;
    Rubix1 := Rafraichir(Rub1);
    Rubix2 := Rafraichir(Rub2);

    for k to 8 do
        for c to 3 do
            Bool := Bool and evalb(Rubix1[k][coul][c] = Rubix2[k][coul][c])
        end do; end do;
        Bool;
    end proc;

proc(Rub1, Rub2) (4.5.1.1)
    local Rubix1, Rubix2, Bool, k, c;
    Bool := true;
    Rubix1 := Rafraichir(Rub1);
    Rubix2 := Rafraichir(Rub2);
    for k to 8 do
        for c to 3 do
            Bool := Bool and evalb(Rubix1[k][coul][c] = Rubix2[k][coul][c])
        end do
    end do;
    Bool
end proc
```

*Fonction de simplification d'une suite de mouvement:

```
Simplifier1 :=proc()
    local mouv, k, l, i; i := 0;
    mouv := [args];
    l := nops(mouv);
    for k while k < l - 2 do
        if EstEgale(Rafraichir(BougerRubix(rubix, mouv[k], mouv[k + 1], patron
=false, volume=false, cube=true)), rubix) then i := i + 1;

        mouv := [op(1..k - 1, mouv), op(k + 2..l, mouv)]; end if;
        l := nops(mouv);
    end do; k := 1;
    for k while k < l - 4 do
        if EstEgale(Rafraichir(BougerRubix(rubix, mouv[k], mouv[k + 1], mouv[k
+ 2], mouv[k + 3], patron=false, volume=false, cube=true)), rubix) then i := i + 1;

        mouv := [op(1..k - 1, mouv), op(k + 4..l, mouv)]; end if;
```

```

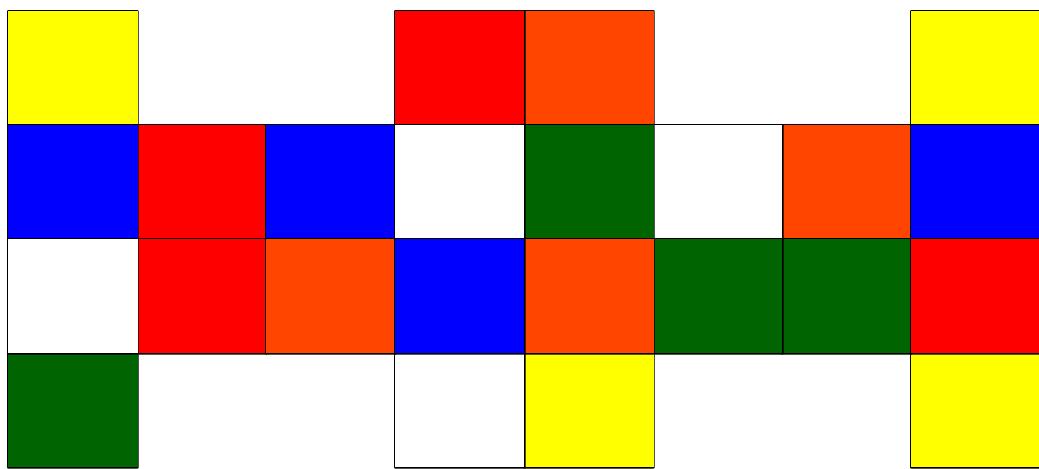
    l := nops(mouv);
end do;
    op(1 ..l, mouv), nops(mouv);
end proc;
proc( ) (4.5.2.1)
    local mouv, k, l, i;
    i := 0;
    mouv := [args];
    l := nops(mouv);
    for k while k < l - 2 do
        if EstEgale(Rafraichir(BougerRubix(rubix, mouv[k], mouv[k + 1], patron
        =false, volume=false, cube=true)), rubix) then
            i := i + 1; mouv := [op(1 ..k - 1, mouv), op(k + 2 ..l, mouv)]
        end if;
        l := nops(mouv)
    end do;
    k := 1;
    for k while k < l - 4 do
        if EstEgale(Rafraichir(BougerRubix(rubix, mouv[k], mouv[k + 1], mouv[k
        + 2], mouv[k + 3], patron=false, volume=false, cube=true)), rubix)
        then
            i := i + 1; mouv := [op(1 ..k - 1, mouv), op(k + 4 ..l, mouv)]
        end if;
        l := nops(mouv)
    end do;
    op(1 ..l, mouv), nops(mouv)
end proc

```

$Rub := CreerRubixAleatoire();$
 $BougerRubix(Rub, ResoudreRubix(Rub), freq=top)$
 $Rubix2$

Patron du cube

$$n = \frac{1}{100000}$$



Représentation 3D du cube

$$n = \frac{1}{100000}$$

