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

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

void rotate(Face F, RotationDirection dir)

{}

RotationDirection getRotationAmount(Face from, Face to)

{

int rotationAmount = ((int)to – (int)from) % 4;

if (rotationAmount == 3)

rotationAmount = -1;

return RotationDirection(rotationAmount);

}

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, getRotationAmount(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, getRotationAmount(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, getRotationAmount(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, getRotationAmount(edge expected face, edge original face); //double negative, one from (expected->original) and one from bottom face
    - apply algorithm to correct face **(morph(F) U morph(L)’ U’)**
* Top layer corners (all operations using the corner at the intersection of U, F, and R)
  + Top layer, oriented correctly
    - Correct location
      * Do nothing
    - Wrong location
      * R’ D’ R
      * U, how many ever times
        + rotationAmount = ((x1-z1)\*z2 - (z1+x1)\*x2)/2 + ((x1\*x2 + z1\*z2)/2)^2
        + x1 = x coord of expected position
        + z1 = z coord of expected position
        + x2 = x coord of corner position
        + z2 = z coord of corner position
      * L D L’
      * Bring back U
  + Top layer, oriented incorrectly
    - R’ D’ R D R’ D’ R or F D F’ D’ F D F’
  + Bottom layer, W pointing sideways
    - Move to correct location
    - F D F’ or R’ D’ R
  + Bottom layer, W pointing down
    - Move to correct location
    - R’ D2 R D R’ D’ R