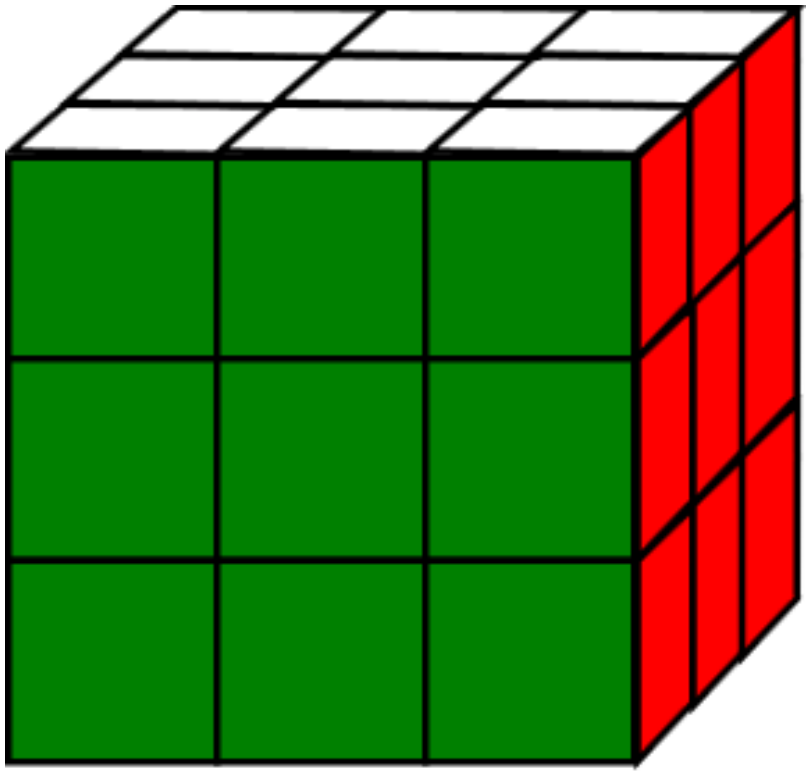
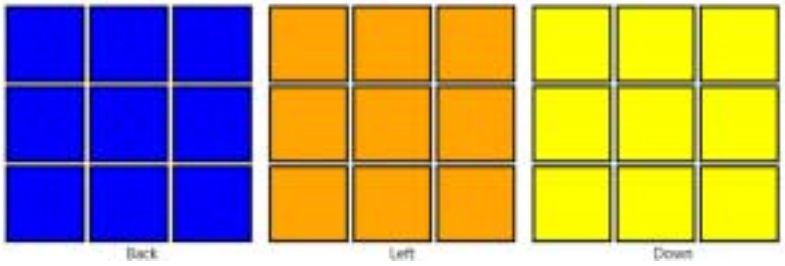
RUBIK’S:Algorithme de Dieu



- Des études théoriques ont démontré qu’il y a un nombre limité de coups qui représente la solution de tout état du Rubiks.
- Des expériences menées par GOOGLE (350 machines en fonctionnement parallèle) ont approuvé cette théorie.

RUBIK’S:Algorithme de Dieu



- L’algorithme permettant de réaliser le nombre minimal de coups pour atteindre la solution est appelé algorithme de Dieu
- Nous essayerons de le redéfinir dans les pages qui suivent. Mais avant tout nous allons réaliser un programme qui permet sa recherche.

RUBIK’S:le cube

- Description du cube :
  - Entité constituante
    - Fields
      - Type
      - Position : coordonnées
      - Couleur
      - Visibilité
    - Mouvement

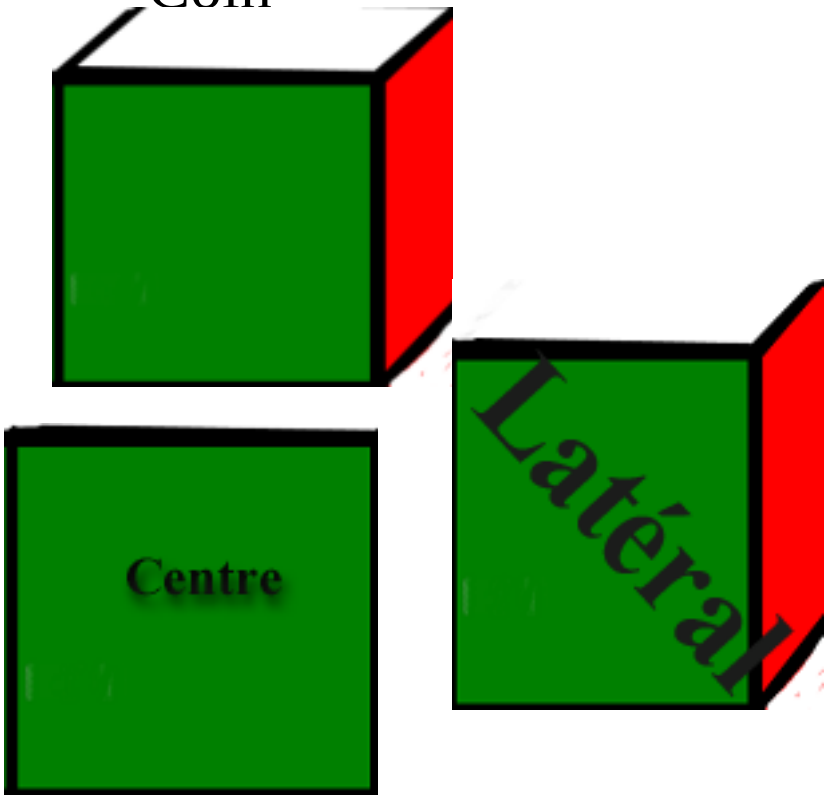
  En termes objet les champs (fields) et les méthodes.

Entité constituante

- Description plane 2D
  - Un carré appartient à
  - Une face et a
  - Une couleur :
    - Blanc(0)
    - Orange(1)
    - Vert(2)
    - Rouge(3)
    - Bleu(4)
    - Jaune(5)
- Les 6 faces à 9 carrés chacune, définissent un tableau linéaire de 54 carrés avec les couleurs correspondantes. Ainsi l'élément qui se trouve sur la face f à la position p a pour couleur : couleur[9*f+p]
- Le cube peut être considéré comme un tableau de 54 éléments dont chaque membre contient une couleur (numérotée de 0 à 5)

Entité constituante

- Description 3D
  - Coin



- Le cube est constitué de 3 types d'éléments:
  - Coins au nombre de 8
  - Latéraux au nombre de 12
  - Centres au nombre de 6

Entité constituante

- Un cube est constitué de
  - 8 coins à 3 faces chacun.
  - 12 latéraux à 2 faces chacun
  - 6 centres à une face chacun
- Un coin a 3 types de faces : frontale (devant ,derrière)-latérale(droite,gauche)-Horizontale(haut,bas)
- **Un latéral est un coin avec une face inexistante**
- **Un centre est un coin avec 2 faces inexistantes**

Entité Constituante

```
public static ObservableCollection<Pieces> AllEntities;//26
Class Pieces
{
        private int nPiece;
        private string nomPiece;
        private string couleurFront;
        private string couleurHaut;
        private string couleurDroit;
}
void InitEntity()
         {
             InitCoins();
             InitLateraux();
             InitCentres();
         }
```

Entité Constituante

```
    #region Coins
        public void InitCoins()
        {
            AllEntities = new ObservableCollection<Pieces>();
            /*Les coins
             * CoinSupGF CoinSupDF CoinInfGF CoinInfDF
             * CoinSupGA CoinSupDA CoinInfGA CoinInfDA
             */
            AllEntities.Add(new Pieces { NPiece = 0, CouleurHaut = "white", CouleurFront = "green", CouleurDroit = "orange", NomPiece = "CoinSupGF" });
            AllEntities.Add(new Pieces { NPiece = 1, CouleurHaut = "white", CouleurFront = "green", CouleurDroit = "red", NomPiece = "CoinSupDF" });
            AllEntities.Add(new Pieces { NPiece = 2, CouleurHaut = "yellow", CouleurFront = "green", CouleurDroit = "orange", NomPiece = "CoinInfGF" });
            AllEntities.Add(new Pieces { NPiece = 3, CouleurHaut = "yellow", CouleurFront = "green", CouleurDroit = "red", NomPiece = "CoinInfDF" });
            AllEntities.Add(new Pieces { NPiece = 4, CouleurHaut = "white", CouleurFront = "blue", CouleurDroit = "orange", NomPiece = "CoinSupGA" });
            AllEntities.Add(new Pieces { NPiece = 5, CouleurHaut = "white", CouleurFront = "blue", CouleurDroit = "red", NomPiece = "CoinSupDA" });
            AllEntities.Add(new Pieces { NPiece = 6, CouleurHaut = "yellow", CouleurFront = "blue", CouleurDroit = "orange", NomPiece = "CoinInfGA" });
            AllEntities.Add(new Pieces { NPiece = 7, CouleurHaut = "yellow", CouleurFront = "blue", CouleurDroit = "red", NomPiece = "CoinInfDA" });
        }
        #endregion
```

Entité Constituante

```
#region Latéraux
public void InitLateraux()
        {
/* * Pour les latéraux:    *LatSupF LatMidFG LatMidFD LatInfF    * LatSupG LatMidG LatInfG *LatSupD LatMidD LatInfD     *LatSupA LatInfA            */
            AllEntities.Add(new Pieces { NPiece = 8, CouleurHaut = "white", CouleurFront = "green", CouleurDroit = "black", NomPiece = "LatSupF" });
            AllEntities.Add(new Pieces { NPiece = 9, CouleurHaut = "black", CouleurFront = "green", CouleurDroit = "orange", NomPiece = "LatMidFG" });
            AllEntities.Add(new Pieces { NPiece = 10, CouleurHaut = "black", CouleurFront = "green", CouleurDroit = "red", NomPiece = "LatMidFD" });
            AllEntities.Add(new Pieces { NPiece = 11, CouleurHaut = "yellow", CouleurFront = "green", CouleurDroit = "black", NomPiece = "LatInfF" });
            AllEntities.Add(new Pieces { NPiece = 12, CouleurHaut = "white", CouleurFront = "black", CouleurDroit = "orange", NomPiece = "LatSupG" });
            AllEntities.Add(new Pieces { NPiece = 13, CouleurHaut = "black", CouleurFront = "blue", CouleurDroit = "orange", NomPiece = "LatMidG" });
            AllEntities.Add(new Pieces { NPiece = 14, CouleurHaut = "yellow", CouleurFront = "black", CouleurDroit = "orange", NomPiece = "LatInfG" });
            AllEntities.Add(new Pieces { NPiece = 15, CouleurHaut = "white", CouleurFront = "black", CouleurDroit = "red", NomPiece = "LatSupD" });
            AllEntities.Add(new Pieces { NPiece = 16, CouleurHaut = "black", CouleurFront = "blue", CouleurDroit = "red", NomPiece = "LatMidD" });
            AllEntities.Add(new Pieces { NPiece = 17, CouleurHaut = "yellow", CouleurFront = "black", CouleurDroit = "red", NomPiece = "LatInfD" });
            AllEntities.Add(new Pieces { NPiece = 18, CouleurHaut = "white", CouleurFront = "blue", CouleurDroit = "black", NomPiece = "LatSupA" });
            AllEntities.Add(new Pieces { NPiece = 19, CouleurHaut = "yellow", CouleurFront = "blue", CouleurDroit = "black", NomPiece = "LatInfA" });
        }
        #endregion
```

Entité constituante

```
    #region InitCentres
        public void InitCentres()
        {
            /*
                *Pour les centres
                *CentH CentB CentA CentG CentD CentF
            */
            AllEntities.Add(new Pieces { NPiece = 20, CouleurHaut = "black", CouleurFront = "green", CouleurDroit = "black", NomPiece = "CentF" });
            AllEntities.Add(new Pieces { NPiece = 21, CouleurHaut = "black", CouleurFront = "black", CouleurDroit = "orange", NomPiece = "CentG" });
            AllEntities.Add(new Pieces { NPiece = 22, CouleurHaut = "black", CouleurFront = "black", CouleurDroit = "red", NomPiece = "CentD" });
            AllEntities.Add(new Pieces { NPiece = 23, CouleurHaut = "black", CouleurFront = "blue", CouleurDroit = "black", NomPiece = "CentA" });
            AllEntities.Add(new Pieces { NPiece = 24, CouleurHaut = "white", CouleurFront = "black", CouleurDroit = "black", NomPiece = "CentH" });
            AllEntities.Add(new Pieces { NPiece = 25, CouleurHaut = "yellow", CouleurFront = "black", CouleurDroit = "black", NomPiece = "CentB" });
        }
        #endregion
```

OBJETS

- L’entité constituante peut être considérée comme un coin à 3 faces avec la propriété de visibilité ou d’existence qui permet de générer les centres et les latéraux.
- Propriétés de l’objet:
  - Type de la face :Front – Droit – Haut- Bas – Gauche- Derrière
  - Position : coordonnées
  - Couleur
  - Visibilité
  - Existence

OBJET de base:3 faces

- <Rectangle Name="front" MouseDown="FrontClick" Fill="green" HorizontalAlignment="Left" Height="60" Margin="46.8,104.4,0,0" VerticalAlignment="Top" Width="62" Loaded="FrontLoaded"/>
- <Path Name="droit" MouseDown="DroitClick" Fill="red" Data="M124.7,89 L124.6,146.10001 107.80025,163.05001 107.80025,104.55 z" HorizontalAlignment="Left" Height="75.05" Margin="107.8,89,0,0" Stretch="Fill" VerticalAlignment="Top" Width="17.9" Loaded="DroitLoaded"/>
- <Path Name="haut" MouseDown="HautClick" Fill="white" Data="M124.5,89.65 L63.6,89.150002 47.599998,103.55 107.8,104.55 z" HorizontalAlignment="Left" Height="16.4" Margin="47.6,89.15,0,0" Stretch="Fill" VerticalAlignment="Top" Width="77.9" Loaded="HautLoaded"/>

Méthodes

```
    private void FrontLoaded(object sender, RoutedEventArgs e)
            {
                Rectangle rFront = (Rectangle)sender;
                string sLien = this.Name;
                char[] seperator = { 'F', 'D' };
                string[] fWords = sLien.Split(seperator);
                int i = Convert.ToInt32(fWords[1]);

                if (i == 50) { rFront.Visibility = Visibility.Hidden; return; }
                if (i > 8) { rFront.Visibility = Visibility.Hidden; i = 4 * 9 + i - 9; }
                else { rFront.Visibility = Visibility.Visible; i = i + 2 * 9; }

                Binding bindElmt = new Binding("Couleur")
                {
                    Mode = BindingMode.TwoWay,
                    Source = CubeEntityViewModel.AllCube[i]
                };
                DrawStroke(rFront);
                rFront.SetBinding(Rectangle.FillProperty, bindElmt);
            }
```

Méthodes

```
    private void FrontClick(object sender, MouseButtonEventArgs e)
           {
    string sName = this.Name;
    Coordonnees pc = (CubeEntityViewModel.CoordCube.First(item => item.Lien == sName));
               switch(CubeEntityViewModel.CoordCube.IndexOf(pc))
               {
                   case 0: CubeEntityViewModel.TurnFirstLayerToLeft(); break;
                   case 1: CubeEntityViewModel.TurnUp(); break;
                   case 2: CubeEntityViewModel.TurnFirstLayerToRight(); break;
                   case 3: CubeEntityViewModel.TurnSecondLayerToLeft(); break;
                   case 5: CubeEntityViewModel.TurnSecondLayerToRight(); break;
                   case 6: CubeEntityViewModel.TurnThirdLayerToLeft(); break;
                   case 7: CubeEntityViewModel.TurnDown(); break;
                   case 8: CubeEntityViewModel.TurnThirdLayerToRight(); break;
               }
           }
```

Méthodes

```
 private void HautLoaded(object sender, RoutedEventArgs e)
        {
            Path pHaut = (Path)sender;
            string sLien = this.Name;
            char[] seperator = { 'H', 'F', 'D' };
            string[] hWords = sLien.Split(seperator);
            int i = Convert.ToInt32(hWords[1]);
            if (i == 50) { pHaut.Visibility = Visibility.Hidden; return; }
            if (i > 8) { pHaut.Visibility = Visibility.Hidden; i = 9 * 5 + i - 9; }
            else pHaut.Visibility = Visibility.Visible;

            Binding bindElmt = new Binding("Couleur");
            bindElmt.Mode = BindingMode.OneWay;
            bindElmt.Source = CubeEntityViewModel.AllCube[i];

DrawStroke(pHaut);
            pHaut.SetBinding(Rectangle.FillProperty, bindElmt);
        }
        public void DrawStroke(dynamic obj)
        {
            SolidColorBrush sbr = new SolidColorBrush()
            {
                Color = Color.FromRgb(0, 0, 0)
            };
            obj.StrokeThickness = 2;
            obj.Stroke = sbr;
        }
```

Méthodes

```
    private void HautClick(object sender, MouseButtonEventArgs e)
           {
               string sName = this.Name;
               Coordonnees pc = (CubeEntityViewModel.CoordCube.First(item => item.Lien == sName));
               switch ((CubeEntityViewModel.CoordCube.IndexOf(pc)))
               {
                   case 0: CubeEntityViewModel.TurnLeftRangeDown(); break;
                   case 1: CubeEntityViewModel.TurnMiddleRangeDown(); break;
                   case 2: CubeEntityViewModel.TurnRightRangeDown(); break;
                   case 12: CubeEntityViewModel.TurnLeftRangeUp(); break;
                   case 13: CubeEntityViewModel.TurnMiddleRangeUp(); break;
                   case 14: CubeEntityViewModel.TurnRightRangeUp(); break;
               }
           }
    private void DroitClick(object sender, MouseButtonEventArgs e)
           {
               string sName = this.Name;
               Coordonnees pc = (CubeEntityViewModel.CoordCube.First(item => item.Lien == sName));
               switch ((CubeEntityViewModel.CoordCube.IndexOf(pc)))
               {
                   case 2: CubeEntityViewModel.TurnFrontToLeft(); break;
                   case 5: CubeEntityViewModel.TurnLeft(); break;
                   case 8: CubeEntityViewModel.TurnFrontToRight(); break;
                   case 11: CubeEntityViewModel.TurnMiddleToLeft(); break;
                   case 14: CubeEntityViewModel.TurnLastToLeft(); break;
                   case 16: CubeEntityViewModel.TurnRight(); break;
                   case 17: CubeEntityViewModel.TurnMiddleToRight(); break;
                   case 18: CubeEntityViewModel.TurnLastToRight(); break;
               }
           }
```

Méthodes

```
    private void DroitLoaded(object sender, RoutedEventArgs e)
           {
               Path pDroit = (Path)sender;
               string sLien = this.Name;
               char[] seperator = { 'D' };
               string[] dWords = sLien.Split(seperator);
               int i = Convert.ToInt32(dWords[1]);

               if (i == 50) { pDroit.Visibility = Visibility.Hidden; return; }
               if (i > 8) { pDroit.Visibility = Visibility.Hidden; }
               else { pDroit.Visibility = Visibility.Visible; i = 3 * 9 + i; }

               Binding bindElmt = new Binding("Couleur")
               {
                   Mode = BindingMode.OneWay,
                   Source = CubeEntityViewModel.AllCube[i]
               };
               //Coordonnees pc = (CubeEntityViewModel.CoordCube.First(item => item.Lien == sLien));
               DrawStroke(pDroit);
               pDroit.SetBinding(Rectangle.FillProperty, bindElmt);
           }
```

Propriété

- La pièce maîtresse qui lie l'entité à ses coordonnées est définie en propriété

```
 #region DependencyProperty NPiece qui indique la pièce
        public static readonly DependencyProperty NPieceProperty =
DependencyProperty.Register("NPiece", typeof(int), typeof(CubeEntity), new PropertyMetadata(0, new
PropertyChangedCallback(OnNPieceChanged)));
        public int NPiece
        {
            get           { return (int)GetValue(NPieceProperty); }
            set           { SetValue(NPieceProperty, value);      }
        }
   private static void OnNPieceChanged(DependencyObject depObj, DependencyPropertyChangedEventArgs e)
        {
            CubeEntity FrameControl = depObj as CubeEntity;
            FrameControl.OnNPieceChanged(e);
        }
        private void OnNPieceChanged(DependencyPropertyChangedEventArgs e)
        {
            if (e.NewValue != null) NPiece = Convert.ToInt32(e.NewValue);
        }
        #endregion
```

Données associées

- La description du cube peut se faire en un tableau de 54 éléments
  - `public static ObservableCollection<CubeDesc> AllCube = new ObservableCollection<CubeDesc>();//54`
  - CubeDesc a deux champs :
    - `private int nPiece;`
    - `private string couleur;`
- Le nombre d'entités constituantes est 26 : 8 coins+12 latéraux+ 6 centres
  - `public static ObservableCollection<Pieces> AllEntities;//26`
  - Pieces a 5 champs:
    - `private int nPiece;`
    - `private string nomPiece;`
    - `private string couleurFront;`
    - `private string couleurHaut;`
    - `private string couleurDroit;`
- 19 pièces sont visibles ce qui donne lieu à une liste de coordonnées
  - `public static List<Coordonnees> CoordCube;//19`
  - CoordCube a 4 champs :
    - `private int nPiece; private string lien; private int xPoint; private int yPoint;`

Assemblage:Cube

```
 <Grid Name="gdForCube">
</Grid>
 public Rubiks()
        {
            InitializeComponent();
            this.DataContext = new CubeEntityViewModel();
            InitCube();
        }
        private void InitCube()
        {
            Grid gd = gdForCube;
if (gd.Children.Count < 19)
            {
                for (int i = 0; i < 19; i++)
                {
                    double leftThick = (double)CubeEntityViewModel.CoordCube[i].XPoint;
                    double topThick = (double)CubeEntityViewModel.CoordCube[i].YPoint;
                    string lien = CubeEntityViewModel.CoordCube[i].Lien;
                    CubeEntity cb = new CubeEntity
                    {
                        Name = lien,
                        Margin = new Thickness(leftThick, topThick, 0, 0),
                        HorizontalAlignment = HorizontalAlignment.Left,
                        VerticalAlignment = VerticalAlignment.Top
                    };
                    CubeElmts.Add(cb);
                    gd.Children.Add(cb);
                }
            }
 else  {string s; for (int i = 0; i < 54; i++){s = WhatColor(i);CubeEntityViewModel.AllCube[i].Couleur = s;}}
        }
 private string WhatColor(int i)
{           return (i < 9) ? "white«  : (i < 18) ? "orange«  : (i < 27) ? "green«  : (i < 36) : "red«  : (i < 45) ?
"blue« : "yellow";}
```

Les Coordonnées

- Load Coordonnées
- Définition des liens
- Définition de la visibilité : Tout élément Contenant 50 dans son nom est invisible D50 v dire que la face D est invisible D0 v dire que la face gauche est visible position 0. D9 face droite position 0 est visible. H50-H0-H9
- Initialisation du Descriptif

```
public void LoadCoordonnees()
        {
            #region Coordonnées
            if (CoordCube is null)
            {
                CoordCube = new List<Coordonnees>();
                for (int k = 0; k < 19; k++)
                {
                    CoordCube.Add(new Coordonnees { XPoint = 0, YPoint = 0 });
                }
            }
            int Face = 0;
            int j;
            for (int i = 0; i < 3; i++)//Front première ligne
            {
                CoordCube[Face * 9 + i].XPoint = 10 + 60 * i;
                CoordCube[Face * 9 + i].YPoint = 0;
                //Front 2ème ligne
                CoordCube[Face * 9 + i + 3].XPoint = 10 + 60 * i;
                CoordCube[Face * 9 + i + 3].YPoint = 58;
                //Front 3ème ligne
                CoordCube[Face * 9 + i + 6].XPoint = 10 + 60 * i;
                CoordCube[Face * 9 + i + 6].YPoint = 116;
            }
```

Les Coordonnées

```
                    Face = 1;                int i = 1;
                    CoordCube[Face * 9 + i - 1].XPoint = 10 + 16 * i;
                    CoordCube[Face * 9 + i - 1].YPoint = -14 * i;
                    CoordCube[Face * 9 + i].XPoint = 70 + 16 * i;
                    CoordCube[Face * 9 + i].YPoint = -14 * i;
                    CoordCube[Face * 9 + i + 1].XPoint = 130 + 15 * i;
                    CoordCube[Face * 9 + i + 1].YPoint = -14 * i;
                    i = 2;                j = 3;
                    CoordCube[Face * 9 + j].XPoint = 10 + 16 * i;
                    CoordCube[Face * 9 + j].YPoint = -14 * i;
                    CoordCube[Face * 9 + j + 1].XPoint = 70 + 16 * i;
                    CoordCube[Face * 9 + j + 1].YPoint = -14 * i;
                    CoordCube[Face * 9 + j + 2].XPoint = 130 + 15 * i;
                    CoordCube[Face * 9 + j + 2].YPoint = -14 * i;
                Face = 2;            j = 3;
                for (int i = 0; i < 2; i++)//Droite uniquement 4 entités centre et inf
                {
            CoordCube[Face * 9 - j].XPoint = 146 + 14 * i; CoordCube[Face * 9 - j--].YPoint = -14 * i + 42;
                }
                for (int i = 0; i < 2; i++)//Droite uniquement 4 entités centre et inf
                {
        CoordCube[17 + i].XPoint = 146 + 14 * i;                CoordCube[17 + i].YPoint = -14 * i + 100;
                }
                #endregion Coordonnées
```

Mouvements

- Nous distinguons 2 types de mouvements:
  - Total : mouvements de direction
  - Partiel: mouvement de couches

Mouvement Total

- Quatre mouvements de direction sont possibles:
  - Haut
  - Bas
  - Gauche
  - Droite

Mouvement Partiel

- Le cube est constitué de 3 couches horizontales et 3 couches verticales et 3 couches transversales
  - Pour chaque couche 2 mouvements sont possibles
    - Horizontales : devant , derrière
    - Verticales : haute , basse
    - Transversales : gauche , droite

Mouvement Partiel

- Une classe CubeInfo pourrait résumer ces mouvements en fusionnant des noms pour éviter les confusions. Il en résulte 22 mouvements entre total et partiel. (D 4 – P 18=3(T-H-V))*2(L,R)*3(couches).
- ```
  public const int CubeUp = 0; public const int CubeDown = 1; public const int CubeLeft = 2; public const int CubeRight = 3; ***Direction - Total
  ```
- ```
  public const int TransFrontLeft = 4; public const int TransFrontRight = 5;public const int TransBackLeft = 6; public const int TransBackRight = 7; public const int TransMiddleLeft = 8;  public const int TransMiddleRight = 9;
  ```
- ```
  public const int VertLeftDown = 10; public const int VertLeftUp = 11; public const int VertRightDown = 12; public const int VertRightUp = 13; public const int VertMiddleDown = 14;public const int VertMiddleUp = 15;
  ```
- ```
  public const int HorizFirstLayerToLeft = 16;public const int HorizFirstLayerToRight = 17;public const int HorizMiddleLayerToLeft = 18;public const int HorizMiddleLayerToRight = 19;public const int HorizLastLayerToLeft = 20; public const int HorizLastLayerToRight = 21;
  ```

CubeInfo

- Cette classe peut être utilisée pour annoncer en clair le mouvement effectué. Il suffit d'avoir une liste de type string qui contient la correspondance entre l'action et sa signification.

```
–    public static List<string> AllMessages = new List<string>
–                {
–                    "Haut","Bas","Gauche","Droite","Trans Front Gauche","Trans Front
   Droite","Trans Arrière Gauche", "Trans Arrière Droite","Trans Milieu
   Gauche","Trans Milieu Droite",              "Vert Gauche Bas", "Vert Gauche
   Haut", "Vert Droite Bas","Vert Droite Haut", "Vert Milieu Bas",
   "Vert Milieu Haut","Horiz Haut Gauche","Horiz Haut Droite","Horiz Milieu
   Gauche","Horiz Milieu Droite","Horiz Bas Gauche","Horiz Bas Droite"
–                };
–   private int actionValue;
```

Mouvements TurnUP

```
    public static void TurnUp()
          {
              //2=5,5=4,0=2,4=0  Mvt 3 Mvt 1
              List<CubeDesc> AllCubeTemp = new List<CubeDesc>();
              CopyTemp(AllCubeTemp);
              string schema = "0=6,2=0,8=2,6=8,4=4,5=1,1=3,3=7,7=5";
              int face = 3;
              MouvementInterne(AllCubeTemp, schema, face);
              face = 1;
              schema = "6=0,8=6,2=8,0=2,4=4,1=5,3=1,7=3,5=7";
              MouvementInterne(AllCubeTemp, schema, face);
              for (int i = 0; i < 9; i++)
              {
                  //                face = 0; origine = 2;
                  CopyFaces(AllCubeTemp, 0, 2, i, i);
                  //                face = 2; origine = 5;
                  CopyFaces(AllCubeTemp, 2, 5, i, i);
                  //                face = 4; origine = 0;
                  CopyFaces(AllCubeTemp, 4, 0, i, i);
                  //                face = 5; origine = 4;
                  CopyFaces(AllCubeTemp, 5, 4, i, i);
              }
          }
```

TurnUP : Haut

```
     public static void CopyFaces(List<CubeDesc> AllCubeTemp, int face, int origine, int a, int b)
            {
                AllCube[face * 9 + a].NPiece = AllCubeTemp[origine * 9 + b].NPiece;
                AllCube[face * 9 + a].Couleur = AllCubeTemp[origine * 9 + b].Couleur;
            }
            public static void CopyTemp(List<CubeDesc> AllCubeTemp)
            {
                for (int i = 0; i < 54; i++)
                {
                    AllCubeTemp.Add(new CubeDesc { Couleur = AllCube[i].Couleur, NPiece = AllCube[i].NPiece
    });
                }
            }
            public static void MouvementInterne(List<CubeDesc> AllCubeTemp, string schema, int face)
            {
                char[] separator = { '=', ',' };
                string[] sDecoupage = schema.Split(separator);
                for (int i = 0; i < 18; i = i + 2)
                {
                    int j = Convert.ToInt32(sDecoupage[i]);
                    int k = Convert.ToInt32(sDecoupage[i + 1]);
                    AllCube[face * 9 + j].NPiece = AllCubeTemp[face * 9 + k].NPiece;
                    AllCube[face * 9 + j].Couleur = AllCubeTemp[face * 9 + k].Couleur;
                }
            }
```

Détails

- **Au début de notre exposé, nous avons défini rapidement la façon de numéroter les carrés: 9*f+p . Les faces sont aussi reconnues par un numéro associé. Ainsi la face 0 est la face haute. 1 celle de gauche, 2 la frontale, 3 celle de droite et 4 celle de derrière, 5 est en bas.**
- **Si on effectue un mouvement du cube (directionnel) vers le haut .. Ceci impliquera un déplacement de la face frontale vers le haut. 2 → 0 - 0 → 4 – 4 → 5 – 5 → 2 ceci est assurée par la méthode CopyFaces**
- **Les deux faces de droite et de gauche subiront une permutation des cases de chacune ainsi la face 1 (gauche) aura les cases suivantes permutées de la sorte:**
- **0 ➔ 6**
- **1 ➔ 3**
- **2 ➔ 0**
- **3 ➔ 7**
- **4 inchangé avec un quart de tour interne .. Autrement s'il y a une ligne droite qui pointe vers la face 2. Elle pointera en haut**
- **5 ➔ 1**
- **6 ➔ 8**
- **7 ➔ 5**
- **8 ➔ 2**
- **Ceci définira le schéma de la méthode Mouvement Interne**

Mouvement Partiel:Exemple

```
    #region Left Horiz
        public static void TurnFirstLayerToLeft()
        {
            List<CubeDesc> AllCubeTemp = new List<CubeDesc>();
            CopyTemp(AllCubeTemp);
            int face = 0;
            string schema = "2=0,8=2,6=8,0=6,4=4,5=1,7=5,3=7,1=3";
            MouvementInterne(AllCubeTemp, schema, face);
            for (int i = 0; i < 3; i++)
            {
                //origine = 4; face = 3;
                CopyFaces(AllCubeTemp, 3, 4, i, 6 + (2 - i));
                //face = 2; origine = 3;
                CopyFaces(AllCubeTemp, 2, 3, i, i);
                // face = 1;  origine = 2;
                CopyFaces(AllCubeTemp, 1, 2, i, i);
                //face = 4; origine = 1;
                CopyFaces(AllCubeTemp, 4, 1, 6 + (2 - i), i);
            }
        }
        #endregion
```

Méthode clé:CubePush

- **Les 22 actions**

```
public void CubePush(int action)
        {
            switch (action)
            {
                case CubeInfo.CubeUp: CubeEntityViewModel.TurnUp(); break;
                case CubeInfo.CubeDown: CubeEntityViewModel.TurnDown(); break;
                case CubeInfo.CubeLeft: CubeEntityViewModel.TurnLeft(); break;
                case CubeInfo.CubeRight: CubeEntityViewModel.TurnRight(); break;
                case CubeInfo.TransFrontLeft: CubeEntityViewModel.TurnFrontToLeft(); break;
                case CubeInfo.TransFrontRight: CubeEntityViewModel.TurnFrontToRight(); break;
                case CubeInfo.TransBackLeft: CubeEntityViewModel.TurnLastToLeft(); break;
                case CubeInfo.TransBackRight: CubeEntityViewModel.TurnLastToRight(); break;
                case CubeInfo.TransMiddleLeft: CubeEntityViewModel.TurnMiddleToLeft(); break;
                case CubeInfo.TransMiddleRight: CubeEntityViewModel.TurnMiddleToRight(); break;
                case CubeInfo.VertLeftDown: CubeEntityViewModel.TurnLeftRangeDown(); break;
                case CubeInfo.VertLeftUp: CubeEntityViewModel.TurnLeftRangeUp(); break;
                case CubeInfo.VertRightDown: CubeEntityViewModel.TurnRightRangeDown(); break;
                case CubeInfo.VertRightUp: CubeEntityViewModel.TurnRightRangeUp(); break;
                case CubeInfo.VertMiddleDown: CubeEntityViewModel.TurnMiddleRangeDown(); break;
                case CubeInfo.VertMiddleUp: CubeEntityViewModel.TurnMiddleRangeUp(); break;
                case CubeInfo.HorizFirstLayerToLeft: CubeEntityViewModel.TurnFirstLayerToLeft(); break;
                case CubeInfo.HorizFirstLayerToRight: CubeEntityViewModel.TurnFirstLayerToRight(); break;
                case CubeInfo.HorizMiddleLayerToLeft: CubeEntityViewModel.TurnSecondLayerToLeft(); break;
                case CubeInfo.HorizMiddleLayerToRight: CubeEntityViewModel.TurnSecondLayerToRight(); break;
                case CubeInfo.HorizLastLayerToLeft: CubeEntityViewModel.TurnThirdLayerToLeft(); break;
                case CubeInfo.HorizLastLayerToRight: CubeEntityViewModel.TurnThirdLayerToRight(); break;
            }
        }
```

Exemple Algorithmique

- Quatre mouvements déplacent les centres de 0 → 2, 2 → 1 , 1 → 0, 3 → 5, 5 → 4, 4→3 :

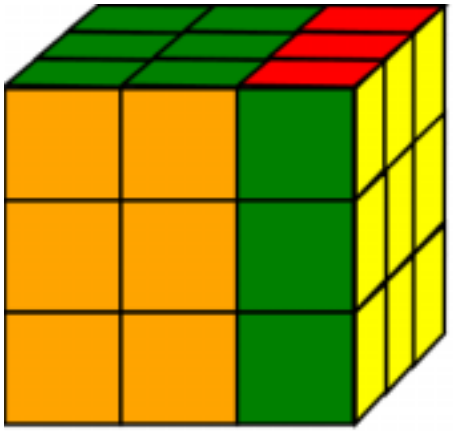
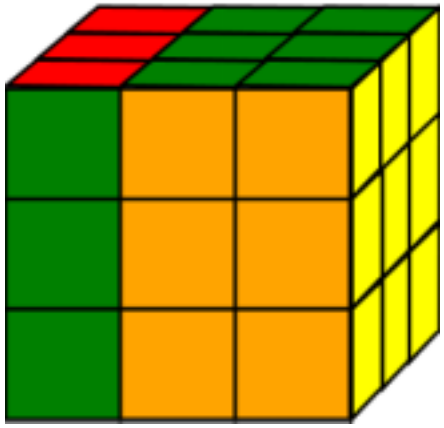
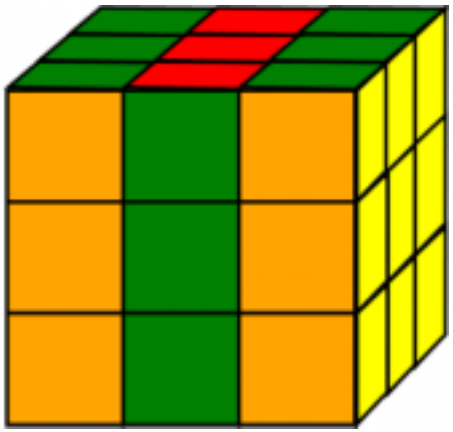
```
 public void PushCube(int action)
        {
            cube.CubePush(action);
            string s = CubeInfo.AllMessages[action];
            lbMvts.Items.Add(s); // pour garder trace
        }
        private void Centre ()
        {
            PushCube(CubeInfo.VertMiddleDown);
            PushCube(CubeInfo.HorizMiddleLayerToRight);
            PushCube(CubeInfo.VertMiddleUp);
            PushCube(CubeInfo.HorizMiddleLayerToLeft);
        }
```

Rubik’s:Approches

- Les algorithmes des solutions sont nombreux, le plus simple à mettre en place est la solution de couches.
  - La face Haute en Croix et Coins
  - Les côtés latéraux pour la première couche avec un algorithme nommé PA
  - Les coins hauts des parties frontale ,droite, gauche et derrière
  - La ligne médiane : 2ème couche
  - La face basse : 3ème couche
- D’autres variantes algorithmiques peuvent être utilisées pour réaliser des solution qui ne sont pas basées sur les couches.
- Une façon de prouver l’existence d’un algorithme de dieu est de balayer toutes les solutions possibles à partir d’un état. Ceci pourrait prendre un temps énorme pour chaque état. Aucun algorithme n’est mis en place. Un parcours de l’état initial vers la solution s’effectue en testant tous les états possibles. Cette méthode exigera un fonctionnement en parallèle et une recherche de solutions dans plusieurs threads. La solution atteinte arrêtera la recherche.
- La méthode la plus intelligente est de considérer qu’il y a n états intermédiaires entre l’état initial EI et l’état final EF:
  - Un état intermédiaire est un passage obligé. Sachant que l’EF est unique, le calcul des passages obligés doit se faire à partir de l’EF en remontant vers l’EI.
  - Un passage obligé est un état qui couvre toutes les symétries et représentatif d’un groupe d’états qui est lui même un ensemble d’états symétriques. Ex. : une face avec 3 identiques sur une colonne(ou ligne) et les 6 autres sont identiques représente un ensemble d’états symétriques à l’état décrit.
  - La recherche devrait se faire d’un passage obligé de rang n vers un autre passage obligé de rang n-1
  - Quand l’état initial appartient à un groupe d’états obligés, la solution est donnée.
  - Le nombre de saut définit le seuil associé à l’algorithme de dieu.

Passage obligé: rang n-1

Cet état représenye un passage obligé au rang n-1.
On ne peut atteindre l'EF sans avoir au rang précédent cet état ou un état dérivé.
Il est clair qu'avec un seul mouvement nous atteignons l'EF
Ces 3 états avec leurs symétries appartiennent au même groupe et représente un passage obligé pour atteindre la solution quelque soit l'EI. Ce qui est intéressant dans cet ensemble c'est la méthode unique qui pourrait générer les divers états du groupe: 18 possibles

Hypothèse de travail

- Si le nombre de mouvements nécessaires de l'EI à l'EF ne peut pas dépasser un seuil théorique qui est de 20 pas. Le nombre d'états possibles est 18 ^ 20 soit 12.748236.216396.078174.437376 ce qui représente un nombre très élevée d'états intermédiaires.
- Les états possibles peuvent être générés à partir d'une vingtaine de méthodes dont chacune produira 18 états.
- La réelle problématique sera de trouver les relations d'équivalences entre les divers états.