enum Face {U = -2, D = -1, L = 0, B = 1, R = 2. F = 3);

enum RotationDirection {NegDouble = -1, CCW = -1, Zero = 0, CW = 1, Double = 2);

void rotate(Face F, RotationDirection dir)

{}

RotationDirection sideFacesToRotation(Face from, Face to) {

if (from == to)

return RotationDirection.Zero;

int x1; int y1; int x2; int y2;

switch (from) {

case Face.L:

x1 = -1; y1 = 0;

case Face.R:

x1 = 1; y1 = 0;

case Face.F:

x1 = 0; y1 = -1;

case Face.B:

x1 = 0; y1 = 1; }

switch (to) {

case Face.L:

x2 = -1; y2 = 0;

case Face.R:

x2 = 1; y2 = 0;

case Face.F:

x2 = 0; y2 = -1;

case Face.B:

x2 = 0; y2 = 1; }

int dir = (x1 == 0) ? x2/y1 : -y2/x1;

if (dir == 0)

return RotationDirection.Double;

else

return (RotationDirection)dir;

}

Change moves to be applied on a different face by taking the index of the face in the Face enum and +/- 1, mod 4

Standardization: White on U

Red on F

Blue on R

Green on L

Orange on B

Yellow on D

* Cross edge:
  + top layer, oriented right (y == 1 && colorY == white)
    - correct location
      * do nothing
    - else (R/W edge, on edge between W and B centers)
      * rotate side face with edge **(R)**
        + get face and turn it
      * turn top face to bring in correct position **(U’)**
        + get other edge color
        + find face with that color
        + get rotation amount

rotate(Face.U, sideFacesToRotation(edge expected face, edge original face);

* + - * unrotate side face with edge **(R’)**
        + get face and turn it
      * unturn top face to bring back to original position **(U)**
        + do all that stuff again, or save it in vars
  + top layer, oriented wrong (y == 1 && color1 != white) (B/W edge, on edge between W and B centers)
    - rotate side face with edge **(R’)**
      * get face and turn it
    - turn top face to bring in correct position **(U)**
      * from edge expected face to face with nonwhite side of edge
    - unrotate new side face with edge **(F’)** 
      * get new face and turn it
    - unturn top face to bring back to original position **(U’)**
      * do all that stuff again, or save it in vars
  + side layer (edge orientation == XY OR y == 0) (B/W edge on edge between R and G centers, W is by R)
    - rotate side face to bring white edge down **(L)**
      * get face
      * get rotation direction
    - rotate bottom layer to bring edge below correct position **(D2)**
      * rotate(Face.D, sideFacesToRotation(edge expected face, edge original face);
    - bring up original side face **(L’)**
      * remember what it was
    - rotate side face twice to bring up edge **(R2)**
      * know where it is
  + bottom layer, oriented upside down (y == -1 && colorY = white) (R/W edge between Y and B centers, R by B)
    - turn bottom face to be in correct position **(D’)**
      * rotate(Face.B, sideFacesToRotation(edge expected face, edge original face); //double negative, one from (expected->original) and one from bottom face
    - turn correct face twice **(F2)**
  + bottom layer, oriented otherwise (y == -1 && colorY != white) (B/W edge between Y and R centers, W by R)
    - turn bottom face to be in correct position **(D)**
      * rotate(Face.B, sideFacesToRotation(edge expected face, edge original face); //double negative, one from (expected->original) and one from bottom face
    - rotate entire cube so correct face is F **[u]**
    - **F U L’ U’**
* Top layer corners
  + Top layer, oriented correctly
    - Correct location, do nothing