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) :
```

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 Chifre⇒Couleur(en Maple)

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

*Attributions des couleurs à chaque indice de C (initiales anglaises pour éviter les confusions bleu/blanc) :
Ces notations permettent exclusivement de facilité 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 "francais")⇒Chifre⇒Indice⇒Couleur(en Maple)

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

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 coordonnees sont eux aussi des tableaux donnée par 3 chiffres (d'après les notations adopté).
Les couleurs sont finallement 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...

$top := 100000$:

*Définitions des 8 cubes d'un Rubik's Cube fini:
Le numéro de chaque cube n'a aucune importance, ils sont prit dans l'ordre de la notation choisit.

$$CUBE.FINI[1] := \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} C[w] \\ C[r] \\ C[b] \end{bmatrix} \end{bmatrix}:$$

$$CUBE.FINI[2] := \begin{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} C[g] \\ C[r] \\ C[w] \end{bmatrix} \end{bmatrix}:$$

$$CUBE.FINI[3] := \begin{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} C[b] \\ C[r] \\ C[y] \end{bmatrix} \end{bmatrix}:$$

$$CUBE.FINI[4] := \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} C[y] \\ C[r] \\ C[g] \end{bmatrix} \end{bmatrix}:$$

$$CUBE.FINI[5] := \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} C[b] \\ C[o] \\ C[w] \end{bmatrix} \end{bmatrix}:$$

$$CUBE.FINI[6] := \begin{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} C[w] \\ C[o] \\ C[g] \end{bmatrix} \end{bmatrix}:$$

$$CUBE.FINI[7] := \begin{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} C[y] \\ C[o] \\ C[b] \end{bmatrix} \end{bmatrix}:$$

$$CUBE.FINI[8] := \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} C[g] \\ C[o] \\ C[y] \end{bmatrix} \end{bmatrix}:$$

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

```
coor := 1 :
coul := 2 :
```

*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]] :
```

(Compte tennu de l'utilisation ultra fréquente de ce dernier dans les arguments des autres fonctions, on prendra la notation *rubix* := *RUBIX.FINI* : jusqu'à l'aboutissement du programme)

*Fonction de correspondance n° Petit Cube ⇒ Coordonnée:
C'est une fonction qui donne les coordonnées corespondantes à un petit cube désigner par son numéro dans la notation choisit.

```
NumeroVersCoordonnees :=proc (num,$)
                          local x, y, z;
#La première coordonnée se récupère très facillement
                          x := 0^(num 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 finnallement le résultat
                          [x, y, z];
                          end proc:
```

*Fonction de correspondance Coordonnée ⇒n° Petit Cube:
C'est une fonction qui donne le numéro d'un petit cube dans la notation choisit 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:
```

*Fonction de correspondance Couleurs ⇒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:
```

*Fonction de rafraîchissement du Rubik's cube, qui le place dans sa notation standard:

```
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:
```

C) Les etats du Rubik's Cube

α) 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 formulle barbare renumérote chacun des petits cubes du Rubik's Cube selon leurs
         coordonnées
              #pour les faire correspondre avec la notation choisit.
              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 ;
#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:
```

β) 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 nécesaire et suffisant d'avoir:

- $\forall$ Cube1,Cube2$\in$Rubix,{Cube1[coul]}$\neq${Cube2[coul]}
- $\forall$ Cube$\in$Rubix,$\exists$ CUBE.FINI$\in$RUBIX.FINI $|\exists n\in\mathbb{N}|$ Cube[coul]=$c^n(CUBE.FINI[coul])$: où c(E) est un cycle de l'ensemble E
- $\Sigma n \equiv 0$ [3]

*Axiome 1:
Chaque petits cubes d'un Rubik's Cube faisable doivent necessairement 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:
```

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

```
EstCycle :=proc(Couleur,$)
        local k, n;
        n := ∞;
#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:
```

*Axiome 2-3:
L'orientation de chaque petits cubes d'un Rubik's Cube faisable doit necessairement 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:
```

II)Les 12 mouvements possibles

A)Selon X

*Fonction qui change les données d'un petit cube ayant subi une rotation du bloc X0 (dans le sens direct) :

```
RotationCubeX0 := 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)·(0^(z + 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)·0^(y + z mod 2)·1) mod 3;
                  e := (x·1 + (1 - x)·(y + z mod 2)·0 + (1 - x)·0^(y + z mod 2)·2) mod 3;
                  f := (x·2 + (1 - x)·(y + z mod 2)·1 + (1 - x)·0^(y + z mod 2)·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:

```
RotationBlocX0 := proc(Rubix1,$)
                  local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                  for k to 8 do
                  Rubix2[k] := RotationCubeX0(RotationCubeX0(RotationCubeX0(Rubix1[k])));
                  end do;
                  Rubix2;
                  end proc:
```

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

```
RotationBlocX0inv := proc(Rubix1,$)
                  local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                  for k to 8 do
                  Rubix2[k] := RotationCubeX0(Rubix1[k]);
                  end do;
                  Rubix2;
                  end proc:
```

Compte tennue de l'utilisation plus que récurente de ces deux fonctions pour la saisi des mouvement, on les nommes égallement, jusqu'à la fin du projet, de manière moins explicite mais plus simple par $x0 := RotationBlocX0:$ et $x0'' := RotationBlocX0inv:$

*Fonction qui change les données d'un petit cube ayant subi une rotation du bloc X1 (dans le sens direct) :

```
RotationCubeX1 := 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·(0^(z + 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·0^(y + z mod 2)·1) mod 3;
```

```
                    e := ((1 - x)·1 + x·(y + z mod 2)·0 + x·0^(y + z mod 2)·2) mod 3;
                    f := ((1 - x)·2 + x·(y + z mod 2)·1 + x·0^(y + z mod 2)·0) mod 3;
#On renvoie les nouvelles données du cube
                    [[[a], [b], [c]], [[C[d]], [C[e]], [C[f]]]];
                end proc:
```

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

```
RotationBlocX1 := proc(Rubix1,$)
                    local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                    for k to 8 do
                    Rubix2[k] := RotationCubeX1(Rubix1[k]);
                    end do;
                    Rubix2;
                    end proc:
```

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

```
RotationBlocX1inv := proc(Rubix1,$)
                    local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                    for k to 8 do
                    Rubix2[k] := RotationCubeX1(RotationCubeX1(RotationCubeX1(Rubix1[k])));
                    end do;
                    Rubix2;
                    end proc:
```

Compte tennue de l'utilisation plus que récurente de ces deux fonctions pour la saisi des mouvement, on les nommes égallement, jusqu'à la fin du projet, de manière moins explicite mais plus simple par `x1 := RotationBlocX1 :` et `x1'' := RotationBlocX1inv :`

B)Selon Y

*Fonction qui change les données d'un petit cube ayant subi une rotation du bloc Y0 (dans le sens direct) :

```
RotationCubeY0 := 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 + 0^z) mod 2;
                    b := y;
                    c := y·z + (1 − y)·(x + z + 0^x) 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], [b], [c]], [[C[d]], [C[e]], [C[f]]]];
                  end proc:
```

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

```
RotationBlocY0 := proc(Rubix1,$)
                    local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                    for k to 8 do
                   Rubix2[k] := RotationCubeY0(RotationCubeY0(RotationCubeY0(Rubix1[k])));
                   end do;
                   Rubix2;
                   end proc:
```

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

```
RotationBlocY0inv := proc(Rubix1,$)
                    local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                    for k to 8 do
                   Rubix2[k] := RotationCubeY0(Rubix1[k]);
                   end do;
                   Rubix2;
                   end proc:
```

Compte tennue de l'utilisation plus que récurente de ces deux fonctions pour la saisi des mouvement, on les nommes égallement, jusqu'à la fin du projet, de manière moins explicite mais plus simple par `y0 := RotationBlocY0:` et `y0'' := RotationBlocY0inv:`

*Fonction qui change les données d'un petit cube ayant subi une rotation du bloc X1 (dans le sens direct)

```
RotationCubeY1 := 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 + 0^z) mod 2;
                b := y;
                c := (1 - y)·z + y·(x + z + 0^x) 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], [b], [c]], [[C[d]], [C[e]], [C[f]]]];
            end proc:
```

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

```
RotationBlocY1 :=proc(Rubix1,$)
                local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                for k to 8 do
                Rubix2[k] := RotationCubeY1(RotationCubeY1(RotationCubeY1(Rubix1[k])));
                end do;
                Rubix2;
                end proc:
```

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

```
RotationBlocY1inv :=proc(Rubix1,$)
                local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                for k to 8 do
                Rubix2[k] := RotationCubeY1(Rubix1[k]);
                end do;
                Rubix2;
                end proc:
```

Compte tennue de l'utilisation plus que récurente de ces deux fonctions pour la saisi des mouvement, on les nommes égallement, jusqu'à la fin du projet, de manière moins explicite mais plus simple par $y1 := RotationBlocY1:$ et $y1'' := RotationBlocY1inv:$

C)Selon Z

*Fonction qui change les données d'un petit cube ayant subi une rotation du bloc X0 (dans le sens direct) :

```
RotationCubeZ0 := 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 mod 2)·x + (1 − (x + z mod 2))·(x + z + 0^(x + y + z mod 2) mod 2);
                    b := (x + z mod 2)·y + (1 − (x + z mod 2))·(y + x + 1 + 0^(x + y + z mod 2) mod 2);
                    c := (x + z mod 2)·z + (1 − (x + z mod 2))·(x + z + 0^(x + y + z mod 2) mod 2);
#On renvoie les nouvelles données du cube
                    [[[a], [b], [c]], [[C[0]], [C[1]], [C[2]]]];
                end proc:
```

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

```
RotationBlocZ0 := proc(Rubix1,$)
                    local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                    for k to 8 do
                    Rubix2[k] := RotationCubeZ0(Rubix1[k]);
                    end do;
                    Rubix2;
                    end proc:
```

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

```
RotationBlocZ0inv := proc(Rubix1,$)
                    local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                    for k to 8 do
                    Rubix2[k] := RotationCubeZ0(RotationCubeZ0(RotationCubeZ0(Rubix1[k])));
                    end do;
                    Rubix2;
                    end proc:
```

Compte tennue de l'utilisation plus que récurente de ces deux fonctions pour la saisi des mouvement, on les nommes égallement, jusqu'à la fin du projet, de manière moins explicite mais plus simple par `z0 := RotationBlocZ0:` et `z0'' := RotationBlocZ0inv:`

*Fonction qui change les données d'un petit cube ayant subi une rotation du bloc X1 (dans le sens direct) :

```
RotationCubeZ1 := 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 + 0^(x + y + z mod 2) mod 2);
                    b := (1 - (x + z mod 2)) · y + (x + z mod 2) · (x + y + 0^(x + y + z mod 2) mod 2);
                    c := (1 - (x + z mod 2)) · z + (x + z mod 2) · (x + z + 1 + 0^(x + y + z mod 2) mod 2);
#On renvoie les nouvelles données du cube
                    [[[a], [b], [c]], [[C[0]], [C[1]], [C[2]]]];
                end proc:
```

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

```
RotationBlocZ1 := proc(Rubix1,$)
                    local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                    for k to 8 do
                   Rubix2[k] := RotationCubeZ1(Rubix1[k]);
                   end do;
                   Rubix2;
                   end proc:
```

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

```
RotationBlocZ1inv := proc(Rubix1,$)
                    local k, Rubix2;
 #On change tous les petits cube du Rubik's considéré
                    for k to 8 do
                   Rubix2[k] := RotationCubeZ1(RotationCubeZ1(RotationCubeZ1(Rubix1[k])));
                   end do;
                   Rubix2;
                   end proc:
```

Compte tennue de l'utilisation plus que récurente de ces deux fonctions pour la saisi des mouvement, on les nommes égallement, jusqu'à la fin du projet, de manière moins explicite mais plus simple par `z1 := RotationBlocZ1 :` et `z1'' := RotationBlocZ1inv :`

III)Les outils

A)Le patron

*Fonction qui dessine un petit carre:

```
DessinePetitCube := proc(Cube,$)
                    local S, x, y, t, C;
 #On récupère les coordonnées
                    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]) - 0^(Cube[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:
```

*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:
```

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:
```

*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:
```

C)La saisi 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,$)
```

$$\left[\left[\begin{bmatrix}0\\0\\1\end{bmatrix},\begin{bmatrix}C[c19]\\C[c1]\\C[c14]\end{bmatrix}\right],\left[\begin{bmatrix}1\\0\\0\end{bmatrix},\begin{bmatrix}C[c5]\\C[c2]\\C[c20]\end{bmatrix}\right],\left[\begin{bmatrix}0\\1\\1\end{bmatrix},\begin{bmatrix}C[c16]\\C[c3]\\C[c21]\end{bmatrix}\right],\left[\begin{bmatrix}1\\1\\0\end{bmatrix},\begin{bmatrix}C[c22]\\C[c4]\\C[c7]\end{bmatrix}\right],\left[\begin{bmatrix}0\\0\\0\end{bmatrix},\begin{bmatrix}C[c13]\\C[c10]\\C[c17]\end{bmatrix}\right],\left[\begin{bmatrix}1\\0\\1\end{bmatrix},\begin{bmatrix}C[c18]\\C[c9]\\C[c6]\end{bmatrix}\right],\left[\begin{bmatrix}0\\1\\0\end{bmatrix},\begin{bmatrix}C[c23]\\C[c12]\\C[c15]\end{bmatrix}\right],\left[\begin{bmatrix}1\\1\\1\end{bmatrix},\begin{bmatrix}C[c8]\\C[c11]\\C[c24]\end{bmatrix}\right]\right];$$

```
                end proc:
```

D)Changement d'origine

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

```
ChangerDeCoin :=proc(a,$)
```

```
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;
s;
end proc:
```

E)Deplacement du 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 = [1/frequence,
    `$`(1..floor((nops([args]) - p)/frequence)), (i - 1)/frequence])) end if;
        if volume then print(animate(DessineVolume, [seq[frequence·n]], n = [1/frequence,
    `$`(1..floor((nops([args]) - p)/frequence)), (i - 1)/frequence])) end if;
        if cube then Rubix; end if;
        end proc:
```

IV)La résolutions

A)Les mouvements clefs

*Mouvement de base:
Qui échanger la posistion du bloc 1 et 2 et l'orientation des blocs 1, 2 et 3

MouvementDeBase := *x0, y1", x0, y1", x1, y1, x0", y1, x0", y0, y0* :

On l'abrège *mb* := *MouvementDeBase* :

*Mouvement d'échange:
Mouvment qui echange le petit cube 1 et 2 sans en changer l'orientation

MouvementDEchange := *mb, mb, mb* :

On l'abrège *me* := *MouvementDEchange* :

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

MouvementDOrientation := *y0", mb, mb, y0", mb, y0, mb, mb, y0", mb, y0, y0* :

On l'abrège *mo* := *MouvementDOrientation* :

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 := x1", me, x1;
                         elif coord = [0, 1, 0] then s := z0, y0, me, y0", z0";
                         elif coord = [0, 1, 1] then s := z1, me, z1";
                         elif coord = [1, 1, 0] then s := x1, me, x1";
                         elif coord = [1, 1, 1] then s := x1, x1, me, x1, x1;
                         elif coord = [0, 0, 0] then s := y0, me, y0";
                         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:
```

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

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

On l'abrège *mpe* := *MouvementPourEchanger* :

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 := x1", mo, x1;
                         elif coord = [0, 1, 0] then s := z0, y0, mo, y0", z0";
                         elif coord = [0, 1, 1] then s := z1, mo, z1";
                         elif coord = [1, 1, 0] then s := x1, mo, x1";
                         elif coord = [1, 1, 1] then s := x1, x1, mo, x1, x1;
                         elif coord = [0, 0, 0] then s := y0, mo, y0";
                         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:
```

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

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

On l'abrège *mpo* := *MouvementPourOrienter* :

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:
```

*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:
```

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

```
ResoudreRubix :=proc(Rubix1,$)
                    local Rubix2, s, temp, k;
                    Rubix2 := Rubix1;
#On vérifie que le cube est bien résoluble
                    if not (Axiome1(Rubix2) and Axiome2et3(Rubix2))
    then print("Cube non résolube") 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:
```

*Fonction finale de résolution d'un rubix crée:

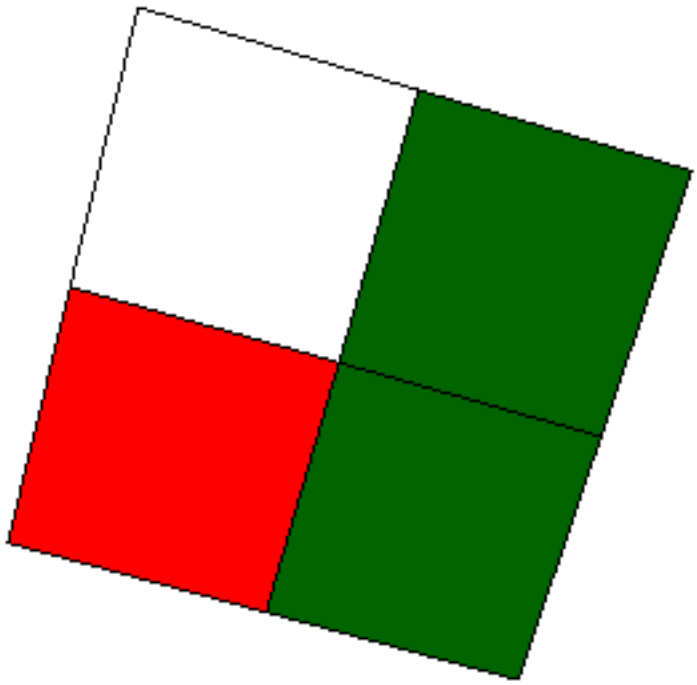
```
Resoudre :=proc(c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20,
    c21, c22, c23, c24,
                    {freq :: posint := 1, vol :: boolean := true, pat :: bollean := false, mouv :: boolean
     := false, coup :: boolean := true},$)
          local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

Resoudre(*o*, *y*, *o*, *g*, *r*, *y*, *y*, *r*, *b*, *w*, *w*, *o*, *b*, *g*, *y*, *b*, *r*, *o*, *w*, *b*, *w*, *r*, *g*, *g*)

```
Cube résolu en 1594 coups
```

Représentation 3D du cube
n = 1



V)Simplification

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

*Fonction de simplification d'une suite de mouvement:

```
Simplifier := proc( )
              local k;
```