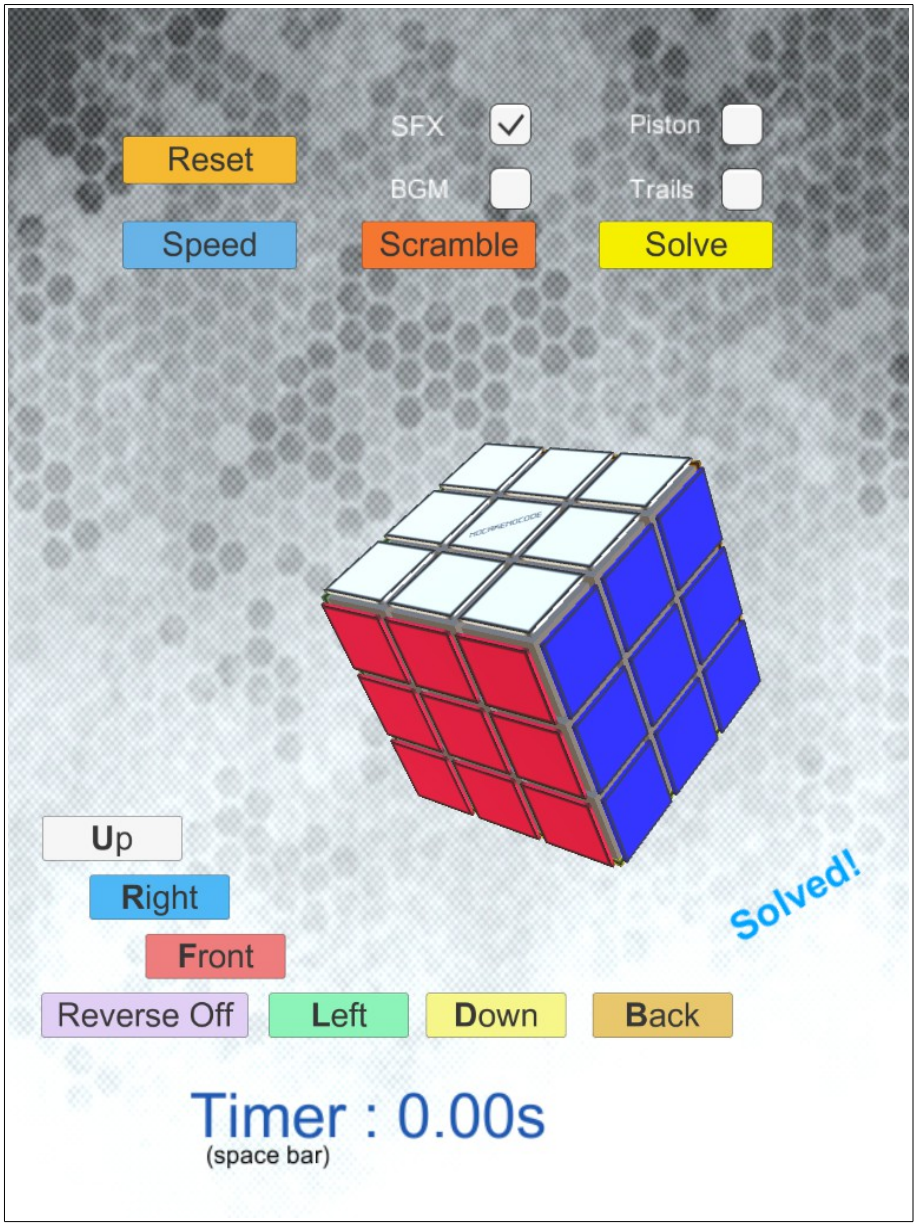
Cube Simulator 3x3x3



By NoCakeNoCode

Written on : Sept 5th 2015 *Last Updated on: Dec 11th 2015*

Welcome to the Cube Simulator starter guide.

This is a short guide where I will be explaining how I made this simulator and the methodology behind it as well some technical things , it is recommended before you start that you have basic to average understanding of Unity and at least average experience in scripting/programming using C# , you should note that this guide will not discuss the source code from scratch but will only highlight major points where a developer will need to know in order to understand the flow of the simulator.

Table of Contents :

I. Cube Construction

Creating the cube is a very simple and straight forward process, you only need to use cubes, 26 of them and place them just like how the rubik cube looks , then you will need tiles which can be again formed from cubes themselves , these tiles are used in order to represent the cell color. You essentially need **3 tiles for each corner piece , 2 tiles for each edge piece , 1 tile for each center piece , you need to make sure you map and parent them correctly to their respective cubes**, you may see the hierarchy in the project at Game scene or prefab for a better understanding. Materials are finally used to paint the cube , simple drag and drop operation , no biggy. :)

After creating all your pieces , remember to give each of them a tag. Ie (Center Piece 1 , Corner Piece 4...etc) , you also need to parent all the cubies under one parent which is Rubik_cube or whatever name you like , that is where you will throw in the CubeController script.

II. 2D Cube logic (Cube2D.cs)

Now you might be asking , why do we need this? Isn't it easier to just use colliders and triggers to detect cubies on some face and then parent them to that face and rotate the face?

This is also correct and can be done , however the main reason we need a 2D Cube Logic is because we might want to add extra functionality in the future , take for example detecting the "Solved" state of the cube , there also might be people who would like to practice solving certain states of the cube , a small convinient script can be written for that too , so it's just simply for future improvements and this is why I will be using it in the project.

There are different approaches that can be used to simulate a 2D cube , one of the easiest approaches is to represent the cube in 2D manner having **three** 3x3 arrays to represent the cubes , as we're using unity it will be easier to simply use tags and in this case we can fill the array with just the tag names respectively.

Next in the code we simulate the 2D cube rotations via swapping , here's a snippet to rotate the red face:

```
==========================================
if (face == 3) { //red
    //rotate corners
    string temp = cube1[0, 0];
    cube1[0, 0] = cube1[2, 0];
    cube1[2, 0] = cube1[2, 2];
    cube1[2, 2] = cube1[0, 2];
    cube1[0, 2] = temp;

    //rotate edges
    temp = cube1[0, 1];
    cube1[0, 1] = cube1[1, 0];
    cube1[1, 0] = cube1[2, 1];
    cube1[2, 1] = cube1[1, 2];
    cube1[1, 2] = temp;
}
==========================================
```

Another thing that is needed which is to give out the selected face tags to the Cube Controller class that will use this information to map cubies to the selected to be rotated face, here's a snippet :

```
==========================================
public string[,] getRedFace(){
        string[,] arr = new string[3,3];
        for(int i = 0;i < 3;i++)
            for(int j = 0;j < 3;j++)
                arr[i,j] = cube1[i,j];
        return arr;
    }
==========================================
```

III. 3D Cube logic (CubeController.cs)

This is the primary class that controls our 3D cube , it uses the 2D cube class to logically and correctly select the correct pieces to be rotated. Here's a list of features it includes:

- Cube face rotation calculation and whole cube rotation.
- Cube scrambling algorithm.
- Trail toggle.
- Keyboard inputs.
- Buttons input.
- Reset Cube.
- Timer.

The **main idea** here is to use the **tags** and then **match the cubies to their to be rotated parent** , as cubies cannot have more than one parent , it is understood that if we place cubies as children to some parent and then rotate that parent then the rest of the cubies will rotate with it , knowing this fact we can proceed and create controls to manage cubies parenting.

Cube face rotation

You can take a look at the rotation() function , inside it we do the whole cube rotation operation as well face rotation , for face rotation we first get the selected face to be rotated via the button or keyboard input , then we use it as a map to our 2D cube's face where it will return to us the previously mentioned method of returning an array of tags for that specific face.

We then proceed to change the parent of the tagged cubies to the selected face , next we rotate the 2D cube (we also rotate it an additional two times if reverse was selected, a little hack but has no visual effect , what comes around goes around :D).

Finally we select the rotation speed from the drop down list combo box and flag start the rotation as well play the sfx sound of rotation , when that happens , the center piece of the selected face will rotate 90 degrees in **factors of 90** speed until the count expires (I'm using factors of 90 to represent the rotation speed as floating point numbers due their nature are prune to precision and marginal errors that can accumulate and cause rotation precision bugs which can make the cube glitchy), then the rotation is complete and variables as well parents are reset for the next rotation.

Whole cube rotation

We simply parent the whole cube pieces to one parent which is Cube , it is where we also attach the script , with this we can control the cube as a whole and at the rotation part we can use the mouse right click drag as an input to rotate the whole cube so other faces are revealed , a code snippet as follows:

```
=========================================
//Mouse Drag Cube Rotation
if (Input.GetMouseButton (1)) {
            float rotationX = Input.GetAxis ("Mouse X") * sensitivityX;
            float rotationY = Input.GetAxis ("Mouse Y") * sensitivityY;
            transform.Rotate (cameraTm.up, -Mathf.Deg2Rad * rotationX * Time.deltaTime *
5000,Space.World);
            transform.Rotate (cameraTm.right, Mathf.Deg2Rad * rotationY * Time.deltaTime *
5000,Space.World);
}
=========================================
```

IV. Cube Scrambling Algorithm

This is a simple aglorithm that populates an arraylist with random faces to be rotated in random directions , currently it rotates 30 random faces, here's a code snippet:

```
=========================================
else if(!rotate && button.Equals("scramble")){
          //trigger scramble mode
          scramble = true;
          //give a number of random faces to be rotated , populate scramble reverse list
          for(int i = 0;i < 30;i++){
               scrambleList.Add(Random.Range(0 , 6));
               scrambleReverseList.Add(Random.Range(0 , 2) == 1 ? true : false);
          }
          scrambleReverseList.Add(reverse); //store current reverse status for later reset
     }
=========================================
```

In our **update mehod** we detect the scramble flag and proceed to flush the next scrambles until over , during this time the player cannot control the cube until it is reset or done scrambling , trails can still be toggled however and rotation speed tweaked along other options.

V. Cube Solving Algorithm

In order to make an AI that solves the cube , there needs to be some data structures to map the cubies and the tiles , this is demonstrated in the Cube2D script , it is where as well the AI solver takes place , the written script simulates real cube cases with pre-written algorithms to tackle each case , for instance by taking a scrambled cube, the following phases are solved on after another (Magic cube style):

- The White Cross.
- The Corner pieces of the 1st layer.
- The Edge pieces of the 2nd layer.
- The Yellow Cross.
- Orientation of the Last Layer (OLL).
- Permutation of the Last Layer (PLL).

IV. Trail Rendering

In order to make face rotation a little bit more eye pleasing , a visual effect is added which is simply adding a trail renderer to each corner piece , this can be customized to your own preference or simply disabled using the UI.

VII. Inputs

==========

Below are the inputs mapped under Controls() method along button inputs with descriptions:

Keyboard Keys/Cube Control UI Buttons (The buttons are colored in respect to the center pieces color of each face)

U / Up	-> Rotate the white face.
D / Down	-> Rotate the yellow face.
R / Right	-> Rotate the blue face.
L / Left	-> Rotate the green face.
F / Front	-> Rotate the red face
B / Back	-> Rotate the orange face.
S / Solve	-> Solves the cube.
Left CTRL / Reverse	-> Changes face rotation from clockwise to anti-clockwise and vice versa.

UI Buttons

Scramble	-> Rotates 30 randomly selected faces clockwise or anti-clockwise.
Reset	-> Resets the cube.
Solve	-> Solves the cube.

Toggles

Trails	-> Enable/Disable Trail Renderer.
BGM	-> Enable/Disable BGM.
SFX	-> Enable/Disable SFX.

Others

Speed	-> Dropdown list with rotation speeds using factors of 90.
Timer	-> Press space bar to start the timer count from 0 , press again to stop counting.
Whole Cube Rotation	-> Hold right click and drag to rotate the cube in the direction you're dragging.

==========

This was all there is regarding this cube simulator, the rest such as sound and canvas as well buttons manipulation can be learned from other tutorials as this guide's objective is to focus on the rubik's cube logic implementation in Unity.

If you feel this guide is missing or needs further explanation regarding something I overlooked , you may send me a message at Fahad@NoCakeNoCode.com and I will be more than happy to take a look at it and answer you back as soon as possible.

==========END==========