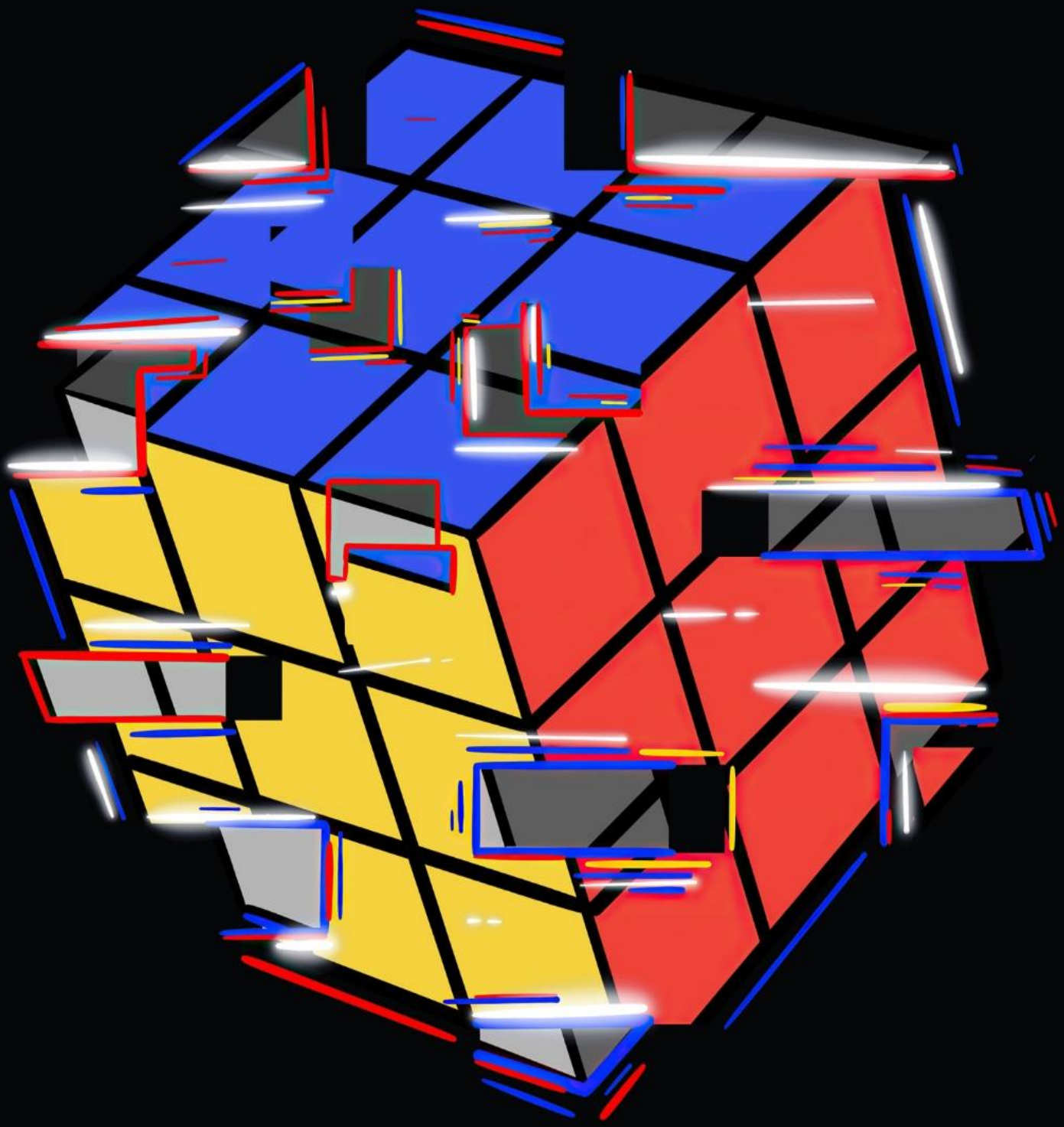
SolviCube

Projet C



Par Cédric YOGANATHAN et Julien STARCK

eFrei PARIS

Le projet que nous avons reçu pour notre deuxième semestre en programmation en Langage C se porte sur la réalisation d'un programme modélisant un Rubik's Cube et permettant de le résoudre et de le manipuler. Ce Rubik's Cube devrait être visible sous forme de patron et implémenter ses différents mouvements pour le résoudre étape par étape. Tout cela sera afficher dans le terminal avec un menu pour choisir ce que vous voulez faire avec le Rubik's Cube. Pour répondre à ces exigences, nous avons créé SolviCube, un programme en langage C qui permet de manipuler le Rubik's Cube 3x3 a votre guise.

- **Les fonctions proposées par SolviCube :**

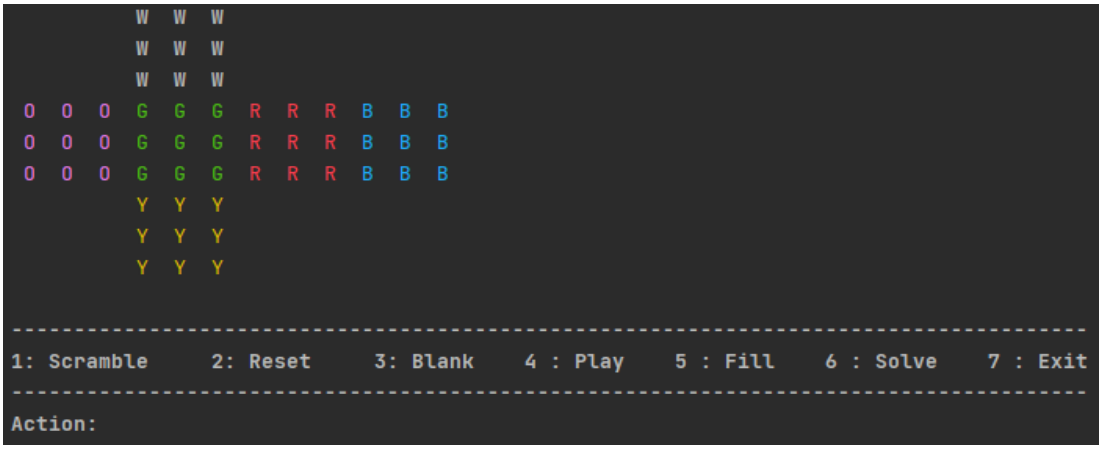
Notre programme est composé de plusieurs fonction qui permette de manipuler le rubik's cube, lors du lancement de celui-ci une modélisation de rubik's cube avec une couleur à chaque face, en dessous, comme demandé dans le cahier des charges, nous avons les différentes fonctionnalités de manipulation du rubik's cube. SolviCube peut donc mélanger le rubik's cube en lui faisant faire des mouvements aléatoires, ensuite si vous avez des difficultés pour résoudre le rubik's cube, vous avez la possibilité de le réinitialiser, il y a aussi la possibilité de vider entièrement le rubik's cube avec des cases sans couleur. Cette fonction ne servirait à rien sans la fonction du remplissage manuel case par case qui respecte les conditions de résolutions d'un rubik's cube, SolviCube dispose évidement de la fonction pour effectuer les rotations de résolution du rubik's cube. Nous avons aussi implémenté une fonction de résolution automatique par l'ordinateur qui applique les algorithmes pour résoudre le rubik's cube. Nous avons la fonction « semi_cross » qui permet de faire une « fleur » de la couleur de la face opposé qui permet de résoudre la première étape du rubik's cube.

SolviCube possède bien toutes les fonctions demandées dans le cahier des charges et permet donc une manipulation et une personnalisation complète du rubik's cube.

- **L'interface graphique :**

L'interface utilisateur a été réalisée comme demander dans le sujet avec son menu, ses options proposer à l'utilisateur ainsi que les sous menu sur certaines d'entre elle, celle-ci est très simple d'utilisation.

Menu principal :



- **Présentation technique de SolviCube :**

SolviCube est composé de 24 fonctions qui sont toutes dans le même fichier, « rubiks.c », mais notre programme est composé deux autres fichiers, le fichier « main.c » qui permet de lancer le programme à partir des fonctions de « rubiks.c » par le biais du fichier « rubiks.h » qui joue le rôle d'intermédiaire entre les deux fichiers. Nous ne pouvons expliquer les 24 fonctions présente dans le programme mais voici les plus importantes pour notre programme

Nous avons tout d'abord la fonction « select_color » qui prend une couleur en paramètres et qui retourne sa valeur associée :

```
Fonction select_color(T_COLOR color):charactère
Debut:
      HANDLE H = GetStdHandle(STD_OUTPUT_HANDLE)
      Selon (color)
            Cas R :
                  Attribue_au_text(H, 12)
                  Retourne 'R'
            Cas B :
                  Attribue_au_text(H, 9)
                  Retourne 'B'
            Cas G :
                  Attribue_au_text(H, 10)
                  Retourne 'G'
            Cas W :
                  Attribue_au_text(H, 15)
                  Retourne 'W'
            Cas Y :
                  Attribue_au_text(H, 14)
                  Retourne 'Y'
            Cas O :
                  Attribue_au_text(H, 13)
                  Retourne 'O'
            Cas LG :
                  Attribue_au_text(H, 7)
                  Retourne 'X'
            Sinon :
                  Retourne ' '
      Fin Selon
Fin
```

Nous avons ensuite la fonction « side_to_index » qui prend un nom de face en paramètre et qui retourne sa position dans le tableau rubiks :

```
Fonction side_to_index(T_SIDE side) : entire
Debut
      Selon (side)
            Cas LEFT:
                  Retourne 0
            Cas UP:
                  Retourne 1
            Cas FRONT:
                  Retourne 2
            Cas DOWN:

            Cas RIGHT:
                  Retourne 4
            Cas BACK:
                  Retourne 5
            Sinon:
                  Retourne 0
      Fin Selon
Fin
```

Nous avons ensuite réalisé la fonction « init_rubiks » qui permet d'initialiser le rubik's cube avec une couleur à chaque face :

```
Fonction init_rubiks(RUBIKS_SIDE* rubikscube)
      T_COLOR val

      Variable locale : i
      i ← 0
      Début
            Pour i  0 A 5 faire
                  Selon (rubikscube[i].typedelaface)
                        Cas LEFT:
                              Val ← 0
                              Sors du selon
                        Cas UP:
                              Val ← W
                              Sors du selon
```

```
                    Cas FRONT:
                         Val ← G
                         Sors du selon
                    Cas DOWN:
                         Val ← Y
                         Sors du selon
                    Cas RIGHT:
                         Val ← R
                         Sors du selon
                    Cas BACK:
                         Val ← B
                         Sors du selon
                    Sinon:
                         Sors du selon
               Fin Selon
          Fin Pour
     Fin
```

« display_rubiks » est la fonction qui permet d'afficher le rubik's cube sous forme de patron a l'écran pour avoir une vision sur chaque face lorsque qu'on veut le manipuler, voici son algorithme :

```
Fonction display_rubiks(*rubikscube)
Paramètres modifiés : T_COLOR display[9][12]
Paramètres copiés : T_SIDE face
Variables locales : i, j, k
Début
     Pour i←0 A 2
          Pour j ←0 A 2
               display[i][3+j]←rubikscube[side_to_index(UP)].face[i][j]
          Pour j←0 A 2

               display[6+i][3+j]←rubikscube[side_to_index(DOWN)].face[i][j]
     Pour i←0 A 3
          Selon i
                    Cas 0:
                         Face = LEFT
                         Sors du selon
                    Cas 1:
                         Face = FRONT
                         Sors du selon
                    Cas 2:
                         Face = RIGHT
                         Sors du selon
                    Cas 3:
                         Face = BACK
                         Sors du selon

                    Sinon:
                         Sors du selon
```

```
            Fin Selon
        Pour j←0 A 2
            Pour k←0 A 2
                display[3+j][i*3+k]←rubikscube[side_to_index(Face)].face[i][j]
            Fin Pour
        Fin Pour



    Pour i←0 A 8
        Pour j←0 A 11
            Afficher(«  %c  », select_color(display[i][j]))
            Attribue_au_text(H,15)
        Afficher(« \n »)
        Fin Pour
    Fin Pour
    Afficher(« \n »)

Fin
```

« move_rubiks » permet de déplacer le rubik's cube de la même façon qu'avec les mains, il suffit de choisir quel mouvement vous voulez faire et le cube s'actualise en temps réel.

```
Fonction move_rubiks(RUBIKS_SIDE* rubikscube)
Variable locale : i, choix
Afficher(« 1Choisir une face 2Rotation horizontale 3Rotation verticale 4Exit »)
Saisir(choix)
Selon (choix)
    Cas 1 :
        Afficher(« 1 Left  2 Front  3 Up  4 Down  5 Right  6 Back 7 exit»)
        Saisir(choix)
        Selon (choix)
            Cas 1:
                Afficher(«1 Quart de tour dans le sens horaire 2 Quart de
                tour dans le sens anti horaire 3 Moitié de tour dans le
                sens horaire de la face choisi 4 Moitié de tour dans le
                sens anti horaire de la face choisi 5 Exit»)
                Saisir(choix)

                Selon (choix)
```

```
                              Cas 1:
                                    Quart de tour dans le sens horaire de la face
                                    choisi
                                    Sors du selon
                              Cas 2:
                                    Quart de tour dans le sens anti horaire de la
                                    face choisi
                                    Sors du selon
                              Cas 3:
                                    Moitié de tour dans le sens horaire de la face
                                    choisi
                                    Sors du selon
                              Cas 4:
                                    Moitié de tour dans le sens anti horaire de la
                                    face choisi
                                    Sors du selon
                              Cas 5:
                                    Revient au menu
                                    Sors du selon
                        Fin selon

                  (Pareil pour les autres cas avec la face differente)

      Cas 2 :
            Rotation horizontale du rubikscube
            Sors du selon
      Cas 3 :
            Rotation verticale du rubikscube
            Sors du selon
      Cas 4 :
            Retour au menu
            Sors du selon

Fin selon  Fin
```

« scramble_rubiks » est une fonction qui mélange le rubik's cube en appliquant des mouvements aléatoires a celui-ci.

```
Fonction scramble_rubiks(RUBIKS_SIDE* rubikscube)

Paramètre modifié : rubikscube

Variable locale: loop, move

Début

Loop ← rand()

Pour i←0 A loop

      Move ← rand()

      Faire

            Move ← move/10

      Tant que (move > 11)

      Selon (move)

            Cas 0:
                  Un tour horaire de la face avant
```

```
            Sors du selon
        Cas 1:
            Un tour antihoraire de la face avant
            Sors du selon
        Cas 2:
            Un tour horaire de la face arrière
            Sors du selon
        Cas 3:
            Un tour antihoraire de la face arrière
            Sors du selon
        Cas 4:
            Un tour horaire de la face gauche
            Sors du selon
        Cas 5:
            Un tour antihoraire de la face gauche
            Sors du selon

        Cas 6:
            Un tour horaire de la face droite
            Sors du selon
        Cas 7:
            Un tour antihoraire de la face droite
            Sors du selon
        Cas 8:
            Un tour horaire de la face haute
            Sors du selon
        Cas 9:
            Un tour antihoraire de la face haute
            Sors du selon
        Cas 10:
            Un tour horaire de la face basse
            Sors du selon

        Cas 11:
            Un tour antihoraire de la face basse
            Sors du selon

    Fin selon

Fin
```

« fill_rubiks » est la fonction qui vous permet de remplir un rubik's cube, vidé préalablement par la fonction « blank_rubiks », case par case avec le choix de la case a remplir sur la face et avec la couleur que vous voulez. Vos placements de case devront respecter les règles de résolution du rubik's cube 3x3 pour être appliqué.

```
Fonction fill_rubiks(RUBIKS_SIDE* rubikscube)

Variable locale: cpt_r, cpt_b,…,cptr_o

                 cpt_centre_r, cpt_centre_b,…, cpt_centre_o

                 cpt_angle_r, cpt_angle_b,…,cpt_angle_o

                 cpt_pole_r, cpt_pole_b,…,cpt_pole_o

                 val

                 face, ligne, colonne, cpt

Début
Pour cpt←0 A 53
    Afficher(« La face : »)
```

```
Saisir(face)
Si face == 7
     Retour au menu
Fin Si
Afficher(La ligne :)
Saisir(Ligne)
Si ligne == 3
     Retour au menu
Fin Si
Afficher(La colonne :)
Saisir(Colonne)
Si Colonne == 3
     Retour au menu
Fin Si
Afficher(La couleur :)
Saisir(Couleur)
Si Couleur == 7
     Retour au menu
Fin Si
Selon(Couleur)
     Cas 1 :
          Si cpt_r >= 9
               Sors de selon
          Fin Si
          Si case choisit à un des pôles des faces
               Si  1 >= cpt_pole_r < 4
                    Si rubikscube[face].face[ligne][colonne] == R
                         Sors de selon
                    Fin Si
               Fin Si
               Si cpt_pole_r >= 4
                    Sors de selon
               Sinon si
                    rubikscube[face].face[ligne][colonne] ← R
                    cpt_pole_r ← cpt_pole_r + 1
                    cpt_r ← cpt_r + 1
               Fin Sinon Si
          Pour l ← 0 A 5
            Pour m ← 0 A 2
              Pour n ← 0 A 2
                Si(l==face)&&(m == ligne)&&(colonne == n)
                 Si (cpt_angle_r >= 1ETcpt_angle_r <4)&&(rubikscube[l].face[m][n]==R)
                      Sors de selon
                 Fin Si
                 Si cpt_angle_r >= 4
                      Sors de selon
                 Fin Si

                 Sinon Si
                      rubikscube[face].face[row][col] ← R
                      cpt_angle_r ← cpt_angle_r + 1
                      cpt_r ← cpt_r + 1
                      Sors de selon
                Fin Sinon Si
               Fin Si
             Fin pour
           Fin pour
          Fin pour


          Pour i←0 A 5
                Si row == 1&&col == 1
                     Si cpt_centre_r >= 1
```

```
                                    Sors de selon
                              Fin Si
                              Sinon Si
                                    rubikscube[face].face[row][col] ← R
                                    cpt_centre_r ← cpt_centre_r + 1
                                    cpt_r ← cpt_r + 1
                              Fin Sinon Si
                        Fin Si
                  Fin Pour

                  Si case adjacente (d'une face à l'autre) de même couleur
                        rubikscube[face].face[ligne][colonne]←sans couleur
                        Sors de selon
                  Fin Si
                  Sors de selon
            Cas 2 :
                 Pareil mais avec autre couleur
            Cas 3 :
                 Pareil mais avec autre couleur
            Cas 4 :
                Pareil mais avec autre couleur
            Cas 5 :
                Pareil mais avec autre couleur
            Cas 6 :
                Pareil mais avec autre couleur

      Affiche le rubikscube


Fin
```

« blanks_rubiks » est une fonction qui vide toutes les cases du rubikscube en lui mettant le caractère 'X' avec la couleur grise.

```
Fonction blanks_rubiks(RUBIKS_SIDE* rubikscube)
Variables locales : i, j, k
Début
      Pour i ← 0 A 5
            Pour j ← 0 A 2
                  Pour j ← 0 A 2
                        Rubikscube[i].face[j][k] ← LG
                  Fin Pour
            Fin Pour
      Fin Pour
Fin
```

La fonction « create_rubiks » permet de créer chaque face du rubik's cube en lui assignant une taille dynamique

```
Fonction create_rubiks()


Debut
      RUBIKS_SIDE* rubikscube ← (RUBIKS_SIDE*) malloc(sizeof(RUBIKS_SIDE)*6
      rubikscube[0].type_face ← LEFT
      rubikscube[1].type_face ← UP
      rubikscube[2].type_face ← FRONT
```

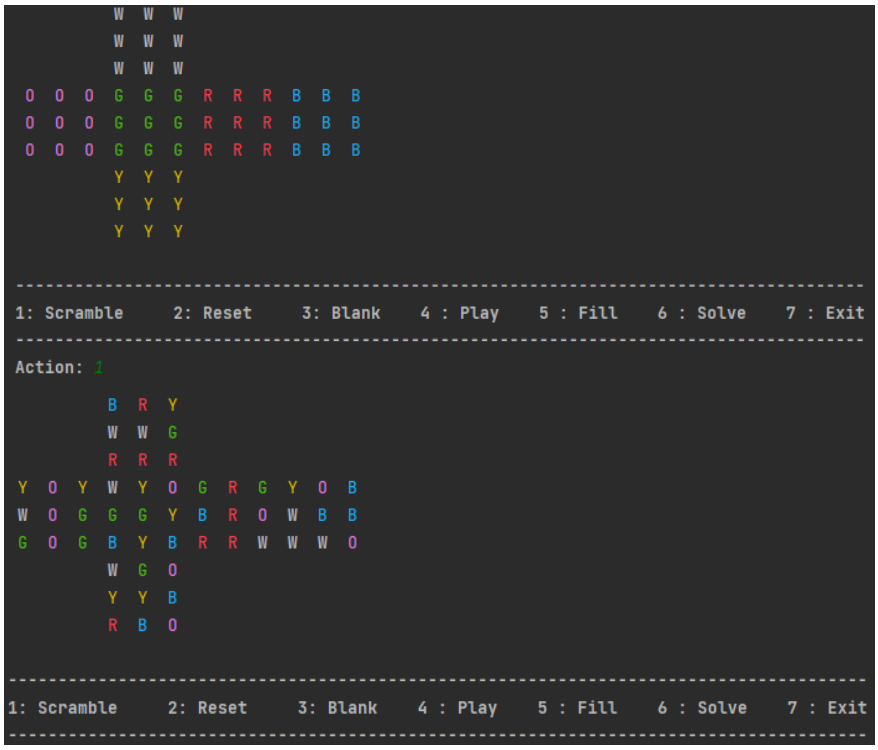
```
        rubikscube[3].type_face ← DOWN
        rubikscube[4].type_face ← RIGHT
        rubikscube[5].type_face ← BACK
        Retourne rubikscube
Fin
```

Ce projet était vraiment compliqué dans sa réalisation, la programmation de la fonction solve_rubiks étais très complexe et nous avons dû beaucoup réfléchir d'une part sur la résolution du rubiks cube en vrai et d'autre part sur comment le représenter dans le programme. Certains bugs ont été découvert au fil de la réalisation du projet, comme des problèmes de mouvement ou de remplissage de case, ceci ont pu être réparer dans les temps.

- **Résultat de notre programme :**

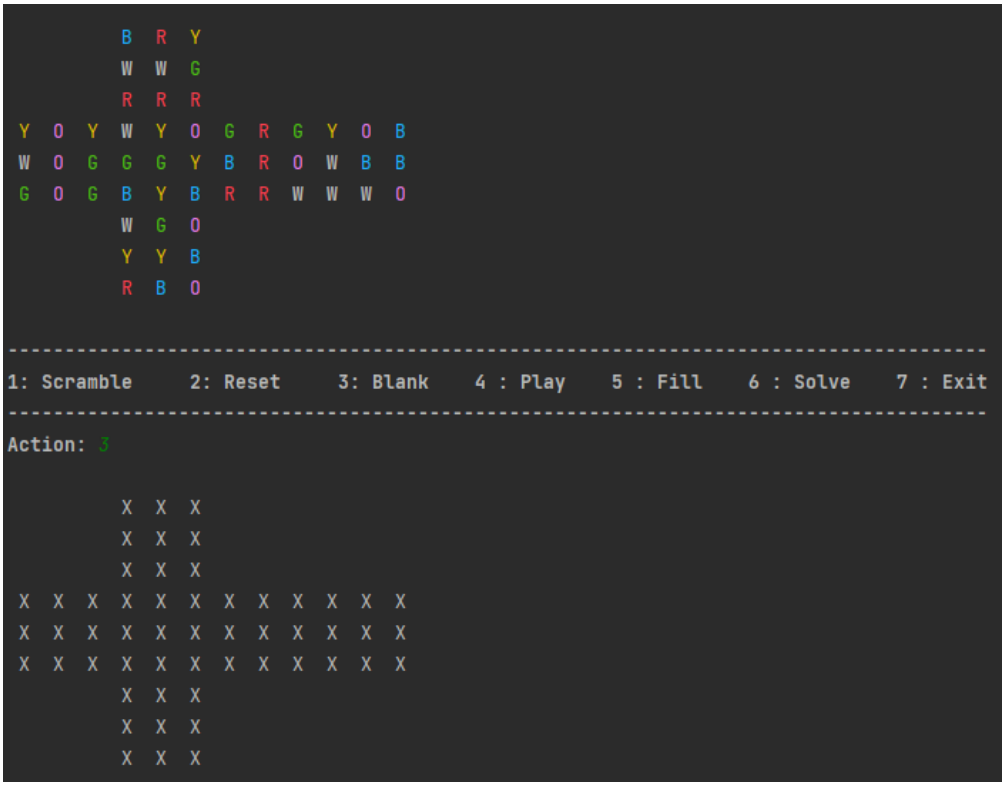
Nous avons réalisé une batterie de test pour vérifier que nos fonctions était correcte, pour cela nous avons simplement utiliser Solvicube pour tout d'abord voir si le mélange était bon. Ensuite nous avons essayer la fonction blank pour vider le cube. Nous avons par la suite rempli une case du cube vide pour voir si cela fonctionnait (vous pouvez aussi essayez de tester des remplissages impossibles comme avoir la même couleur sur 2 faces). Nous avons réussi à résoudre un rubik's cube avec les fonctions de mouvement. Le solveur automatique ne fonctionne malheureusement pas et nous n'avons pas trouver de solution dans les temps.

Utilisation du scramble :

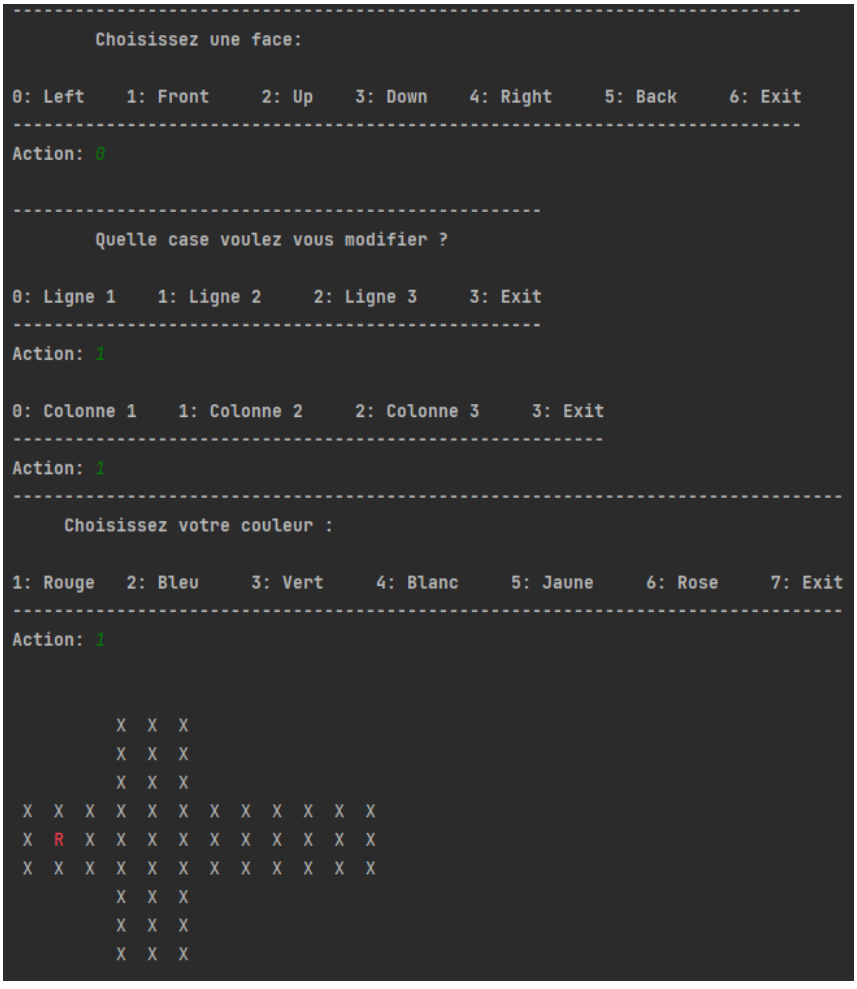


Utilisation de la fonction de mouvement :

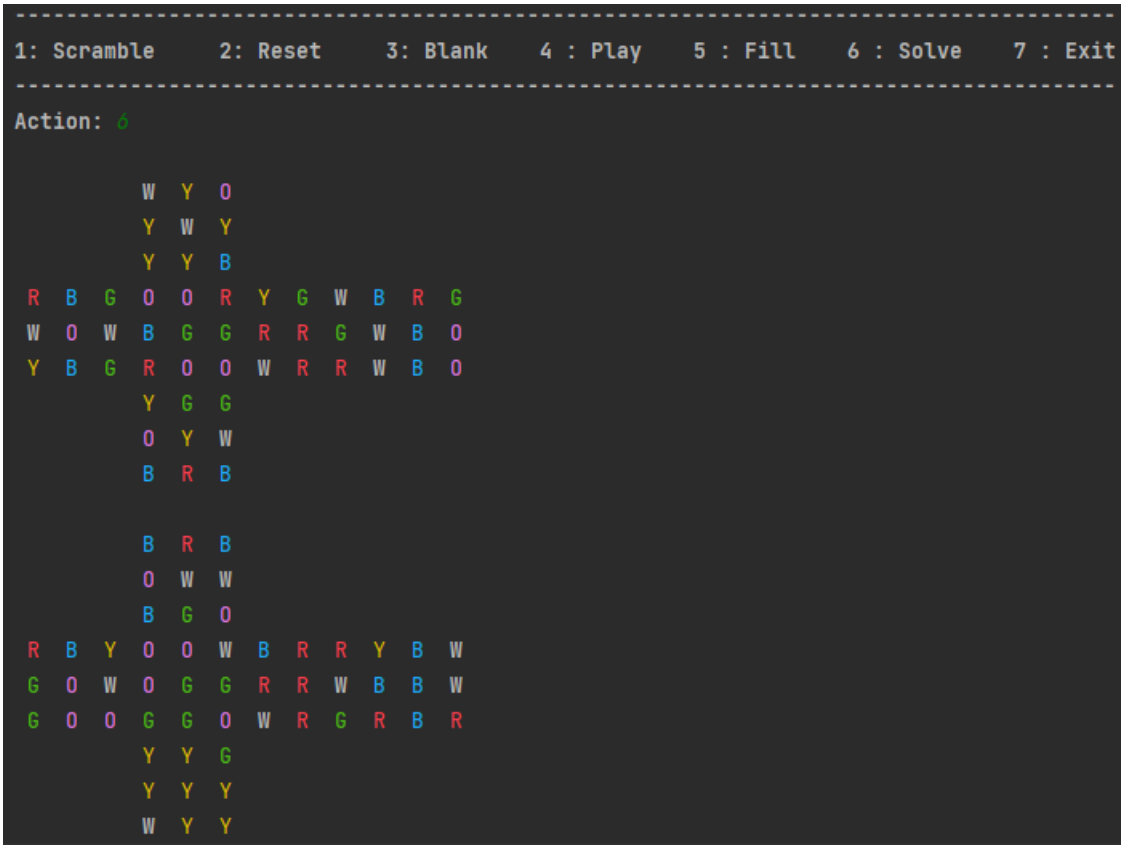
Utilisation de la fonction blank :



Utilisation de la fonction de remplissage (sur une case) :



Utilisation du solve qui permet de faire seulement la croix de résolution du rubik’s cube :



Vous pourrez effectuer vos tests comme ci-dessus, l'interface est intuitive.

- **Conclusion :**

Ce projet nous appris énormément d'une part sur le plan technique, mais aussi sur l'organisation et la communication. Nous avions tous deux jamais résolu de Rubik's Cube, ce qui nous faisait partir avec un manque de connaissances sur le sujet, nous avons dû faire beaucoup de recherches sur internet pour se renseigner sur la résolution d'un rubik's cube. Nous devions tout d'abord comprendre son fonctionnement et savoir le résoudre pour pouvoir le retranscrire en programmation en langage C. Nous avons fait plusieurs réunions pour faire des points sur l'avancement du projet et pouvoir s'expliquer ce que l'on avait fait l'un l'autre. En matière de gestion du temps, nous n'avons pas eu de précipitations, nous étions dans les temps grâce à une bonne communication et entente entre nous. En conclusion ce projet nous a appris à travailler en groupe et à s'organiser pour finir le plus que l'on pouvait faire dans les temps impartis, ainsi que l'acquisition de nouvelles connaissances en langage C et sur le rubik's cube en lui-même. Nous sommes fiers de SolviCube mais il y a la fonction de résolution automatique que nous n'avons pas pu terminer ce qui, avec plus de temps, aurait pu être terminé.