A Report on the '2x2 Rubik's cube solver' project

The '2x2 Rubik's cube solver' mini-project was implemented as a web-application using Hypertext Markup Language(HTML), Cascading Style Sheets(CSS) and JavaScript(JS). Following is the breakdown of the various parts of the project that were encountered during the implementation-

[A] Problem Definition-

The '2x2 Rubik's cube solver', as the name suggests, focusses on taking-in a scrambled configuration of a 2x2 Rubik's cube and providing the user, a set of moves that must be performed on the scrambled cube to bring it back to the solved state.

[B] Scope of Work and Objectives-

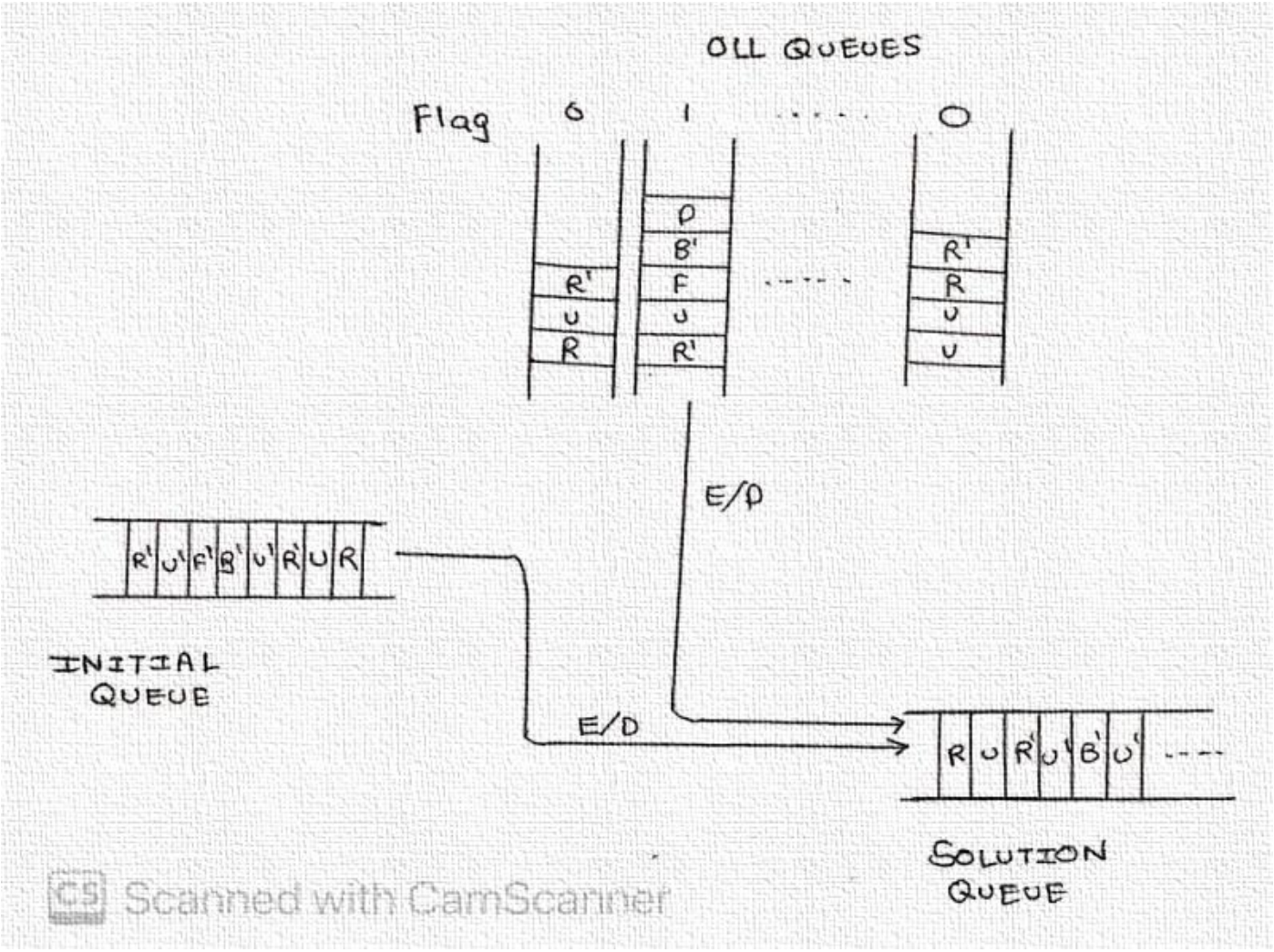
During the implementation of the project idea, there were a set of important functionalities that were needed to be achieved so as to make the web-application user friendly. The work consisted of multiple stages. Building the structure of the website was the first priority followed by its designing. The logic implementation was done at the very end. The use of event listeners in JavaScript played an extremely important role in making the process of input and output dynamic. The most important part of the project implementation was the logic, however, without a proper interface, the efficient user-program communication was not possible. Some of the constraints that we had to deal with included the consideration of various valid and invalid input cases which were needed to be identified first. Another important constraint was to get rid of the redundant moves that would generate as the solution moves are getting executed on the cube and changing its state.

The main objective of the project was to make the user understand how different moves work on a Rubik's cube. Step by step solution was provided so that the user understands the various stages that are encountered while solving a Rubik's cube. Another important objective of the project was to let the user experiment with the 2D map structure of the cube and make respective conclusions on the same.

[C] Data structure used and its usage-

Queue data structure was implemented in the project. The First-in-first-out property of the queue played an important role in proper solution generation of a particular scrambled configuration of the cube.

Given below is a conceptual view of how the queues were implemented in the project-



The 'initial queue' was used as a tracker in which the various solution moves were enqueued at the beginning. 'OLL queues' were used to store the various specific sequence of moves that were used in the intermediate solution stage. Similarly, there were 'PBL queues' used for the final solution stage. During the solution generation, one of these queues is emptied and the moves are transferred to the 'solution queue' right behind the moves of the 'initial queue'. This process is repeated till the cube achieves the solved configuration. Note that OLL stands for 'orienting the last layer' and PBL stands for 'permuting both the layers'. These two terms are very important when it comes to solving a Rubik's cube.

[D] Project plan and Timeline-

Below is the duration of time that was put into the project for working on its various parts-

October 21st - October 23rd
Thinking and finalizing the website structure using HTML and providing an interactive design to it using CSS.

October 25th - October 27th
Trying to develop the logic for the project implementation and deciding on suitable algorithms to use for the same.

November 2nd - November 5th
Implementing the logic in JavaScript.

November 6th - November 8th
Project testing.
(About 25000 scrambles were provided to the program to confirm the proper functioning of the logic).
(About 20-30 random scrambles were manually tested as well).

[E] Implementation Details-

The HTML part of the project consisted of structuring the various components of the website like the instruction box, 2D Rubik's cube map, solution box, solve and reset buttons, etc. Below are some of the structuring codes-

```
<!-- 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>
```

```
<!-- 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>
```

```
<!--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>
```

All these structures were provided with an interactive design with the help of CSS. Below are some of the designing codes that were written-

```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;
}
```

As mentioned earlier, the logic of the project was implemented using JavaScript. The solution generation consisted of three stages which were implemented one after the other. The first stage dealt with solving the white surface of the cube. The second stage dealt with solving the yellow surface of the cube, also known as orientation of the last layer. The final stage was concerned with solving the mismatched edges of the cube, also known as permutation of both the layers. However, to implement all of this, the most important requirement were the move functions.

Move functions are those that replicate the moves that are actually performed on a Rubik's cube in real life. These functions were written with the help of array swaps and rotations. These arrays contained the configurational information of the 2x2 cube. In total, six arrays, each with four memory locations, were used, one for each face.

Below are the move functions as implemented in JavaScript-

```
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');
    }
}
```

These are the non-prime functions. Their prime alternatives were written by performing the respective move functions thrice.

The stage wise solution generation logic is as follows-

[I] Stage one-

Here, as mentioned earlier, the focus is on solving the white surface of the cube. This was basically done through search and fit method. In this method, the cube was searched for a white surface in any orientation. The found piece was then moved to the front and then moved to the base layer.

The functions below were written to implement the said part of the logic-

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

[II] Stage two-

Here, pattern recognition was done to search for one of the seven possible cases of the yellow pieces. Then the required set of moves were performed to solve the yellow surface.

Functions for this part of the logic are as follows-

```
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();
    }
}
```

[III] Stage three-

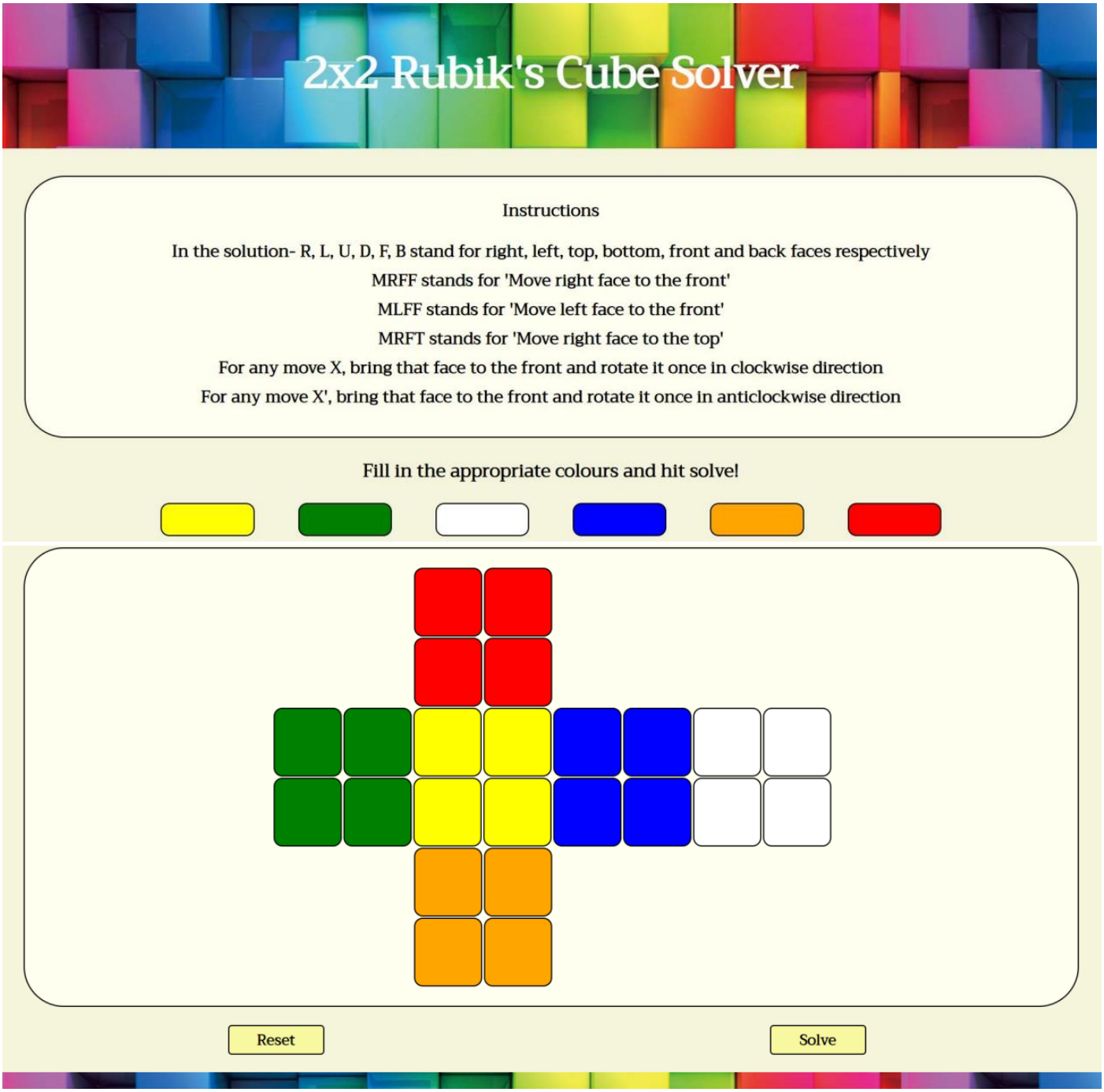
Here as well, a different version of pattern recognition was implemented to identify one of the six possible cases. The required moves were then performed to finally solve the entire cube.

The functions displayed below were written to implement the above logic-

```
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 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
        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;
        }
    }
}
```

After the structuring, designing of the website and the implementation of all the stages, following is a glimpse of the web-application-



[F] Conclusion and Further work-

By implementing this project, we were able to learn a lot of important skills that can help us in our future projects. We got an idea of how the various parts of the code work together to provide a user-friendly program. We got an insight about how a project must be planned for its proper implementation. Moreover, we were able to understand the importance of queue and various other data structures.

Talking about the future scope of the project, there are certain improvements that can be made. Primarily, instead of the 2D map display that was used in this project, a 3-dimensional structure could be displayed which would make the solver more user friendly. Also, instead of writing the sequence of moves in the solution box, the cube could be animated to give the user step-by-step guidance for the solution.

Report by-

Ansari Mohammed Shanouf Valijan, UID – 2021300004
Utsav Avaiya, UID – 2021300005
Allen Andrew, UID – 2021300006

SE BTech Computer Engineering – A(Batch A1)