

# Twisty Puzzle Masterz 2.0

## (Project Notes)

Idea Conceived: 05/05/2023

- ☐ Update the project to API 33 and AndroidX
- ☐ Add detailed comments that explain all magic numbers and algorithms.
- ☐ Use proper code style guide for java ( reference software engineering document )
- ☐ Make Rayfuzu Learning YouTube video
- ☐ Merge the opencv code into this project's code base. Keep the 2D cube interface code as a debugging option that can be changed in settings.
- ☐ Try to merge my code and the opencv code into the 2.0 repository
  - or try merging the opencv code into my project ?
  - Create a copy to test the merging?
- ☐ Create design diagrams and descriptions on exactly how the data structure and algorithms work.
- ☐ Follow S.O.L.I.D. principles
- ☐ How to make a proper Java library for the solver feature
- ☐

### **Working Java Android Cube Solver and OpenCV**

<https://github.com/ucchiee/AndroidRubikCubeSolver>

### **How to add OpenCV to android:**

- <https://www.youtube.com/watch?v=olk2hTPxFqs&t=255s>
  - At 6:28 whatever you name that module, open that folder in file explorer and paste everything from C:\Users\downs\Desktop\OpenCV-android-sdk\sdk into it
- **Update build.gradle for (:opencv)** add this line
  - `android {`
  - `namespace 'org.opencv'`
- **Update build.gradle for (:opencv)** with 4 fields to match your project build.gradle

- a) compileSdkVersion
  - b) buildToolsVersion
  - c) minSdkVersion and
  - d) targetSdkVersion.
- Comment out each line that is throwing an error inside of **AsyncServiceHelper.java** (I don't use that anywhere)
- <https://www.youtube.com/watch?v=bR7IL886-uc&t=221s>
- Also helpful
  - <https://stackoverflow.com/questions/63254458/could-not-import-the-opencv-library-in-android-studio> (last post in this stackoverflow thread )
  - <https://www.geeksforgeeks.org/how-to-add-opencv-library-into-android-application-using-android-studio/>
  - <https://www.geeksforgeeks.org/different-ways-to-delete-a-module-in-android-studio/>

Video Format Inspiration:

[https://www.youtube.com/watch?v=fAX27\\_FyU9g](https://www.youtube.com/watch?v=fAX27_FyU9g)

# How “*Twisty Puzzle Masterz*” solves the 3x3x3 Rubik’s Cube

## 1. Terminology

- a. Location:
  - i. All interesting layers for the given cubie
- b. Orientation
  - i. Placement of the stickers on the given cubie with respect to their correct center piece colors.

## 2. Cross

### a. Solve Order

- i. {  
[WHITE, GREEN] , [WHITE, BLUE] , [WHITE, ORANGE]  
[WHITE, RED]  
}

### b. Steps

- i. Get all cross pieces on down layer
- ii. Rotate bottom layer until at least 2 pieces are in correct location
- iii. Set the cubies that are in correct locations to their correct orientation
- iv. Iff 2 cubies are in incorrect locations then swap them
  - 1. Set those 2 cubies to correct orientation Iff need to.

### c. Step 1

- i. Get all cross pieces on down layer

### d. Avoidance Maneuver

This is done by rotating the down layer such that there is no white cubie on the spot that intersects with the down and chosen layer to move.

// Chosen layer to move

$n = [L \text{ or } R]$

- D (until no white edge cubie interests  $[n, D]$  )
- $n$  or  $n'$

- **IMPROVEMENT 1:** *At this point you already know which white edges have been solved so you don't need to (do while) rotate the down layer until you are above an unsolved piece.*
- *Instead, you can just keep track of those 4 pieces and which ones are solved and unsolved. That way you can know exactly what movements are needed to put the unsolved white edge piece under your solving cubie.*
- **IMPROVEMENT 2:** *If you have to do D D D then undo those steps and do a D'*

**e. If cubie on up layer**

$[top, right] \Rightarrow R2$

$[top, front] \Rightarrow F2$

$[top, left] \Rightarrow L2$

$[top, back] \Rightarrow B2$

If there is a white edge cubie already located at  $[n, D]$  then use the **avoidance maneuver** to solve the cubie in the up layer.

**f. If cubie on middle layer**

$[left, front] \Rightarrow L \text{ or } F'$

$[right, front] \Rightarrow R' \text{ or } F$

$[left, back] \Rightarrow L' \text{ or } B$

$[right, back] \Rightarrow R \text{ or } B'$

Prefer the move that puts the cubie in the correct location.

( or )

Move that won't kick out an already positioned white cubie on the down layer.

g. **If cubie on down layer**

- i. Go to the next white edge cubie to solve. This one is already on the desired layer.
- ii. Maintain location on down layer while putting other cubies on down layer using the **avoidance maneuver**.

h. **Step 2**

- i. Rotate bottom layer until at least 2 pieces are in correct location

i. **Step 3**

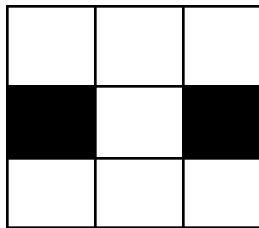
- i. Set all cubies to their correct orientations.
  - ii. If solving cubie is on **Right**
    1.  $R^2 U F R' F'$
  - iii. If solving cubie is on **Front**
    1.  $F^2 U L F' L'$
  - iv. If solving cubie is on **Left**
    1.  $L^2 U B L' B'$
  - v. If solving cubie is on **Back**
    1.  $B^2 U R B' R'$
2. **IMPROVEMENT 3:** Add  $X_w$  and  $X_w'$  to data structure possible rotations
3. If solving cubie is on **Right**
    - a. Execute:  $R U_w' R U_w$
  - b. If solving cubie is on **Front**
    - i. Execute:  $F U_w' F U_w$
  4. If solving cubie is on **Left**
    - a. Execute:  $L U_w' L U_w$
  5. If solving cubie is on **Back**
    - a. Execute:  $B U_w' B U_w$

j. Step 4

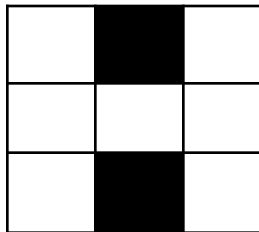
- i. Iff 2 cubies are in incorrect locations then swap them

**NOTE:** Black squares on the below diagrams represents a white cubie on the down layer.

1.



2.



- ii. Case 1:

R2 L2 U2 R2 L2

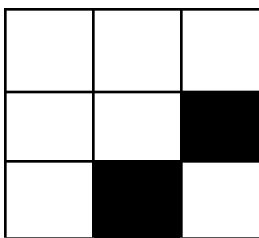
- iii. Case 2:

F2 B2 U2 F2 B2

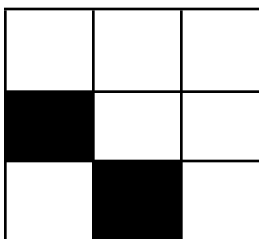
1. **IMPROVEMENT 4:** Improve the rubik's cube solution code and data structure to be able to handle whole cube rotations.
2. **IMPROVEMENT 5:** The above two cases can be swapped by rotating the up layer to match the alignment of 2, then executing the following:

M2, U2, M2

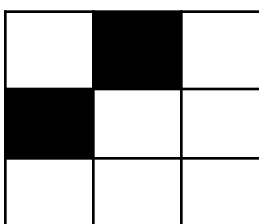
3.



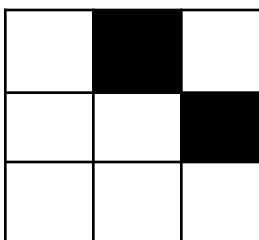
4.



5.



6.

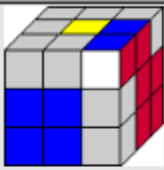
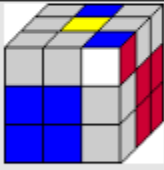
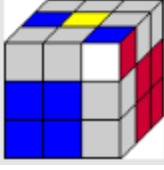


7. To swap the white cubies in cases 3-6 above, use these four algorithms respectively:

- |                      |                      |
|----------------------|----------------------|
| a. R2, U, F2, U', R2 | // [ [D,F] , [D,R] ] |
| b. F2, U, L2, U', F2 | // [ [D,F] , [D,L] ] |
| c. L2, U, B2, U', L2 | // [ [B,D] , [D,L] ] |
| d. B2, U, R2, U', B2 | // [ [B,D] , [D,R] ] |

- iv. **IMPROVEMENT 6:** If you have setup 3 above and they both need to be swapped and oriented, then you can use a single algorithm:
1.  $F' R F R^2$

### 3. F2L

Corner on top, FL color facing side, edge colors match		
Case #	Case Image	Algorithm
1		$U (R U' R')$ $R' F R F'$
3		$U' R U R' U^2 (R U' R')$
5		$U' R U^2 R' U^2 (R U' R')$

a. F2L Configuration Format:

```

“Case Number”: {
    <Cubie1>
    , <Cubie2>
    , <Solution Algorithm>
}

```

```

“Case 1”: {
    “Cubie 1”: {
        // sticker_color = surface_color, ...
        “Orientation”: {“RED”=”RED”, “BLUE”=”YELLOW”, “WHITE”=”BLUE”},
        “Location”: “F R U”
    }
    “Cubie 2”: {
        “Orientation”: {“RED”=”RED”, “BLUE”=”YELLOW”},
        “Location”: “R U”
    }
}

```



```

    }
    "Solution Algorithm": "R' F R F"
}

// end case 1

...

}

```

Cube Orientation = { {F->BLUE}, {U->YELLOW}, {R->RED} }

- b. **Edge Cases** ( that require a specific correction algorithm )
  - i. White stickered Cubie is in the D layer in the wrong location.
  - ii. White stickered Cubie is in the D layer in the correct location with wrong orientation.
  - iii. White stickered Cubie is in the D layer in the correct location and correct orientation.
  - iv. F2L edge cubie is in the wrong location in the middle layer.
- c. **F2L Solution Steps:**
  - i. Locate both cubies
  - ii. If one or both of the F2L cubies are slotted in the wrong corner, then remove them from that corner. (*removal maneuver* )
  - iii. If one or both of the F2L cubies are in the U layer, then rotate the top layer to correctly align the F2L pair.
  - iv. Execute the corresponding algorithm
    1. **Improvement 7:** Add algorithms for each F2L pair to be executed from any of the four cube orientations.

#### 4. Pseudo-Code:

```

// Pairs to solve, in order
{ [ORANGE, WHITE, BLUE], [ORANGE, BLUE] }
{ [RED, WHITE, BLUE], [RED, BLUE] }
{ [RED, WHITE, GREEN], [RED, GREEN] }
{ [ORANGE, WHITE, GREEN], [ORANGE, GREEN] }

List<SurfaceName> locationOfCubie1 = findLocationOfCubie( cubie1 );
List<SurfaceName> locationOfCubie2 = findLocationOfCubie( cubie2 );

```

- 5. OLL
- 6. PLL

Other Helpful Resources

1. You can use this website to generate an animation of the execution of the solution algorithm for those who don't know the rubik's cube notations:
  - a. <https://ruwix.com/widget/3d/> [ solution algorithm ]
  - b. This is helpful if my solution algorithm ends up being 80-100 movements
2. Working python OpenCV (using open source solver)
  - a. <https://github.com/nicpatel963/CubeSolvingScript/blob/master/cubenew.py>
3. Working Android OpenCV and Solver (But using USB cameras? )
  - a. <https://github.com/geoffreywwang/CubeBot>
4. <https://www.youtube.com/watch?v=afAGtExoiLQ>
5. [https://www.youtube.com/watch?v=RMo\\_CLi1Z5g](https://www.youtube.com/watch?v=RMo_CLi1Z5g)
6. <https://www.youtube.com/watch?v=CWmKHcx1X6A>
7. <https://www.youtube.com/watch?v=3pqp6SMmtS4>
8. 9 years old. Can't build into Android Project without Gradle.
  - a. <https://sgelb.github.io/projects/arcs>
9. 3D cube animation
  - a. <https://github.com/cjurjiu/AnimCubeAndroid>
  - b.