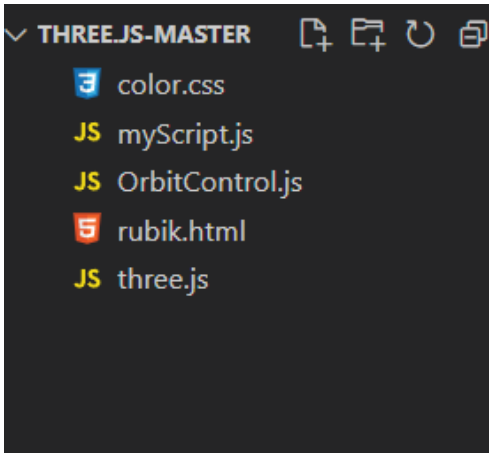
Solución

La solución a este ejercicio consta de varias partes. Lo primero de todo es que necesitaremos usar las librerías de OrbitControl y three.js para poder ejecutar el cubo, las cuales están en local como nos indica el enunciado. También dispondremos de un HTML donde ejecutaremos y llamaremos a los scripts y un archivo CSS.



El HTML **Rubik.html** es el que mostrará la solución, en este llamaremos a los scripts que y al estilo definido en el CSS.

Primero defino la escena, algunas variables que necesitaré a lo largo de la solución, los cubos que formarán el cubo de Rubik y la cámara, indicando en esta la distancia máxima y mínima cuando la creamos.

myScript.js > ...

```
1   //Defino las variables que necesito usar
2   var cube = [];
3   var x=0;
4   var y=0;
5   var z=0;
6   var c=0;
7   var key;
8   var cursorX;
9   var cursorY;
10
11  //Creamos la escena
12  var scene = new THREE.Scene();
13  // Creo el cubo y su tamaño
14  var geometry = new THREE.BoxGeometry( 1, 1, 1 );
15  //
16  const pointer  = new THREE.Vector2();
17  // Para poder seleccionar los cubos seleccionados con el ratón
18  var raycaster = new THREE.Raycaster();
19
20  //Creamos la cámara
21  // Definiendo el campo de visión, el aspect ratio y lo cerca/lejos que se renderiza
22  var camera = new THREE.PerspectiveCamera( 90, window.innerWidth/window.innerHeight, 0.1, 1000 );
23  //Para que no se solapen movemos la cámara un poco
24  camera.position.z = 6;
```

Defino el “renderer” que va a mostrar el cubo, se ha usado WEBGL y también cargo en un array los colores de las caras de los cubos.

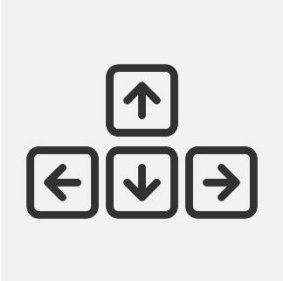
```
//Defino el renderder para poder mostrar el cubo, hemos usado WEBGL
var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
//Control de la camara
var control_camara = new THREE.OrbitControls(camera,renderer.domElement);

//Array con los colores del cubo
let material = [];
material.push(new THREE.MeshBasicMaterial( { color: 'white' }));
material.push(new THREE.MeshBasicMaterial( { color: 'red' }));
material.push(new THREE.MeshBasicMaterial( { color: 'yellow' }));
material.push(new THREE.MeshBasicMaterial( { color: 'green' }));
material.push(new THREE.MeshBasicMaterial( { color: 'blue' }));
material.push(new THREE.MeshBasicMaterial( { color: 'orange' }));
```

Este método es para crear los 27 cubos que forman el cubo de Rubik, moviendo los valores de x, y ,z por cada cubo para que no se solapen.

```
//Aqui creo los 27 cubos del cubo de rubik
for (let f = 0; f < 3; f++) {
    for (let i = 0; i < 9; i++) {
        //Asigno al cubo la posicion
        cube[c] = new THREE.Mesh(geometry, material);
        cube[c].position.x = (x*1.05-1);
        cube[c].position.y = (y*1.05-1);
        cube[c].position.z = (f*1.05-1);
        x+=1;
        //Si llego a estos valores tengo que saltar a otra fila y reinicio
        if(x==3 || x ==6){
            x=0;
            y+=1;
        }
        scene.add(cube[c]);
        c+=1;
    }
    // al mover la z reinicio las variables
    x=0;
    y=0;
}
```

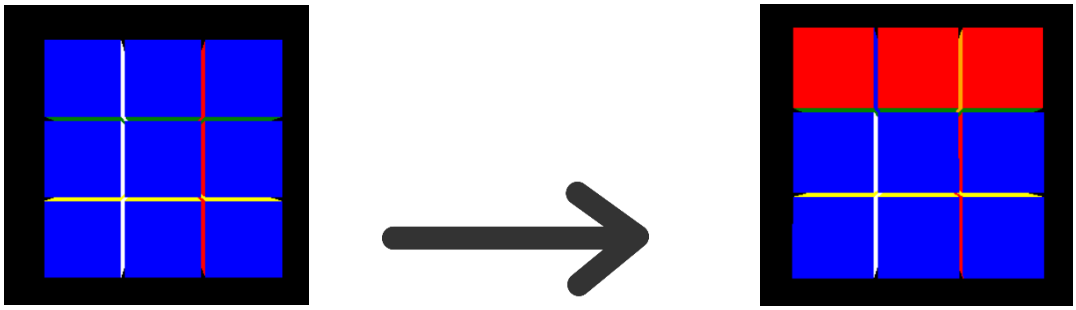
Aquí llamamos a un evento que se activa cuando hacemos **click** con el ratón y se desactiva al ejecutar el movimiento, por lo que para cada movimiento necesitaremos volver a pulsar la tecla correspondiente.

Las teclas que definen el movimiento son las 4 flechas del teclado:



```
const onMouseMove = (event) => {

    // Calculo la posición del ratón
    pointer.x = (event.clientX / window.innerWidth) * 2 - 1;
    pointer.y = -(event.clientY / window.innerHeight) * 2 + 1;

    //Aqui defino una variable con los objetos que intersecciona con mi ratón al hacer click
    raycaster.setFromCamera(pointer, camera);
    var intersects = raycaster.intersectObjects(scene.children);

    //Recorro los cubos y si están en la intersección definida anteriormente los muevo si corresponde
    for (const element of cube) {
        if(element.position.x.toFixed(2) == (intersects[0].object.position.x).toFixed(2) ){
            //Ahora compruba la tecla del movimiento y si corresponde, llamo a la funcion para rotar
            //El angulo a rotar es negativo o positivo para que gire en el lado correcto.
            if(key =='ArrowDown'){
                var puntoRotacion = new THREE.Vector3(0, 0.05, 0.05);
                var axis = new THREE.Vector3(1, 0, 0).normalize();
                rotateAroundPoint(element,puntoRotacion,axis,Math.PI/2)
                }

            if(key =='ArrowUp'){
                var puntoRotacion = new THREE.Vector3(0, 0.05, 0.05);
                var axis = new THREE.Vector3(1, 0, 0).normalize();
                rotateAroundPoint(element,puntoRotacion,axis,Math.PI/-2)
                }
```

Para realizar un movimiento tendremos que seleccionar la dirección del movimiento **pulsando** una flecha y luego haciendo **clic** en la fila o columna a mover



Con **raycaster** seleccionamos al hacer clic, los cubos que hacen una intersección con aquel que he seleccionado para saber cuales de ellos mover, luego comprobamos si el movimiento es horizontal o vertical ,para saber con respecto a que vector movernos, y despues para cada cubo de los seleccionados con **raycaster** llamamos a la función rotateAroundPoint que pasándole el elemento que en nuestro caso es un cubo, el punto, el eje de referencia sobre el que queremos girar y el angulo de la rotación(pi/2 o -pi/2) dependiendo del lado del giro,nos devuelve el elemento con las trasformaciones correspondientes para lo que se considera un movimiento del cubo.

```
// element es el cubo a rotar
// point es el punto de rotacion
function rotateAroundPoint(element, point, axis, angle) {
    // Se establece esta matriz como una m de transformacion:
    var r =  new THREE.Matrix4().makeTranslation(point.x,point.y,point.z);
    //Defino un nuevo quaternion para la rotacion
    //roto con respecto a axis y en un angulo de x radianes
    var q = new THREE.Quaternion();
    q.setFromAxisAngle(axis, angle);
    // Creo una copia del punto y lo invierto
    var copy = new THREE.Vector3().copy(point);
    var n= copy.negate();
    //Creo una matrix de 4x4 como matriz de transformacion con los valores de la variable anterior
    var uMatrix = new THREE.Matrix4().makeTranslation(n.x,n.y,n.z);
    //Establece el componente de rotación de esta matriz en la rotación especificada por el quaterion definido anteriormente
    var matrix2 = new THREE.Matrix4().makeRotationFromQuaternion(q);
    var matrix3 = matrix2.multiply(uMatrix);
    r.multiply(matrix3);
    //Multiplico este cubo por la matriz, y divido por la perspectiva.
    element.applyMatrix4(r);


}
```

Finalmente llamamos a la función de animar, la cual va actualizando todos los valores, tales como la cámara y el propio renderer.

```
var animate = function () {
    //Actualizamos la cámara todo el rato antes de renderizar
    control_camara.update();
    requestAnimationFrame( animate );
    //renderizo la escena
    renderer.render( scene, camera );
    //Actulizo la información del cursor
    document.getElementById("info").innerHTML = "Cursor en: " + cursorX + ", " + cursorY

};

animate();
```