2X2 RUBIK'S CUBE SOLVER

-BY-
MOHAMMED SHANOUF VALIJAN ANSARI – 2021300004
UTSAV AVAIYA – 2021300005
ALLEN ANDREW – 2021300006
(SE-COMPUTER ENGINEERING BRANCH-A1 BATCH)

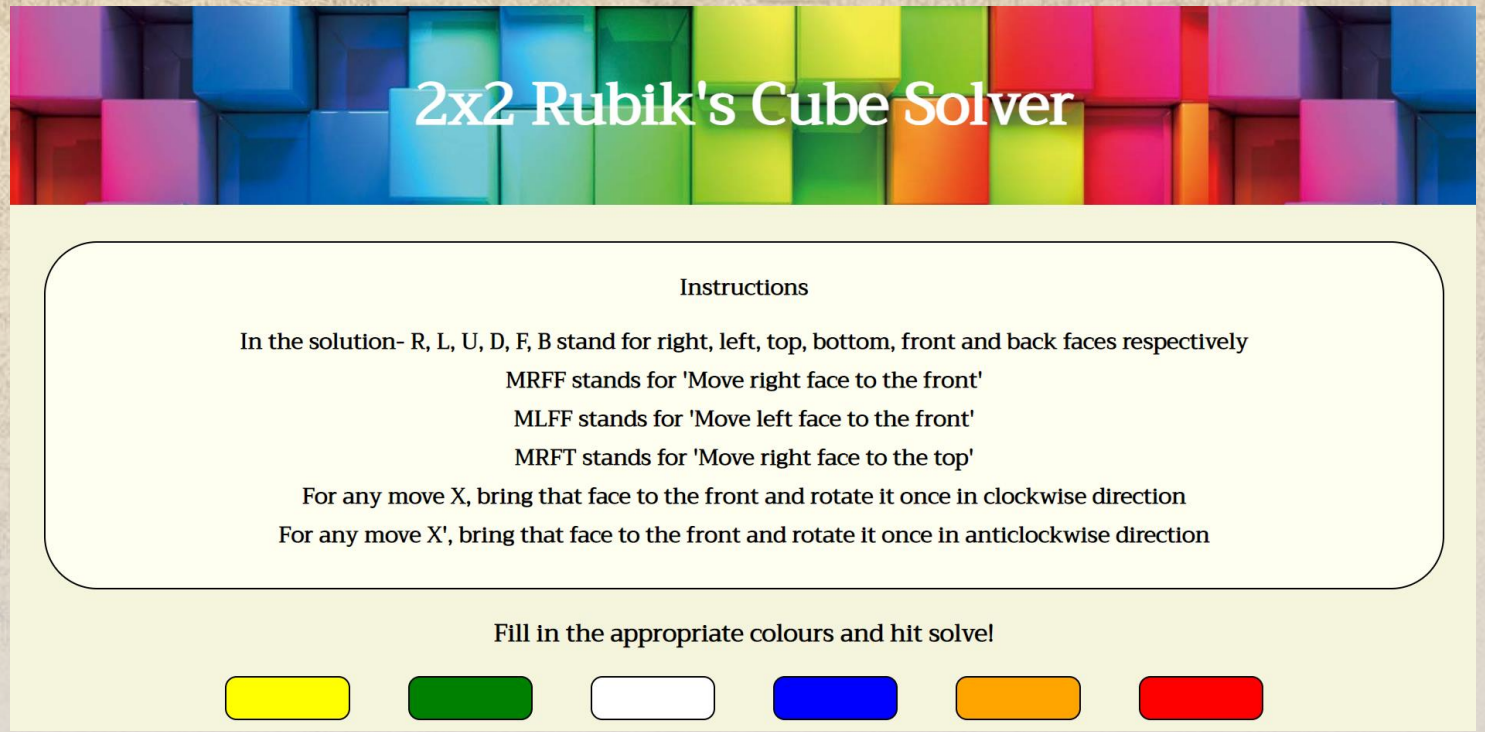
TO BE DISCUSSED-

- Project structure
- Data structure used
- Incorporation of arrays into the project
- Implementation of queues in the project
- Overview of the solution generation methodology
- Algorithms for the various stages
- Example of implementation
- Overview of the code implementation

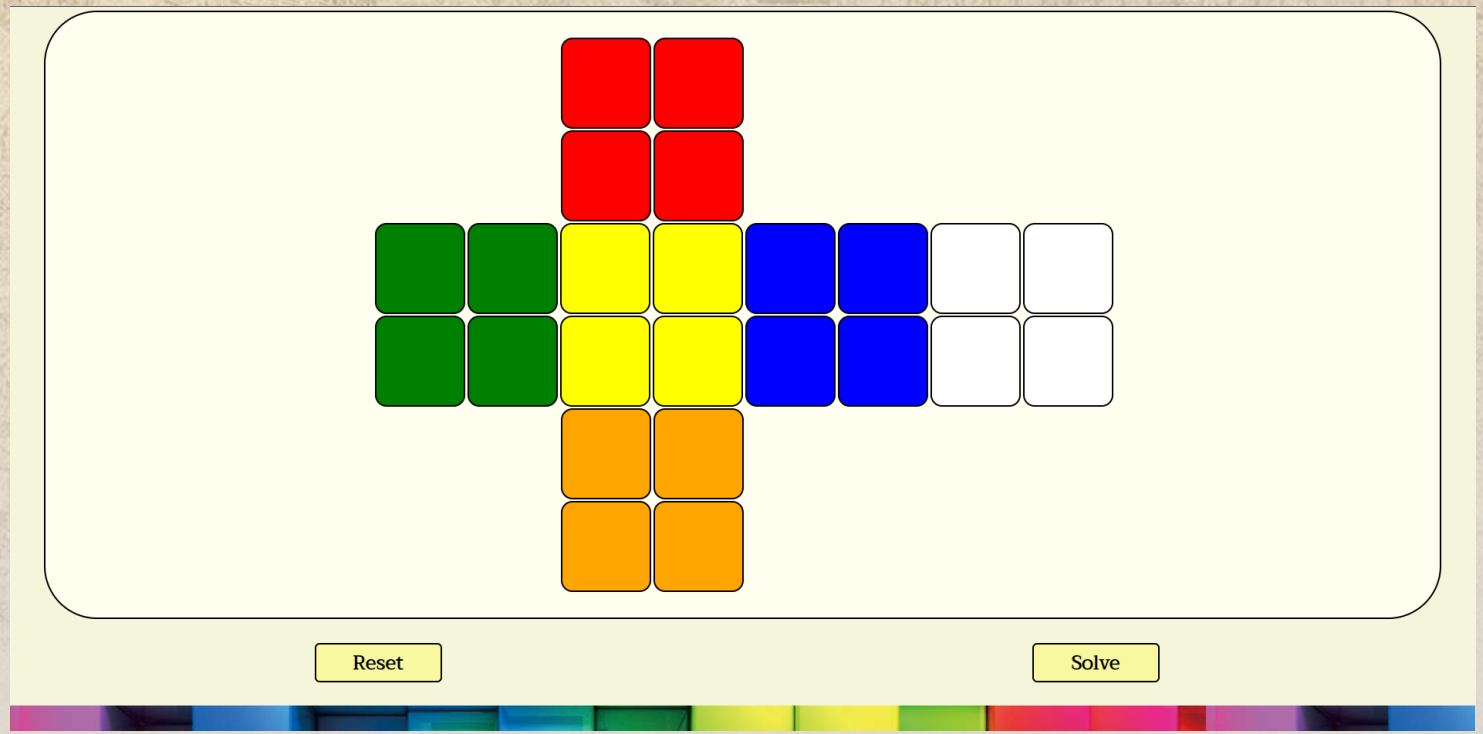
PROJECT STRUCTURE-

- The project was implemented as a web-application
- Hyper text markup language(HTML) was used to provide a skeleton to the website
- Styling was done through the use of cascading style sheets(CSS)
- Pure/Vanilla JavaScript was used for writing the logic of the project
- Event listeners were used to handle the input and output of the program

A GLIMPSE OF THE WEB-APPLICATION-



A GLIMPSE OF THE WEB-APPLICATION-



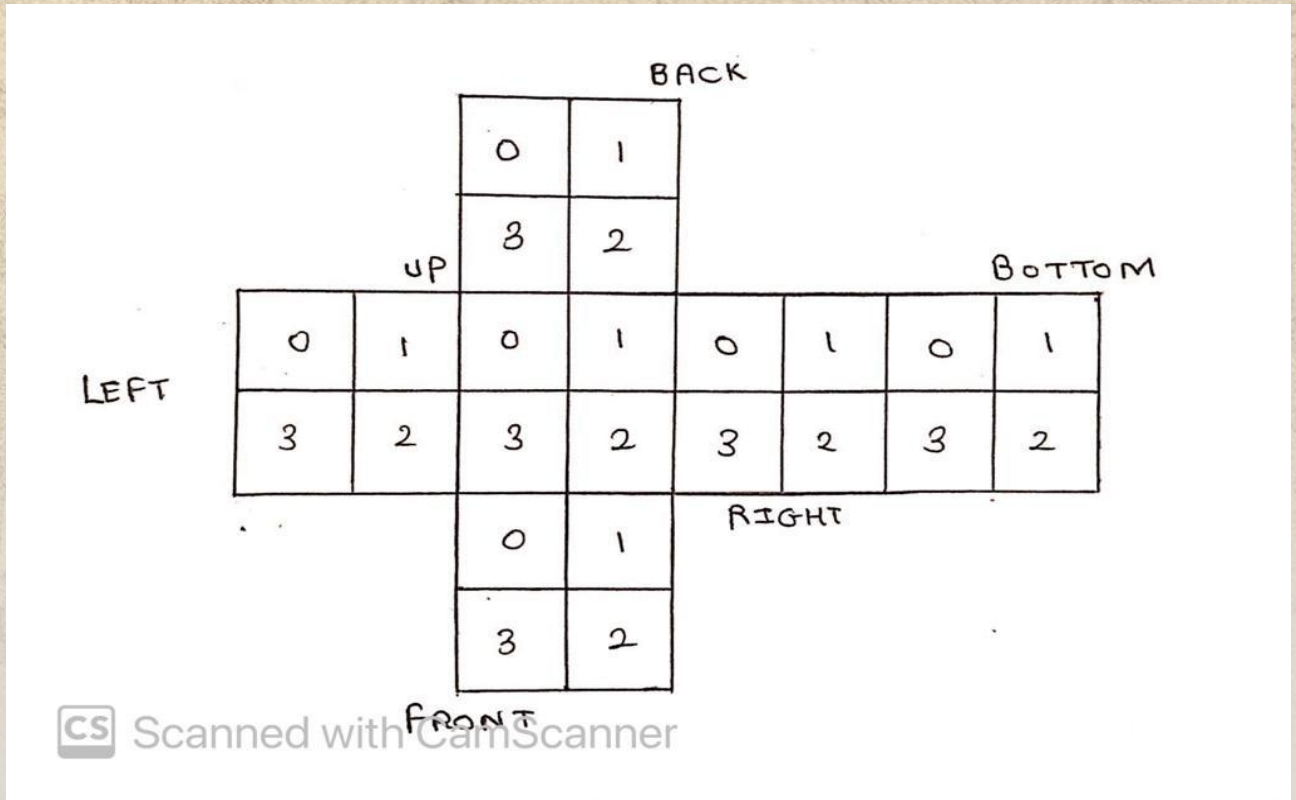
DATA STRUCTURE USED-

- Queue data structure was used for the implementation of the project
- It was used to handle the solution moves of the given scrambled cube input
- As a side note, arrays were used to handle the information of the cube

INCORPORATION OF ARRAYS INTO THE PROJECT-

- Arrays were used to store the colour of each part of each piece of the cube
- Using arrays is beneficial as the data set of the cube is fixed
- In total, 6 arrays- one for each face, were used
- These arrays were updated through out the program so as to reflect the final state of the cube after determination of a solution
- They helped in the confirmation of proper solution

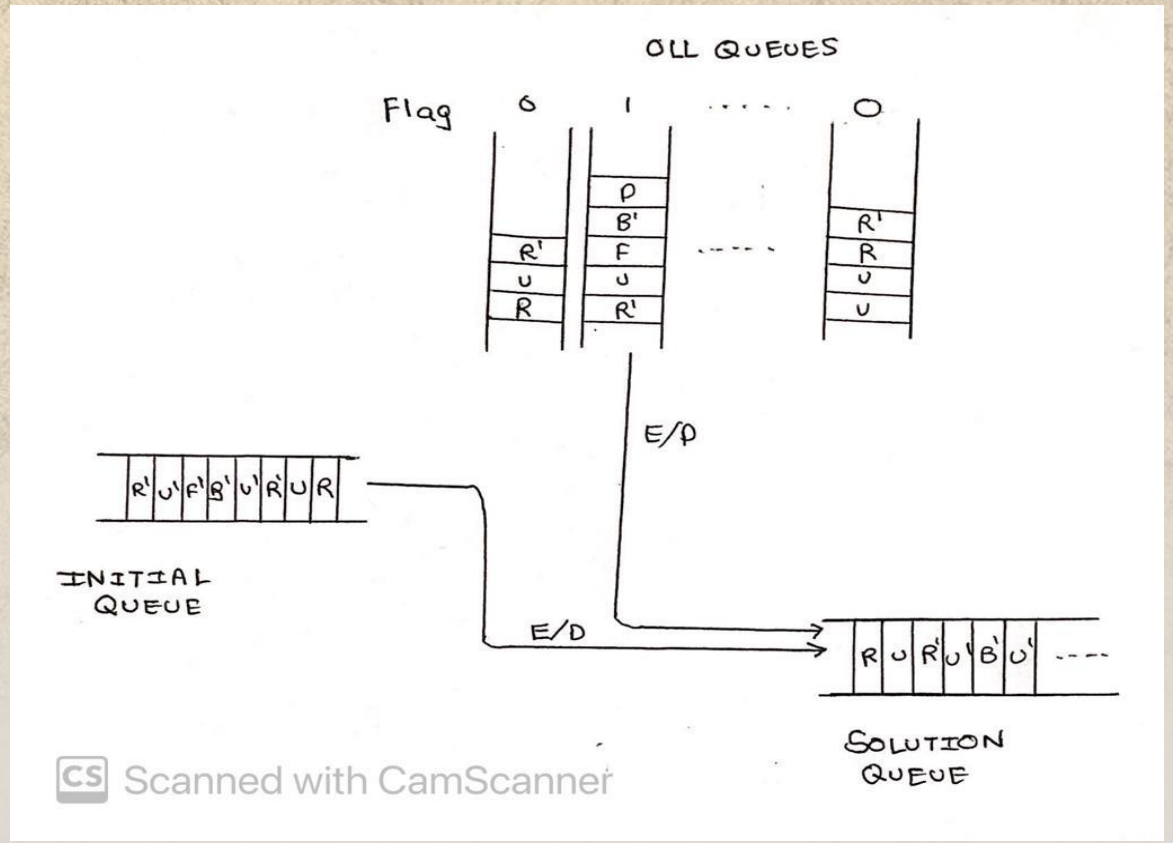
CONCEPTUAL VIEW OF THE ARRAYS USED-



IMPLEMENTATION OF QUEUES IN THE PROJECT-

- Queues were basically used to keep a sequential track of the solution of a particular scrambled cube configuration
- They were mainly used in the second and third stage of solution generation
- First in first out property of queue helps to maintain an order in the set of moves required to solve a particular configuration

CONCEPTUAL VIEW OF THE QUEUES USED-



OVERVIEW OF THE SOLUTION GENERATION METHODOLOGY-

- Primarily, move functions were written so as to imitate the changes in the cube configuration when a particular move is performed on the cube
- Solution was generated in three stages
- Stage-1 dealt with solving the white surface of the cube
- Stage-2 dealt with solving the yellow surface of the cube, also known as orientation of the last layer(OLL)
- Stage-3 dealt with arranging the pieces of both the layers, also known as permutation of both the layers(PBL), so as to reach the solved configuration

ALGORITHM FOR STAGE-1-

- Rotate the base layer till a non-white surface appears at the bottom of the right-front piece of the cube
- Search for a piece which contains white surface and bring it directly above the location mentioned in the previous step
- Insert it in the provided location
- Repeat till the base layer is solved

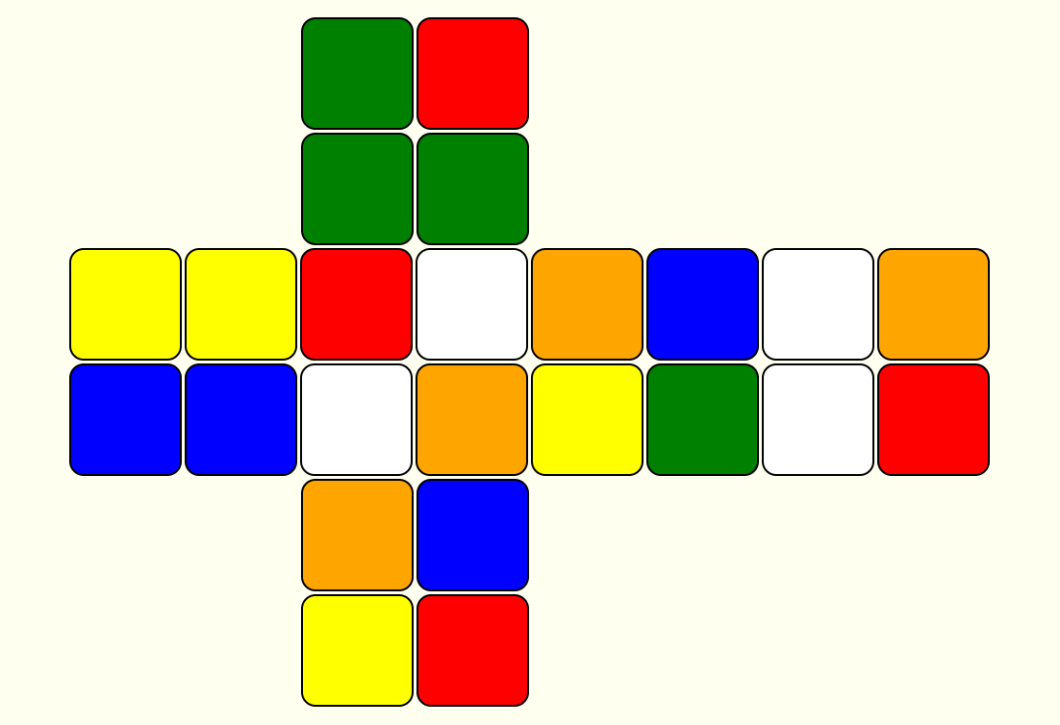
ALGORITHM FOR STAGE-2-

- Check if the yellow surface is solved
- If solved, move to the next stage
- Else, check for the presence of one of the seven possible patterns of yellow surfaces on the cube and perform the required moves respectively

ALGORITHM FOR STAGE-3-

- Check if the cube is already solved
- If solved, end the solution generation process
- Else, check for the presence of one of the six possible configurations for the last stage and execute the moves required
- Repeat till the cube is solved

EXAMPLE OF IMPLEMENTATION(SCRAMBLE)-



EXAMPLE OF IMPLEMENTATION(SOLUTION)-

Step-by-step Solution

Step-1: Solving the base layer

D U R U2 R' U' R U R' D U' R U2 R' U' R U R'

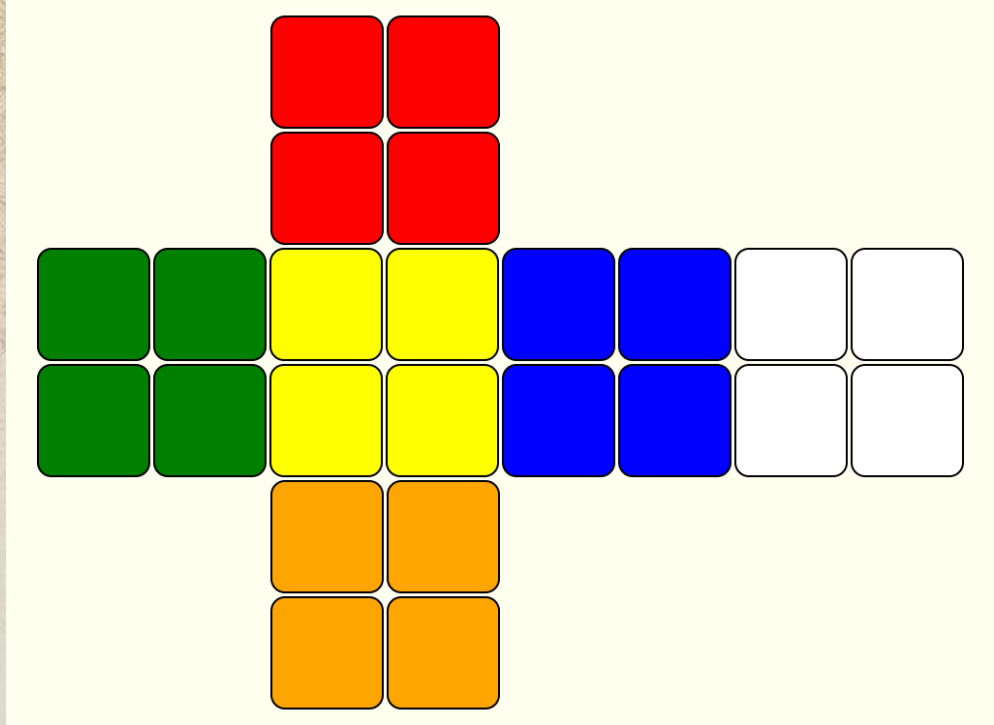
Step-2: Orienting the top layer

MLFF F R' F' R U R U' R'

Step-3: Permuting both the layers

MRFF D R U' R F2 R' U R'

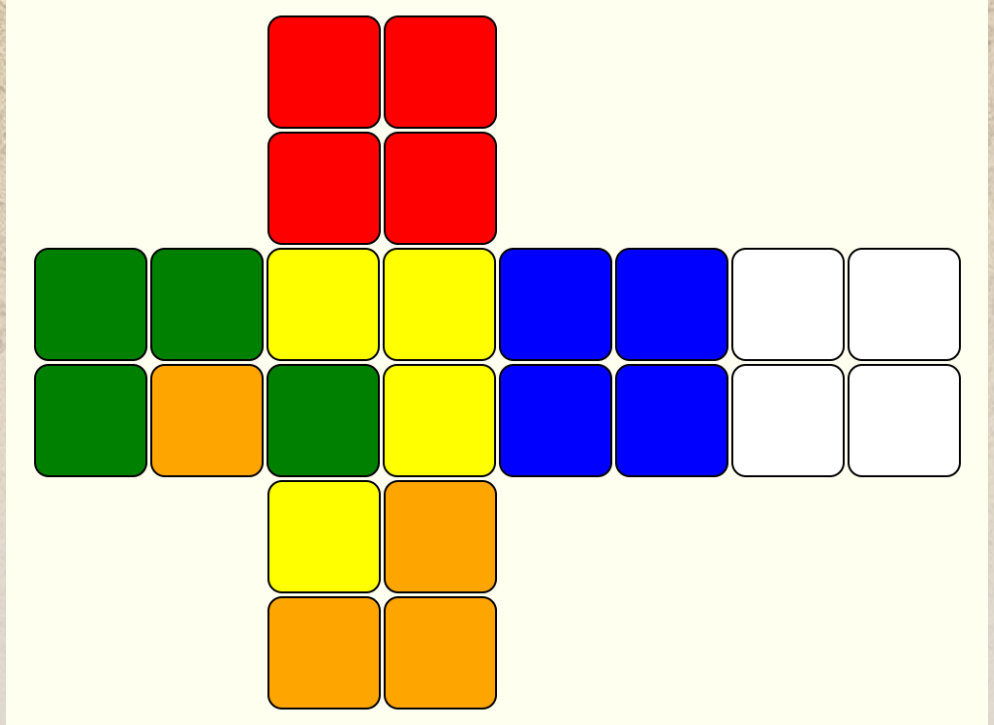
EXAMPLE OF IMPLEMENTATION(SCRAMBLE)-



EXAMPLE OF IMPLEMENTATION(SOLUTION)

Already Solved!!

EXAMPLE OF IMPLEMENTATION(SCRAMBLE)-



EXAMPLE OF IMPLEMENTATION(SOLUTION)

Wrong Inputs!!

OVERVIEW OF THE CODE IMPLEMENTATION-

- Important sections of the HTML file are displayed first
- A glimpse of CSS file is displayed further to explain the designing of the web-application
- Functions that were written for the logic implementation are included as well, written in JavaScript

HTML CODE FOR CREATING THE STRUCTURE OF INSTRUCTION BOX AND 2D MAP OF THE CUBE-

```
<!--instructions section -->
<div id="instructions">
    <div id="instructHead" class="points">
        Instructions
    </div>
    <div id="firstPoint" class="points">
        In the solution- R, L, U, D, F, B stand for right, left, top, bottom, front and back faces respectively
    </div>
    <div class="points">
        MRFF stands for 'Move right face to the front'
    </div>
    <div class="points">
        MLFF stands for 'Move left face to the front'
    </div>
    <div class="points">
        MRFT stands for 'Move right face to the top'
    </div>
    <div class="points">
        For any move X, bring that face to the front and rotate it once in clockwise direction
    </div>
    <div id="lastPoint" class="points">
        For any move X', bring that face to the front and rotate it once in anticlockwise direction
    </div>
</div>
```

```
<!-- 2D map of the rubik's cube -->
<div id="map">
    <div id="layerone">
        <div id="back" class="surfaces">
            <div class="line">
                <div class="boxes"></div>
                <div class="boxes"></div>
            </div>
            <div class="line">
                <div class="boxes"></div>
                <div class="boxes"></div>
            </div>
        </div>
    </div>
    <div id="layertwo">
        <div id="left" class="surfaces">
            <div class="line">
                <div class="boxes"></div>
                <div class="boxes"></div>
            </div>
            <div class="line">
                <div class="boxes"></div>
                <div class="boxes"></div>
            </div>
        </div>
        <div id="top" class="surfaces">
            <div class="line">
                <div class="boxes"></div>
                <div class="boxes"></div>
            </div>
            <div class="line">
                <div class="boxes"></div>
                <div class="boxes"></div>
            </div>
        </div>
```

HTML CODE FOR CREATING THE STRUCTURE OF COLOUR BUTTONS AND SOLUTION BOX-

```
<!-- colour buttons -->
<div id="heading">
    Fill in the appropriate colours and hit solve!
</div>
<div id="colours">
    <div id="yellow" class="colourbtn"></div>
    <div id="green" class="colourbtn"></div>
    <div id="white" class="colourbtn"></div>
    <div id="blue" class="colourbtn"></div>
    <div id="orange" class="colourbtn"></div>
    <div id="red" class="colourbtn"></div>
</div>
```

```
<!-- for displaying the solution -->
<div id="solution">
    <div class="soln">Step-by-step Solution</div>
    <div class="soln marg">Step-1: Solving the base layer</div>
    <div id="step1" class="soln"></div>
    <div class="soln marg">Step-2: Orienting the top layer</div>
    <div id="step2" class="soln"></div>
    <div class="soln marg">Step-3: Permuting both the layers</div>
    <div id="step3" class="soln"></div>
</div>
<div id="solved" class="soln">
    Already Solved!!
</div>
```

CSS CODE FOR DESIGNING THE 2D MAP OF THE CUBE AND SOLUTION BOX-

```css
/* 2D map of the rubik's cube */
.boxes{
    width: 90px;
    height: 90px;
    border: 2px solid black;
    margin: 1px;
    border-radius: 12px;
}

.boxes:hover{
    border: 2px dashed black;
    cursor: pointer;
}

.line{
    display: flex;
}

#map{
    display: flex;
    flex-direction: column;
    border: 2px solid black;
    border-radius: 55px;
    padding-top: 25px;
    padding-bottom: 25px;
    margin-top: 25px;
    margin-bottom: 25px;
    margin-left: 35px;
    margin-right: 35px;
    background-color: ivory;
}

#layertwo{
    display: flex;
    margin-left: 340px;
}
```

```css
/* solution structure */
#solution{
    display: none;
    border: 2px solid black;
    margin-bottom: 23px;
    margin-left: 35px;
    margin-right: 35px;
    background-color: ivory;
    border-radius: 55px;
    padding-top: 25px;
    padding-bottom: 25px;
}

.soln{
    font-size: 23px;
    font-weight: bold;
    display: flex;
    justify-content: center;
    text-align: center;
}

.marg{
    margin-top: 20px;
}

#solved{
    display: none;
    margin-bottom: 22px;
}

/* footer */
#footer{
    height: 38px;
    background-image: url('banner.jpg');
}
```

CREATING A JAVASCRIPT CLASS FOR QUEUE DATA STRUCTURE-

```
//using queue data structure for maintaining and modifying the solution
class queue{
    constructor(solution){
        this.solution=solution;
    }
    isEmpty(){
        return this.solution.length==0;
    }
    enqueue(move){
        this.solution.push(move);
    }
    dequeue(){
        return this.solution.shift();
    }
    tempDisplay(){
        for(var i=0; i<this.solution.length; i++){
            console.log(this.solution[i]);
        }
    }
}
```

MOVE FUNCTIONS OF THE CUBE-

```
function R(value=true){
    temp=back[1],temp2=back[2];
    back[1]=up[1],back[2]=up[2];
    up[1]=front[1],up[2]=front[2];
    front[1]=bottom[3],front[2]=bottom[0];
    bottom[3]=temp,bottom[0]=temp2;
    rotateFaceClockwise(right);
    if(value){
        solution.enqueue('R');
    }
}

function L(value=true){
    temp=front[3],temp2=front[0];
    front[3]=up[3],front[0]=up[0];
    up[3]=back[3],up[0]=back[0];
    back[3]=bottom[1],back[0]=bottom[2];
    bottom[1]=temp,bottom[2]=temp2;
    rotateFaceClockwise(left);
    if(value){
        solution.enqueue('L');
    }
}

function U(value=true){
    temp=back[2],temp2=back[3];
    back[2]=left[1],back[3]=left[2];
    left[1]=front[0],left[2]=front[1];
    front[0]=right[3],front[1]=right[0];
    right[3]=temp,right[0]=temp2;
    rotateFaceClockwise(up);
    if(value){
        solution.enqueue('U');
    }
}
```

```
function D(value=true){
    temp=back[0],temp2=back[1];
    back[0]=right[1],back[1]=right[2];
    right[1]=front[2],right[2]=front[3];
    front[2]=left[3],front[3]=left[0];
    left[3]=temp,left[0]=temp2;
    rotateFaceClockwise(bottom);
    if(value){
        solution.enqueue('D');
    }
}

function F(value=true){
    temp=bottom[2],temp2=bottom[3];
    bottom[2]=right[2],bottom[3]=right[3];
    right[2]=up[2],right[3]=up[3];
    up[2]=left[2],up[3]=left[3];
    left[2]=temp,left[3]=temp2;
    rotateFaceClockwise(front);
    if(value){
        solution.enqueue('F');
    }
}

function B(value=true){
    temp=left[0],temp2=left[1];
    left[0]=up[0],left[1]=up[1];
    up[0]=right[0],up[1]=right[1];
    right[0]=bottom[0],right[1]=bottom[1];
    bottom[0]=temp,bottom[1]=temp2;
    rotateFaceClockwise(back);
    if(value){
        solution.enqueue('B');
    }
}
```

MOVE FUNCTIONS OF THE CUBE-

```
function Rprime(value=true){
    R(false);
    R(false);
    R(false);
    if(value){
        solution.enqueue("R'");
    }
}

function Lprime(value=true){
    L(false);
    L(false);
    L(false);
    if(value){
        solution.enqueue("L'");
    }
}

function Uprime(value=true){
    U(false);
    U(false);
    U(false);
    if(value){
        solution.enqueue("U'");
    }
}
```

```
function Dprime(value=true){
    D(false);
    D(false);
    D(false);
    if(value){
        solution.enqueue("D'");
    }
}

function Fprime(value=true){
    F(false);
    F(false);
    F(false);
    if(value){
        solution.enqueue("F'");
    }
}

function Bprime(value=true){
    B(false);
    B(false);
    B(false);
    if(value){
        solution.enqueue("B'");
    }
}
```

FUNCTIONS FOR SEARCHING AND ORIENTING A PIECE FOR STAGE ONE SOLUTION-

```
function searchPiece(){
    if(up[2]=='W'||front[1]=='W'||right[3]=='W'){
        return;
    }
    else if(up[1]=='W'||back[2]=='W'||right[0]=='W'){
        U();
    }
    else if(up[3]=='W'||left[2]=='W'||front[0]=='W'){
        Uprime();
    }
    else if(up[0]=='W'||back[3]=='W'||left[1]=='W'){
        U();
        U();
    }
    else if(right[1]=='W'||back[1]=='W'){
        Rprime();
        U();
        Bprime();
    }
    else if(front[3]=='W'||left[3]=='W'){
        Lprime();
        Uprime();
        L();
    }
    else if(front[2]=='W'||right[2]=='W'){
        R();
        U();
        Rprime();
        Uprime();
    }
    else if(left[0]=='W'||back[0]=='W'){
        L();
        U();
        U();
        Lprime();
    }
}
```

```
function orientPiece(){
    if(up[2]=='W'){
        R();
        U();
        U();
        Rprime();
        Uprime();
        R();
        U();
        Rprime();
    }
    else if(right[3]=='W'){
        R();
        U();
        Rprime();
    }
    else if(front[1]=='W'){
        Fprime();
        Uprime();
        F();
    }
    rotateBottom();
}
```

FUNCTION FOR STAGE ONE SOLUTION-

```
function solveStage1(){
    loopB=0;
    rotateBottom();
    while(bottom[0]!='W'||bottom[1]!='W'||bottom[2]!='W'||bottom[3]!='W'){
        searchPiece();
        orientPiece();
        loopB++;
        if(loopB==15){
            break;
        }
    }
}
```

FUNCTIONS FOR PATTERN RECOGNITION AND STAGE TWO SOLUTION-

```
function searchPattern(arr1,ind1,arr2,ind2,arr3,ind3,arr4,ind4){
    for(var i=0; i<4; i++){
        if(arr1[ind1]=='Y'&&arr2[ind2]=='Y'&&arr3[ind3]=='Y'&&arr4[ind4]=='Y'){
            return true;
        }
        MRFF();
    }
    return false;
}

function solveStage2(){
    if(searchPattern(up,1,front,0,right,3,left,1)){
        OLL1();
    }
    else if(searchPattern(up,3,front,1,back,3,right,0)){
        OLL2();
    }
    else if(searchPattern(up,1,up,2,back,3,front,0)){
        OLL3();
    }
    else if(searchPattern(up,1,up,2,left,2,left,1)){
        OLL4();
    }
    else if(searchPattern(up,0,up,2,front,0,right,0)){
        OLL5();
    }
    else if(searchPattern(front,0,front,1,back,2,back,3)){
        OLL6();
    }
    else if(searchPattern(left,1,left,2,back,2,front,1)){
        OLL7();
    }
}
```

FUNCTION FOR ORIENTING THE CUBE FOR FINAL PATTERN RECOGNITION-

```
function orientCube(){
    if(back[0]==back[1]&&left[0]==left[3]&&front[3]==front[2]&&right[1]==right[2]){
        isBotBar=true;
    }
    if(back[3]==back[2]&&left[1]==left[2]&&front[0]==front[1]&&right[3]==right[0]){
        isTopBar=true;
    }
    if(isTopBar&&isBotBar){
        return;
    }
    if(isTopBar){
        MRFT();
        MRFT();
    }
    for(var i=0; i<4; i++){
        if(front[0]==front[1]){
            isTopAdj=true;
            break;
        }
        U();
    }
    if(!isTopBar&&!isBotBar){
        for(var i=0; i<4; i++){
            if(front[2]==front[3]){
                isBotAdj=true;
                break;
            }
            Dprime();
        }
    }
    if(isTopAdj&&isBotAdj){
        MRFF();
        MRFF();
    }
}
```

FUNCTION FOR STAGE THREE SOLUTION

```
function solveStage3(){
    loopB=0;
    orientCube();
    while(left[0]!=left[1]||left[1]!=left[2]||left[2]!=left[3]||left[3]!=left[0]||back[0]!=back[1]||back[1]!=back[2]||back[2]!=back[3]||back[3]!=back[0]){
        if(isTopBar&&isBotBar){
            U();
        }
        else if((isTopBar||isBotBar)&&isTopAdj){
            PBL1();
        }
        else if((isTopBar||isBotBar)&&!isTopAdj){
            PBL2();
        }
        else if(!isTopAdj&&!isBotAdj){
            PBL3();
        }
        else if(isTopAdj&&isBotAdj){
            PBL4();
        }
        else if(isTopAdj&&!isBotAdj){
            PBL5();
        }
        else if(!isTopAdj&&isBotAdj){
            PBL6();
        }
        isTopBar=true;
        isBotBar=true;
        loopB++;
        if(loopB==10){
            break;
        }
    }
}
```

THANK YOU