Rubik’s Cube Solver

Kaja Bordenave

Candidate ID: 1277

Centre ID: 42241

# Contents

[Contents 2](#_Toc153437899)

[Description 6](#_Toc153437900)

[Analysis – TO BE FINISHED BY 15/9/23 8](#_Toc153437901)

[Analysis/Research Plan 8](#_Toc153437902)

[Existing Systems 8](#_Toc153437903)

[ruwix.com 8](#_Toc153437904)

[rubiks-cube-solver.com 9](#_Toc153437905)

[Type of Application 10](#_Toc153437906)

[How to represent the cube in code 10](#_Toc153437907)

[Class diagram 11](#_Toc153437908)

[Which Algorithm to Solve? 11](#_Toc153437909)

[Kociemba’s Algorithm 11](#_Toc153437910)

[How does the algorithm work? 12](#_Toc153437911)

[Scrambling 12](#_Toc153437912)

[Graphics 12](#_Toc153437913)

[UI 13](#_Toc153437914)

[Moves 13](#_Toc153437915)

[Dialogue with the end-user 13](#_Toc153437916)

[Interview 13](#_Toc153437917)

[Analysis of user ideas 14](#_Toc153437918)

[User requirements 15](#_Toc153437919)

[Objectives 16](#_Toc153437920)

[Design – TO BE FINISHED BY 13/10/23 17](#_Toc153437921)

[Overall System Design 17](#_Toc153437922)

[Underlying structure 17](#_Toc153437923)

[Moves and Rotations 18](#_Toc153437924)

[UI 18](#_Toc153437925)

[Designing the UI 19](#_Toc153437926)

[Explanation of the UI design 21](#_Toc153437927)

[Palette 21](#_Toc153437928)

[Algorithms 21](#_Toc153437929)

[Resetting the cube (meets objective 7) 21](#_Toc153437930)

[CheckSolvable 22](#_Toc153437931)

[CheckSolved 24](#_Toc153437932)

[Scrambling Algorithm 24](#_Toc153437933)

[Solving Algorithm (Kociemba’s Algorithm) 25](#_Toc153437934)

[Implementation – TO BE FINISHED BY 17/11/23 26](#_Toc153437935)

[Face 26](#_Toc153437936)

[Facelet 26](#_Toc153437937)

[Cubie 26](#_Toc153437938)

[Property CubieType 27](#_Toc153437939)

[Property Colours 27](#_Toc153437940)

[Property Permutation 27](#_Toc153437941)

[Moves 28](#_Toc153437942)

[reset 28](#_Toc153437943)

[AssignCubiesArray 28](#_Toc153437944)

[L 29](#_Toc153437945)

[R 29](#_Toc153437946)

[U 30](#_Toc153437947)

[D 30](#_Toc153437948)

[F 31](#_Toc153437949)

[B 31](#_Toc153437950)

[E 32](#_Toc153437951)

[S 32](#_Toc153437952)

[M 33](#_Toc153437953)

[La 33](#_Toc153437954)

[Ra 34](#_Toc153437955)

[Ua 34](#_Toc153437956)

[Da 35](#_Toc153437957)

[Fa 35](#_Toc153437958)

[Ba 36](#_Toc153437959)

[Ma 36](#_Toc153437960)

[Ea 37](#_Toc153437961)

[Sa 37](#_Toc153437962)

[x 38](#_Toc153437963)

[y 39](#_Toc153437964)

[z 40](#_Toc153437965)

[xa 41](#_Toc153437966)

[ya 42](#_Toc153437967)

[za 43](#_Toc153437968)

[CheckSolvable 44](#_Toc153437969)

[checkface 45](#_Toc153437970)

[SOLVE 45](#_Toc153437971)

[PaletteChanges 46](#_Toc153437972)

[PaletteY\_click 46](#_Toc153437973)

[PaletteW\_click 46](#_Toc153437974)

[PaletteB\_click 47](#_Toc153437975)

[PaletteG\_click 47](#_Toc153437976)

[PaletteR\_click 48](#_Toc153437977)

[PaletteO\_click 48](#_Toc153437978)

[faceletclicked 49](#_Toc153437979)

[Form1 49](#_Toc153437980)

[Scramble\_Click 50](#_Toc153437981)

[Form1\_Load 52](#_Toc153437982)

[btnReset\_Click 53](#_Toc153437983)

[L\_Click 53](#_Toc153437984)

[R\_Click 53](#_Toc153437985)

[U\_Click 54](#_Toc153437986)

[D\_Click 54](#_Toc153437987)

[F\_Click 54](#_Toc153437988)

[B\_Click 54](#_Toc153437989)

[M\_Click 54](#_Toc153437990)

[E\_Click 54](#_Toc153437991)

[S\_Click 54](#_Toc153437992)

[btnLa\_Click 54](#_Toc153437993)

[btnRa\_Click 54](#_Toc153437994)

[btnUa\_Click 55](#_Toc153437995)

[btnDa\_Click 55](#_Toc153437996)

[btnFa\_Click 55](#_Toc153437997)

[btnBa\_Click 55](#_Toc153437998)

[btnMa\_Click 55](#_Toc153437999)

[btnEa\_Click 55](#_Toc153438000)

[btnSa\_Click 55](#_Toc153438001)

[btnx\_Click 55](#_Toc153438002)

[btny\_Click 55](#_Toc153438003)

[btnz\_Click 55](#_Toc153438004)

[btnxa\_Click 56](#_Toc153438005)

[btnya\_Click 56](#_Toc153438006)

[btnza\_Click 56](#_Toc153438007)

[btnHelp\_Click 56](#_Toc153438008)

[ArrangeButtons 57](#_Toc153438009)

[CheckSolved 59](#_Toc153438010)

[ShowLog 59](#_Toc153438011)

[PaletteY\_Click 59](#_Toc153438012)

[PaletteW\_Click 60](#_Toc153438013)

[PaletteB\_Click 60](#_Toc153438014)

[PaletteG\_Click 60](#_Toc153438015)

[PaletteR\_Click 60](#_Toc153438016)

[PaletteO\_Click 60](#_Toc153438017)

[btnSolve\_Click 61](#_Toc153438018)

[checkEditingMode 61](#_Toc153438019)

[Testing – TO BE FINISHED BY 8/12/23 62](#_Toc153438020)

[Test Log 62](#_Toc153438021)

[Objectives Testing 76](#_Toc153438022)

[Evaluation – TO BE FINISHED BY 19/12/23 82](#_Toc153438023)

[Evaluating objectives 82](#_Toc153438024)

[Feedback from user 84](#_Toc153438025)

[Improvements 84](#_Toc153438026)

[Actually use the underlying cube (objectives 1 and 4.1) 84](#_Toc153438027)

[Arrangement of buttons (objectives 3 and 5.7) 84](#_Toc153438028)

[Box around palette reflects the selected colour 84](#_Toc153438029)

[Animation (objectives 5.4 and 5.5) 85](#_Toc153438030)

[The algorithm to solve the cube (Kociemba’s algorithm) (objective 10) 85](#_Toc153438031)

[Checking if the cube is solved 85](#_Toc153438032)

[Links 86](#_Toc153438033)

# Description

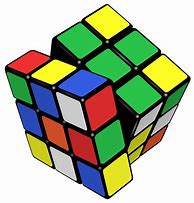
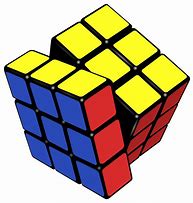
## Description of the problem to be investigated

I am going to be making my own Rubik’s Cube program, in order to investigate how a Rubik’s Cube can be stored on a computer and to experiment with the algorithms used.

## The Rubik’s Cube

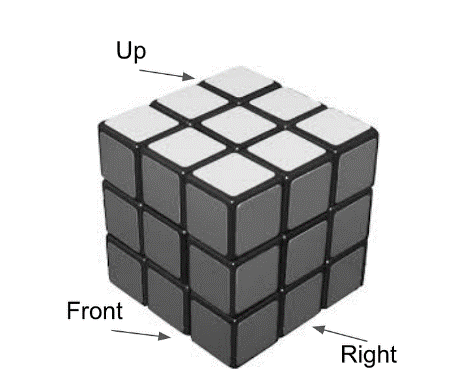
The classic Rubik’s Cube was developed by Erno Rubik when he was looking for a solution to get his students to understand 3D architecture. It took him a month to solve it when he scrambled it for the first time. He originally called it the “Magic Cube”.

Original state: Scrambled state:



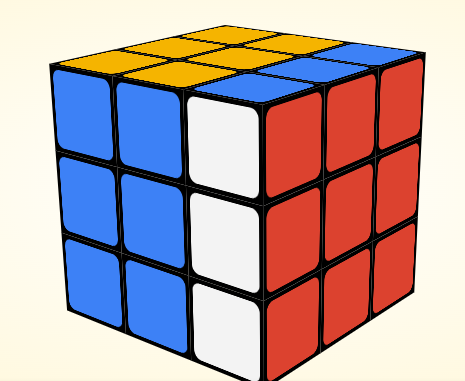
The classical Rubik’s Cube consists of a 3x3x3 physical grid of 27 ‘cubies’ which can move around. There are 6 ‘faces’ which consist of 9 ‘facelets’ each, meaning there are 6x9=54 facelets in total. Each face is one of 6 colours from white, yellow, blue, green, red and orange. The yellow and white centres are opposite each other, the blue and green centres are opposite each other, and the red and orange centres are opposite each other. By default, in the Rubiks community, the yellow centre is on top and the white centre is on the bottom, although people could use green and blue or red and orange for fun.

There are notations used by the Rubiks community to name the possible moves that can be done on a Rubiks cube. Each face is moved in relation to how it looks as if it was being looked at. Each face is denoted by its initial, that is (R)ight, (L)eft, (F)ront, (B)ack, (U)p and (D)own.

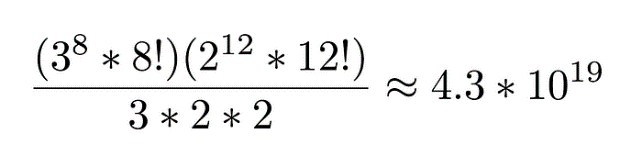


If we want to move the front face clockwise, we will notate that move with the front face’s initial (F). If we want to turn the face anticlockwise, we add an apostrophe after the face’s initial (F’). If we want to turn the face 180°, we put the face’s initial and a 2 to denote that we turn it 90° twice (F2). There are also lowercase moves (f, r, f2, r2, f’, r’, etc.) which denote turning the two columns on that side. This also means turning the cube so that for example, with the move r, the front face becomes the up face and so on.

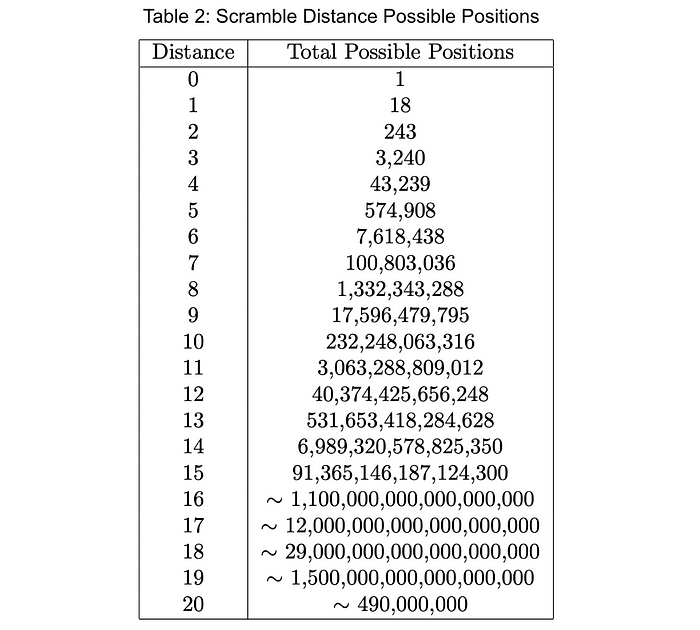
R performed on the cube:



There are 43 quintillion possible cube states. This is because of all the possible positions of the corner and edge cubies, although we do have to filter out impossible orientations and permutations. This is because there are cube states which cannot be solved, where the edge and corner orientations and permutations do not add up correctly, and it would be as if someone twisted either a corner or an edge piece to make a prank. However, if a correct number of cubies were twisted, then the cube would be solvable again as the permutations would cancel each other out.



In 2010, Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge reduced the 43 quintillion possible combinations by a factor of 48 using mirrored states and symmetry. There are 24 ways a Rubik’s cube can be oriented in space and this can be multiplied by two to deal with mirrored states. After this reduction, 55,882,296 cube states remain. Using Google’s computers, they were able to generate the numbers shown in Table 2. Calculating these numbers would take a good desktop computer 35 CPU-years.



From this, we can conclude that it is possible to solve any cube state in a maximum of 20 moves (Rubiks Cube’s God’s number)! A puzzle’s “God’s number” can be calculated by working out the least possible number of moves to solve the puzzle in any randomly generated state. Any algorithm which works out how to find these optimal moves is therefore called a “God’s algorithm”.

There are many different algorithms that can be used. There are “human algorithms”, which can be solved by a human, usually intuitively. Some are more intuitive than others, which follow more of a flowchart, such as the beginner’s method. Then there are also “computer algorithms”, which can be used by computers. Some are quicker than others based on the programming language and the data structure used and how complicated the skills to program them are.

# Analysis – TO BE FINISHED BY 15/9/23

## Analysis/Research Plan

First of all, I need to interview a user who will use my Rubik’s cube solver as this will define any requirements that I will have.

I have made a list of things that I will research:

* How to display the cube on screen to the user.
* Which algorithm is the best to use in relation to which data structure I am going to use.
* How to represent the cube in code in conjunction with and considering which algorithm I am going to use.

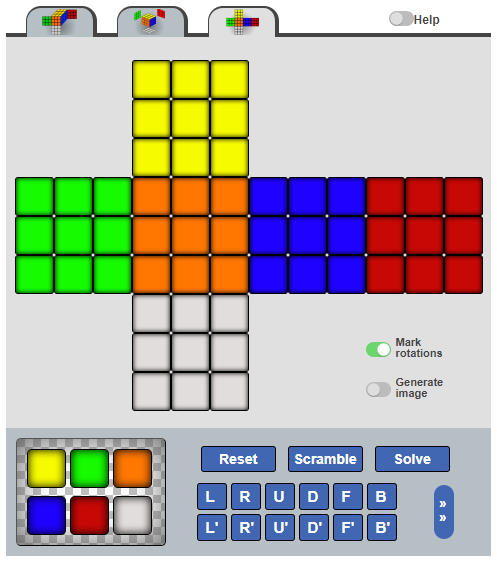
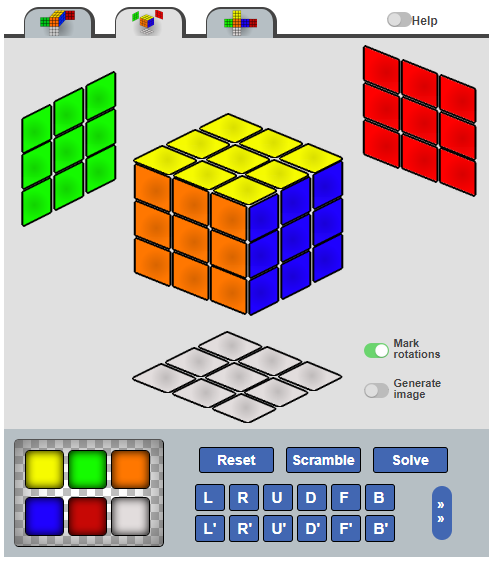
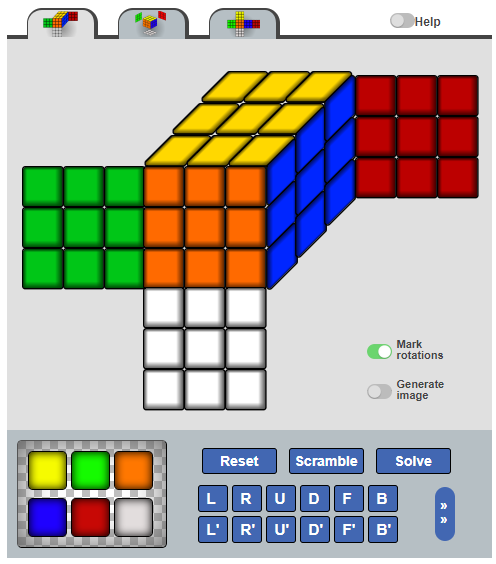
There are some things that I will need to consider before others. I will look at different data structures that others have used first so that I can get a strong foundation when I look at different algorithms. I learnt that the algorithm used will greatly define which internal data structure to use. Finally, I will look at how to represent the cube onscreen for users and for myself to test if the cube is working.

## Existing Systems

I searched for existing Rubiks cube solvers to get some inspiration. Therefore, they will all be web applications but they are similar in user-interface to a form application.

### ruwix.com

[Online Rubik's Cube Solver App (ruwix.com)](https://ruwix.com/online-rubiks-cube-solver-program/) had an option to choose the layout of the cube and how it was displayed on screen. This included three options as shown below.



The third looks easier to program to display on screen, as all of the facelets are of the same type, whereas for the other two there would have to be three different types of facelet class as the three types look different on screen. This will require object-oriented programming and composition, as the faces will be composed of a 2D 3x3 array of facelets. The issue with this approach is that it will be significantly harder to visualize the cube.

The buttons are ordered so that the anticlockwise moves are underneath the clockwise moves. When clicking on the arrows to the right (in the red boxes), more moves appear, including the middle rows and columns and rotating the cube.



I am also going to have a palette of colours. However, I think I will put the extra moves under the basic moves.

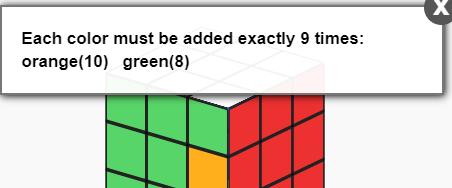
### rubiks-cube-solver.com

This one also had the three layouts and also had animation, but I will not use 3D animation. It had arrows to show the rotations, but I don’t know how to do that and I will stick to the letter annotations.

When clicking on the solve button, if the centres didn’t match up, it showed a message saying:

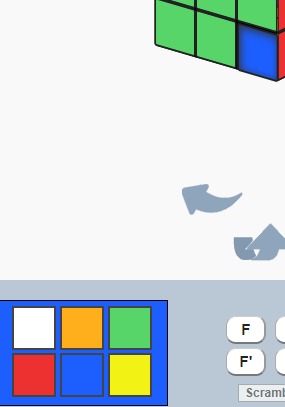
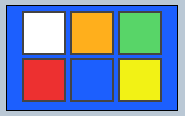


If the centres were alright but the total number of facelets of each colour didn’t match up, it showed a message saying:



Whilst also specifying how many facelets were of each colour that didn’t have 9 total facelets of that colour.

It had a box around the palette which changed its colour to that of the selected palette colour. It sometimes changed its colour to nothing when a facelet on the cube was clicked, and sometimes stayed the same colour. I will ask my end-user what he thinks about this.



## Type of Application

There are a few options, of which a console application, a web application, and a form application. I have never made a web application, as I would need a domain name to show it to other people, which costs money. I would also have to learn HTML which I do not have the time to do. A form application uses OOP in the interface, and it will be easier to display buttons. A console application would require a command line interface where the user types in commands (which will have to be remembered), however I think that the experience would be more enjoyable if buttons were involved. A form application also looks better. The examples online all use a web application, which looks like a form application, so mine will look better if it looks like the others available online. However, I have never seen a Rubik’s cube solver online that uses a command line, so if I use a console application, it will stand out, and give some variety to users who are looking for a Rubik’s cube solver.

I will use a Windows Form rather than a Console Application as it uses OOP more, I can output the faces more easily and it will be easier to animate.

## How to represent the cube in code

This StackOverflow discussion ([data structures - How would you represent a Rubik's Cube in code? - Stack Overflow](https://stackoverflow.com/questions/500221/how-would-you-represent-a-rubiks-cube-in-code)) suggests a lot of good ways to represent the cube, such as binary, a 6x3x3 2D array and a 3x3x3 3D array. The 3D array seems more intuitive, as that is what a Rubiks Cube is. The cube will be a 3x3x3 array of cubies. A cubie will have a CubieType, which will dictate how many colours it is allowed to have. I will also need to store their permutations for each cubie. For simplicity, I will denote each cubie’s position as (U|D)(F|B)(L|R)

There was also a solution using binary, which stores the six faces in six strings. It was deemed to be much faster, as moves and rotations would be represented by bitwise operations. To store a face, only the outer 8 facelets would be stored as the middle facelet wouldn’t move. This means that a face would be represented by 8 3-bit patterns, as there are only 6 colours. For example, the face “

WGR

G B

WYO

” will be represented in code for example as “00000000 00000001 00000010 00000011 00000100 00000101 00000000 00000001”. It starts at the top left facelet and goes clockwise around the face. To turn the face clockwise, a bit rolling operation turns it into “00000000 00000001 00000000 00000001 00000010 00000011 00000100 00000101” which is equivalent to “

WGW

Y G

OBR

”. Any adjacent facelets affected could be manipulated manually using trivial manipulation. I thought of reducing it to 3 bits per facelet.

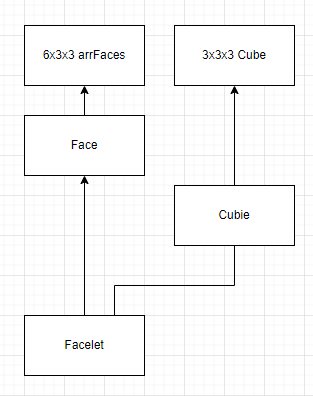
I had the idea to not store it in binary, but instead as a string of colour initials (still going clockwise around the face), as it would be easier to read it when looking at the code.

The 3x3x3 array will make rotation easier to visualize, despite apparently being slow to move around. I could try the 6x3x3 array of faces, although this will mean that the faces’ position in the array reflects their positions as seen from the user. This means that every time the cube is rotated, the faces will all have to switch positions and some will be reordered, which is very confusing, however I think it is worth a try.

After some reflection, I have decided that I will use both a 3x3x3 array of cubies and a 6x3x3 list of faces, so that I can experiment and see which one is easier to move its elements around.

## Class diagram

The 3x3x3 cube will be composed of cubies, and a face is composed of nine facelets. This is worthy of a simple class diagram. When making it, I realised I could allocate facelets to a cubie, which would make it easier when updating the onscreen cube net.



## Which Algorithm to Solve?

I will obviously need to have a solve button which will call a subroutine which will work out how to solve the cube from the current state. Therefore, I will need a solving algorithm.

I had to find out which algorithm was going to be the best one to use in my coursework. I was well aware that there were “human algorithms” and “computer algorithms”, but the problem was going to be to identify which one I would use and explain why.

I quickly learnt that the underlying data structure and programming language used aren’t that important for human algorithms, however they have to be carefully chosen for computer algorithms.

I chatted with my end-user and he said that he would like a computer algorithm as it is more impressive and also what’s the point of implementing a human algorithm when the user can just do it themselves. Therefore, I will use a computer algorithm.

## Kociemba’s Algorithm

After some research, I read about Kociemba’s algorithm, an algorithm developed by Herbert Kociemba and which was later developed upon by Koch, who made the 20-move God’s algorithm for the Rubik’s cube. It follows the Two-Step Approach. I got all of my inspiration from Kociemba’s own website: [Solve Rubik's Cube with Cube Explorer (kociemba.org)](http://kociemba.org/cube.htm).

### How does the algorithm work?

The algorithm follows a two-step approach. The first step tries to get the fully scrambled cube to a subset of cubes (called G1). Step 2 uses just the moves U, D,

## Scrambling

Obviously, there will need to be a scrambling function so that the user has the challenge to solve it. I wondered if the scrambling function would continue scrambling from the current state of the cube, or from the default position. It would be more sensible to continue from the current state, because if the user wants the cube to be scrambled from the default position, they can just press the Reset button and then scramble.

The program will generate a random series of moves and execute them.

It would be nice to have a textbox in which the user can input how many moves they would like the program to execute to scramble the cube. To prevent errors, the textbox will be an up-down control with up and down arrows, which the user can press if they want more or less moves in the scramble. There could also be a tick box if the user doesn’t care about how many moves there will be in the scrambling process. I asked my brother what he thought about that and what range he would like, and he said \_ to \_\_. The program will choose a random number between 5 and 20.

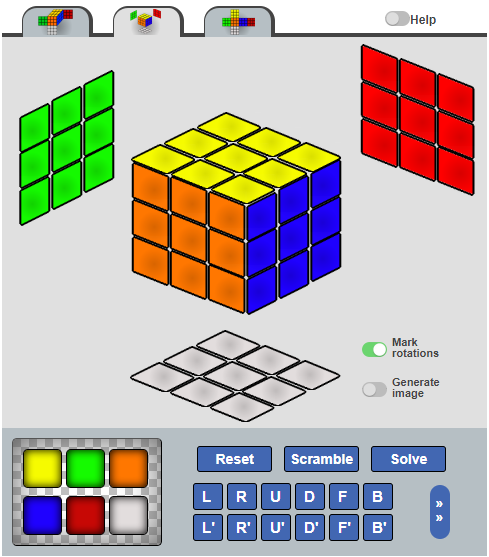
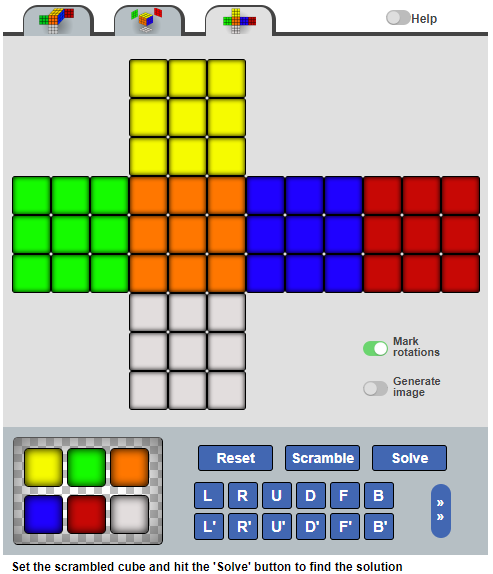
I will keep a score of the moves needed to solve the cube. Each time the user makes a move, I will add it on to the beginning of the list of moves needed to solve the cube. When hitting ’solve’, I will count the number of moves required to solve the cube and compare it with \_\_, which is the maximum number of moves which Kociemba’s algorithm solves any cube in. If the number of moves in the score exceeds \_\_, a new solving process will begin.

I don’t know yet if the scrambling process will be animated or if the facelets will be updated all in one go. I will ask my end-user.

## Graphics

There are a few options, of which one I really liked from ruwix.com (see below) as it shows the non-visible faces like you are looking at them like an X-ray and it is easier to make the links between which facelet is on the same cubie as another facelet. However, a classic net would be easier to represent on a Windows Form. If I have time after making the rest of the fundamentals of the program, such as the moves, rotations and the solving algorithm, I could experiment with the graphics. I will put the yellow face at the top and the white face at the bottom, as that is what is considered the ‘default’ position in the Rubiks community.

From ruwix.com: Classic net:

Clearly, each face will need to be comprised of nine facelets in a 3x3 grid, no matter which design I choose. Therefore, I will use object-oriented programming in order to do this and create a class called Facelet.

## UI

I could choose between a 3D animation using Java, or I could use a cube net.

|  |  |
| --- | --- |
| **3D** | **Cube net** |
| * More advanced * I will need to use another IDE than I am used to and learn a new language (Java). | * Simpler to implement. * I can implement it using the skills I already have. |

Overall, I think I will use a simple cube net, as I feel like the 3D animation would be outside of the scope of what I am trying to achieve: I am focusing on the implementation of a Rubik’s Cube and the algorithms involved.

I will have a palette of colours, and when a facelet is clicked, it will change colour. However, this must be done in an editing mode, therefore I will introduce an Editing Move tickbox.

The move buttons will need to be neatly arranged to avoid confusion and make it obvious which buttons are the opposites of which.

## Moves

For executing the moves, I will make six subroutines for R, L, F, B, U, and D. I would like to prevent the Cube from being public. Therefore, the moves will also be functions, returning the new state of the cube.

When a 180° turn is required, I will just call the relevant move two times. I will not bother putting buttons for 180° rotations, as online Rubiks websites do not have them, and the user can press either of the clockwise and anticlockwise buttons twice. For an anticlockwise move, I will create separate move subroutines as the animation flows more easily, rather than call the move 3 times. I decided to keep each move stationary, that is to say relative to the forward-facing face. For M(iddle) and E (turning the middle layer 90° to the left), I will implement the algorithm that I am in the process of such that it will compliment the new state of the cube. For example, when an M happens, the centre at the front will go to the top centre spot. Then I will switch each rotation such that for example a U (top layer 90° clockwise) turns into B (back layer 90° clockwise).

## Dialogue with the end-user

### Interview

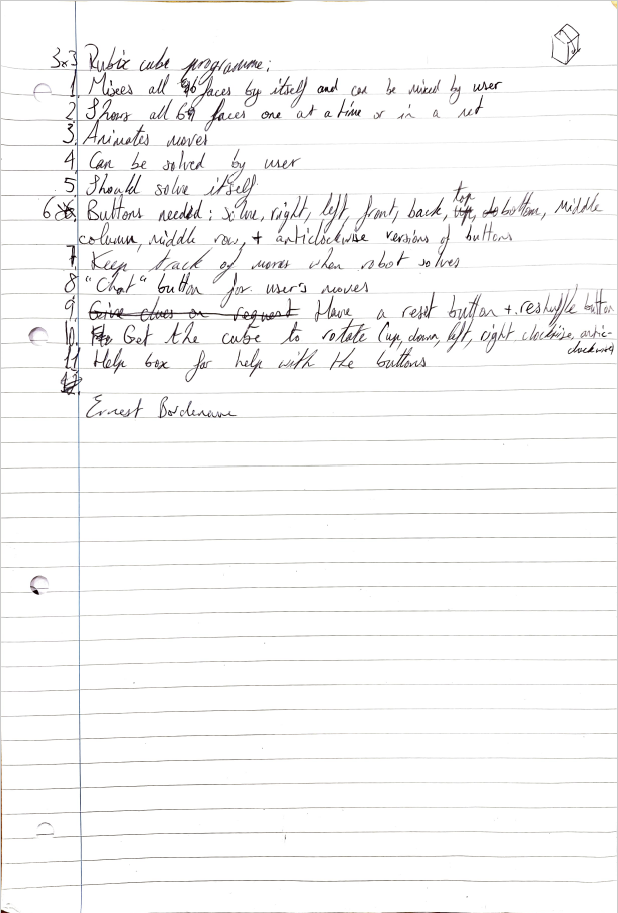
I asked my brother if he would be willing to use my Rubik’s Cube solver. Before I showed him anything, I asked him what he would like the program to do. He put the following down:

1. “Mixes all 6 faces by itself and can be mixed by user”
2. “Shows all 6 faces one at a time or in a net”
3. “Animates moves”. He wanted it to show a 3D cube, but I said that I will not be doing that as it would be time wasted on animation. We decided on the end that the moves will be animated (in the direction that the cubies are moving in) on the net that I had already put on the Windows Form.
4. “Can be solved by user”
5. “Should solve itself”. He elaborated on this afterwards that it should use an algorithm with as few moves as possible, so that it is more impressive and otherwise he could have done it himself.

Then I showed him how far I had come at that point, and I asked him if he had any further ideas from inspiration.

1. “Buttons needed: solve, right, left, front, back, top, bottom, middle column, middle row”. I asked him if he wanted anticlockwise buttons for each of the moves and he said yes, so he added “+ anticlockwise versions of buttons”. He also stressed on the fact that the buttons needed to be more explicit.
2. I thought of adding a textbox to the right under where the solve button was going to go, as there was space on the right of the Windows Form. He said yes and added “Keep track of moves when robot solves”.
3. “‘Chat’ button for user’s moves”. I was confused, so he explained that it was going to keep track of what has happened, such as the user’s moves, scrambles, and the moves to do when the solve button is pressed.
4. He put down “Clues on request”, that is to say the program would output in the chat what the next move to solve the cube was. I said that that was a bit useless, because the user will probably press the clue button loads of times, and that would just be the same as clicking the solve button once.
5. I showed him the reset button that I put on, to put the cube back to its original state. He was a bit confused as to why, and I said it was to consider users that can’t solve the cube, and the user could do their own moves on a solved cube straightaway if they wanted to. He put down “Have a reset button + reshuffle button”. The reshuffle button is to re-scramble the cube.
6. Get the cube to rotate (up, down, left, right, clockwise, anticlockwise).
7. Help box for help with the buttons. I suggested putting a help button that when it is pressed, another window will appear, explaining what all the buttons do, for example for a user who is new to the Rubik’s Cube or has never been on an online Rubik’s application.

Here is a scan of the paper he used to note his ideas down:



### Analysis of user ideas

* Number 1 is just scrambling. I am wondering whether I will use a computer algorithm (Kociemba’s algorithm), or scramble with random moves. I will ask my end-user whether he wants to have a textbox which allows him to input how many moves he wants to scramble the cube with.
* For user requirement 3, I will be using the sleep function, although I will need to experiment with how many milliseconds will need to be slept on.
* User requirement 5 is a good point, and I did not think about that. I will need to use a computer algorithm (I have chosen Kociemba’s algorithm) instead of a human algorithm.
* In regards to user requirement 6, I will line the buttons up such that the anticlockwise buttons will be underneath their respective clockwise rotations. There were only the moves’ initials shown on the buttons when I first showed my program, so I was thinking that I will need to put “Right clockwise” or “Up anticlockwise”. For anticlockwise moves, I think I will need to make separate functions for them, as the animation will be more suitable, instead of animating a clockwise move 270°.
* For number 7, I think I will erase the moves in the solve box one by one once each move has been executed.
* Number 8 is just a log to keep track of what has happened since the program was opened. I am thinking of making a text file which will only be able to be added to. It will include everything that has happened in the Rubiks Cube solver from the moment the log file has been created and works. There will be a variable which will mark the beginning of the current session, which will allow the user to see only what has happened in the current session, or even everything that has happened ever in the Rubik’s Cube solver.
* There is no need for number 9. The Reset button is a good idea, as it allows a user to reset the cube if they don’t know how to solve the cube. The reshuffle is just a scrambling function.
* I thought for a moment that I could rotate the cube in 3D space, by rotating the 3x3x3 Cube and adding to the permutations of the corner and edge pieces. I mentioned it to my brother and he said yes, that was cool. The net on the Form will also be updated. When animating, I will start from the left- or right- or up- or down-most face, depending on which rotation is being executed, and animate with a wave of the appropriate Facelet colours going in the relevant direction. I will also need to make clockwise and anticlockwise buttons for each of the three ways a Rubiks Cube can be orientated in.
* I will add a Help button which when it is pressed, I will instantiate a new window with a pre-made text, which will be stored in a constant string variable.

### User requirements

1. A reset button to reset the cube back to its original state was already made but is in the user requirements. A reshuffle button is the same as a scrambling button, which will call a scrambling function. The scrambling function will either use a computer algorithm (Kociemba’s algorithm) or generate moves at random, depending on how many moves the user wants to scramble the cube with.
2. Animate the moves, both clockwise and anticlockwise, as well as rotations of the whole cube.
3. Have a solve button which calls a solving function which makes use of Kociemba’s algorithm.
   1. Have a solve box to show the user which moves lead to solving the cube.
   2. Erase the moves in the solve box one by one as they are executed by the user.
4. The buttons have to be more explicit in terms of the text that is shown on the buttons, and they will be lined up according to which face of the cube they are moving and in which direction. This also goes with whole cube rotations.
5. There will be a log window which will pop up when a History button is pressed. It will include things like what moves the user has made, any scrambles, and when the solve button has been pressed.
6. Rotate the whole cube in 3D space. This will need to include relevant buttons for each of the three axes in which the cube can be rotated about, both clockwise and anticlockwise. The rotations will also be animated.
7. A Help button is going to be made. When it is pressed, a new window will appear, containing instructions on what the different buttons do.

## Objectives

1. The underlying cube

1.1 3x3x3 array of cubies

* + 1. Make a Cubie class with a CubieType property. Make sure each Cubie has the right number of colours for its CubieType. Preferably instantiate the cubies at run-time.
    2. Store the cubies’ permutations.
    3. Make a private Cube made of 27 Cubies and preferably instantiate at run time.

1.2 6x3x3 array of faces

1. Create the clockwise and anticlockwise moves as functions and return the new cube state, which will prevent the Cube from being public.
2. The buttons on the form have to be neatly arranged and explicit, and there will be a button for each clockwise and anticlockwise move function.
3. Make six rotation functions, a clockwise and an anticlockwise rotation for each of the three axes.
   1. The underlying cube will be updated first before any animation happens.
4. UI
   1. Make a Face class, which will be composed of 9 Facelets in a 3x3 2D array.
   2. Display six Faces in a classic net on a Windows Form.
      1. Make sure that yellow is above and white is on the bottom, so as to not confuse anyone.
      2. (Optional) Create the faces at run-time.
      3. (Optional) Recreate Option 2 from ruwix.com.
   3. Make a default position where each face is only one colour and show it when the Reset button is clicked.
   4. Animate each move, both clockwise and anticlockwise. For 180° rotations, I will call and animate the clockwise move twice. Use the sleep subroutine when animating moves.
   5. Animate the rotations. The facelets on the far side will be updated first, and the wave will continue in the direction that the cube is being rotated in. Any faces on the axis of rotation will be animated in a clockwise or anticlockwise manner, and will be animated along with the rest of the cube as it rotates.
   6. Make a colour palette so that the user can input individual colours.

5.6.1 Have a tickbox so the user can say that they are in Editing Mode.

* 1. Have the move buttons in neat order. Align the clockwise and anticlockwise move buttons with their respective partner.

1. Make a Reset button which will call an algorithm that resets the state of the cube and facelets.
2. Make a scramble function. It will work in conjunction with a textbox(?) in which the user can input how many moves they want the scrambling function to scramble with.
3. A History button will be made.
   1. When it is pressed a new window will appear, which will contain a log of everything that has happened with the cube so far in the session.
   2. A text file is to be created to keep track of what has happened.
   3. An integer variable will contain the line number of the file which starts the current session.
4. A Help button will be created.
   1. When it is pressed, a new window will appear containing instructions on how each button works and describe their relevant moves.
   2. The text string in the Help window will be stored in a constant variable.
5. The algorithm to solve the cube (Kociemba’s algorithm)

# Design – TO BE FINISHED BY 13/10/23

## Programming Diagram

trfdfg

## Overall System Design

This table shows the inputs, processes, storage and outputs of the program.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Processes** | **Storage** | **Output** |
| -Colour palette  -Editing Mode button | -Solving algorithm  -Reset algorithm  -Scrambling algorithm | -3x3x3 cube of cubies  -6 faces with 9 facelets | -Classic net  -Solving box  -Readable log |
| Solve button | Solving algorithm | -state of cube at the start of scrambling  -current state of cube. | Moves required to solve the cube in the solve box |
| Move buttons | A function per button | -3x3x3 cube  -6x3x3 faces | New state of cube |
| Scrambling button and up-down control | Scrambling algorithm | -state of cube at start of scrambling  -state of cube as moves are executed | New state of cube |
| Palette (6 colours) | Change colour of facelets |  | Change colour of facelets |

## Underlying structure

A Face and a Facelet class will be made. Each Face will be comprised of 9 Facelets in a 3x3 2D array (objective 5.1). I will instantiate the faces and facelets at run-time (objective 5.2.2). If I go for option 2 from ruwix.com, there will need to be 3 different types of facelet, and I do not yet know how I will connect them with the Cube and animating the moves.

The Cube will be instantiated at run-time, which will be a 3x3x3 3D array of 27 Cubies (objective 1.1.3), which will also be instantiated at run time. A Cubie also has a CubieType, which identifies whether it is a corner, edge, centre, or the middle. Each Cubie will have a Colours array, but depending on its CubieType, a different number of colours will be in the array (objective 1.1.1). The face centres will be CubieType 1; the edges will be CubieType 2; the corners will be CubieType 3, and the unseen middle Cubie as CubieType 0. This is so a Cubie’s CubieType is the same as how colours it is allowed to have. This will make it easier in the future when assigning colours, so that the program can check if a cubie has the right number of colours allowed. I don’t think it matters in what order the colours are stored, as I can use the Linear Search algorithm on the colour arrays that have a maximum of 3 elements. There will be an integer Permutation variable in the cubie class, which will satisfy objective 1.2.

Cubie class:

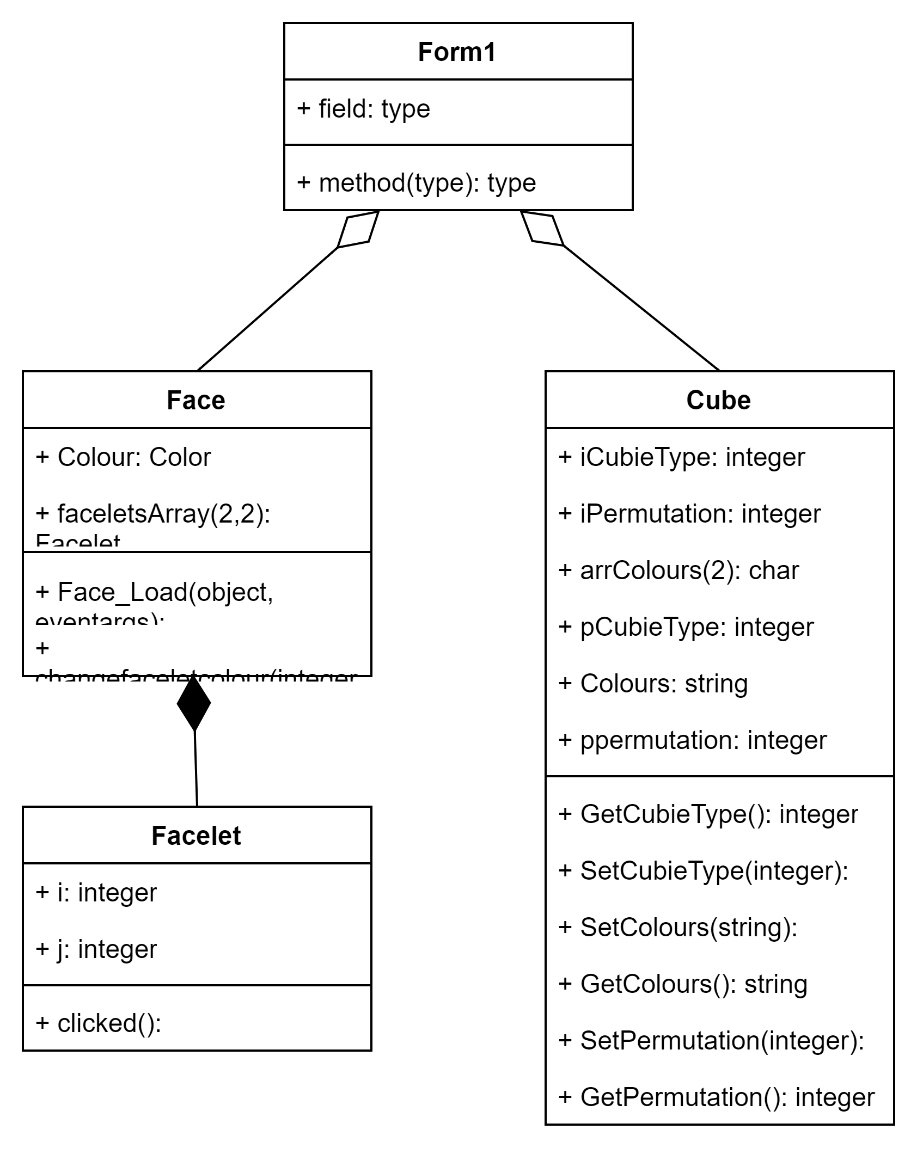
|  |  |  |  |
| --- | --- | --- | --- |
| **Data item** | **Property name** | **Variable type** | **Purpose** |
| Number of colours | Type | Integer (iCubieType) | This variable will also be used to check whether the cubie is a corner, edge, centre or the middle cubie. It will define how many colours a cubie is allowed to have. |
| The cubie’s colours | Colours | An array of characters (arrColours) | This stores the cubie’s colours and will have the number of elements as dictated by the Type variable. |
| Permutation | Permutation | Integer (iPermutation) |  |

There will also be a 6x3x3 array of faces (objective 1.2), where each face is a 3x3 array of characters which represent the colours. I will order the faces in the following order:

|  |  |
| --- | --- |
| **Position** | **Face position** |
| 0 | Front |
| 1 | Back |
| 2 | Left |
| 3 | Right |
| 4 | Up |
| 5 | Down |

I will store the facelet colours from left to right and from top to bottom for each face as if I was looking at it.

## Class Diagram



## Moves and Rotations

For the moves, I will first rotate the facelets on screen all at once so that the move is static on-screen, and when the time comes, I will rearrange the lines of code and add sleep lines so that it animates the moves (objective 5.4). I will need to group facelets that will change colour at the same time.

The moves subroutines will all be public and in a Moves module. This will make sure the program satisfies the Excellent Coding style category.

In VB.net, the apostrophe symbol (‘) is used to comment lines of code. Therefore, I cannot use this symbol when naming my anticlockwise buttons and subroutines. I will replace it with a lowercase ‘a’ to stand for anticlockwise. When implementing the anticlockwise subroutines, I will simply swap the lines of code of ‘opposite’ facelets.

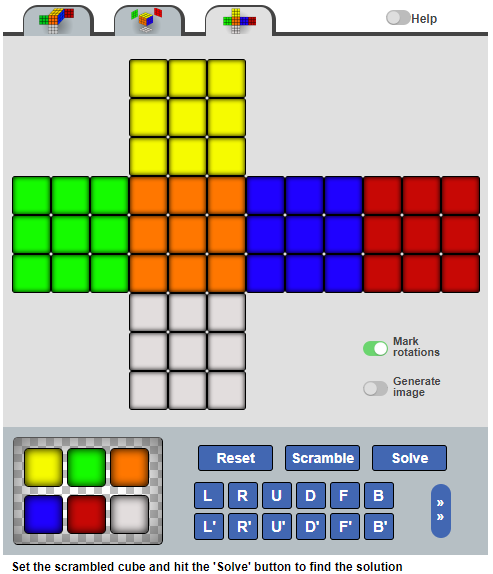
For rotating, I have decided to do the whole rotation in one go. For example, x could also be done with the M, R and L moves in any order. However, for the simplicity of things and to make sure the animation works, I have decided that the rotations will be implemented as their own subroutines.

The problem with testing these is that the facelets on the face being rotated don’t change if the move being tested is the first one executed. Therefore, I will have to use a combination of moves and have a physical 3x3x3 Rubik’s Cube next to me or compare against a website so I can see if the facelets match.

In general for the moves, I have moved the facelets on the face being rotated, then the neighbouring facelets of the neighbouring faces, in the order top, middle, bottom, or left, middle, right depending on which direction the face is facing.

## UI

For displaying the cube on screen, I will use a simple classic net (objective 5.2):

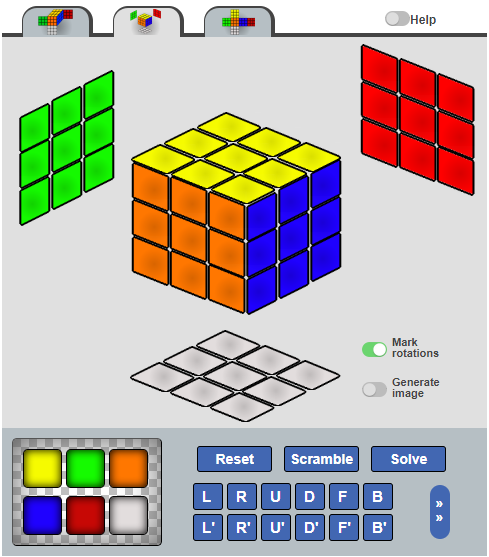


(screenshot from ruwix.com)

I will also put the yellow face on top and the white face on the bottom (objective 5.2.1).

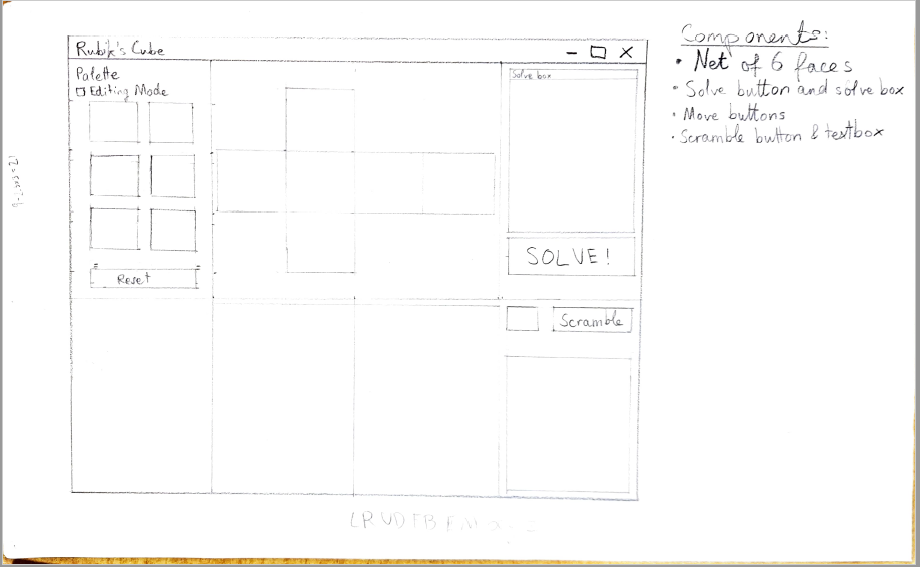
When the program opens, the cube will be reset to the default position. Hence, the default position will occur only when the program opens and when the Reset button is pressed.

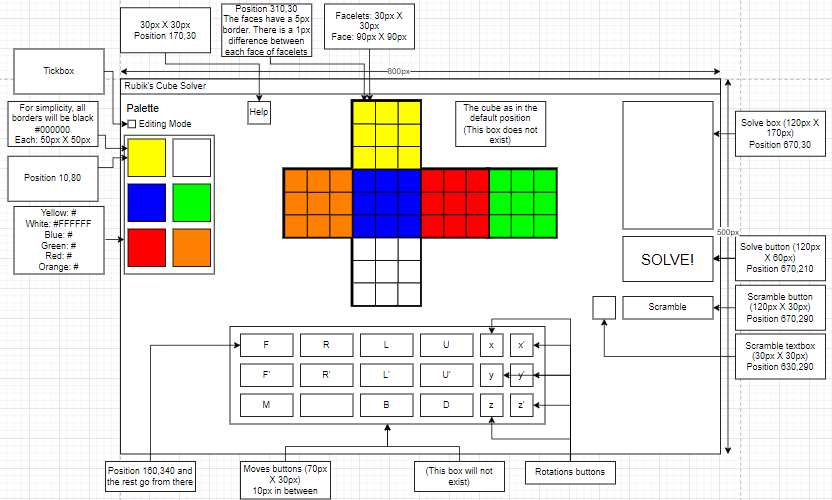
However, if I have enough time after having completed all of the other requirements/objectives of the program, I will try doing option 2 from ruwix.com (objective 5.2.3).



(screenshot from ruwix.com)

### Designing the UI

I first drew a draft of what the UI was going to look like on a piece of paper:

Next, I recreated it digitally using draw.io and added labels: 

### Explanation of the UI design

|  |  |  |
| --- | --- | --- |
| **Component** | **Position coordinates and size** | **Purpose** |
| Face | Size:90x90 |  |
| Facelet | Size:30x30 |  |
| Cube net | Positions from top-left corner  Y:310,30  W:310,210  B:310,120  G:490,120  R:400,120  O:220,120 | Show the current state of the cube |
| Solve button | Gdf | ghd |

## Palette

For the palette, I want it to be clear when a palette colour has been clicked. Therefore, the facelets will have to be buttons.

When clicking on a palette colour, if there is no facelet selected, the palette’s colour will be stored in memory. If a facelet was selected just before selecting the palette colour, the colour of the facelet will be changed to the palette’s colour.

When a facelet is clicked and no palette colour was selected as the last event before the facelt was clicked, the facelet will not change colour, as otherwise it will become too complicated.

If a move happens, all colours and selected facelets will be forgotten.

## Algorithms

### Resetting the cube (meets objective 7)

After some thought, I have realised that I will need a list of all the cubies in order, which will be a constant string array (CubiesColours). I will start with the top left cubie, go from left to right and up to down, and go from front to back. i.e. [UFL, UF, UFR (top front row), FL, F, FR, DFL, DF, DFR (front face), UL, U, UR, L, M, R, DL, D, DR (middle column), UBL, UB, UBR, BL, B, BR, DBL, DB, DBR (back face)]. I will declare the array in the Reset subroutine but I will make a function which assigns the colours of the cubies and call it from the Reset subroutine to keep it tidy.

SUB Reset(arrFaces, Cube)

CubiesColours 🡨 AssignCubiesArray(CubiesColours)

‘Put all cubies in cube

FOR cubienumber 🡨 0..26

FBcolumn 🡨 i/9 ‘front, middle or back

LRcolumn 🡨 i mod 3 ‘L, M or R

UDrow 🡨 (i mod 9)/3 ‘U, E or D

‘this prevents 3 nested iteration loops inside each other and uses logic instead

Cube(i, j, k).Colours 🡨 CubiesColours(i, j, k)

‘Reset all facelets

FOR facecount 🡨 0..5:

FOR i 🡨 0..2

FOR j 🡨 0..2

CASE SELECT facecount

1: arrFaces(facecount, i, j) 🡨 “W”

2: arrFaces(facecount, i, j) 🡨 “Y”

3: arrFaces(facecount, i, j) 🡨 “B”

4: arrFaces(facecount, i, j) 🡨 “G”

5: arrFaces(facecount, i, j) 🡨 “R”

6: arrFaces(facecount, i, j) 🡨 “O”

Me.Refresh()

### CheckSolvable

This algorithm checks if the cube is solvable when the Solve button is pressed. This prevents any unnecessary use of the Solving algorithm. Instead of returning a Boolean value, this function will return a string containing the error in the cube which will then outputted if not empty. This function does not need the cube of cubies as a parameter as it will only check the facelets. It firstly checks if the cube is already solved, then it checks if there ae any centres in the wrong places, then adds up all of the facelets to check if there are enough facelets of each colour.

FUNCTION CheckSolvable()

booleanarray() 🡨 {0, 0, 0, 0, 0, 0}

solved 🡨 Form1.CheckSolved()

IF solved = True THEN

RETURN “The cube has already been solved.”

ENDIF

IF FaceU.faceletsArray(1, 1).BackColor <> Yellow THEN

booleanarray(0) 🡨 true

ELSEIF FaceD.faceletsArray(1, 1).BackColor <> White THEN

booleanarray(1) 🡨 true

ELSEIF FaceF.faceletsArray(1, 1).BackColor <> Blue THEN

booleanarray(2) 🡨 true

ELSEIF FaceB.faceletsArray(1, 1).BackColor <> Green THEN

booleanarray(3) 🡨 true

ELSEIF FaceR.faceletsArray(1, 1).BackColor <> Red THEN

booleanarray(4) 🡨 true

ELSEIF FaceL.faceletsArray(1, 1).BackColor <> Orange THEN

booleanarray(5) 🡨 true

ENDIF

iWrongCentres 🡨 0

FOR i = 0 TO 5

IF booleanarray(i) = True THEN

iWrongCentres 🡨 iWrongCentres + 1

ENDIF

NEXT

i 🡨 0

IF iWrongCentres <> 0 THEN

returnstring = “The “

WHILE iWrongCentres > 0

IF booleanarray(i) = True THEN

SELECT CASE i

CASE 0

returnstirng 🡨 returnstring + “yellow”

CASE 1

returnstirng 🡨 returnstring + “white”

CASE 2

returnstirng 🡨 returnstring + “blue”

CASE 3

returnstirng 🡨 returnstring + “green”

CASE 4

returnstirng 🡨 returnstring + “red”

CASE 5

returnstirng 🡨 returnstring + “orange”

END SELECT

IF iWrongCentres > 1 THEN

returnstring 🡨 returnstring + “, “

ENDIF

ENDIF

iWrongCentres 🡨 iWrongCentres – 1

i 🡨 i + 1

ENDWHILE

returnstring 🡨 returnstirng + “ centres are in the wrong place.”

RETURN returnstring

ENDIF

RETURN “”

### CheckSolved

This algorithm checks if the cube has been solved after every move. It will be in its own subroutine in the Form’s code and returns a Boolean value. I would rather have the Boolean cubesolved variable begin as true and turn false, as the majority of the time when the function is called, it will be after a move, so less facelets will be checked before one is spotted to be an incorrect colour. It will not be called when the cube is reset. The subroutine has to check all facelets of all facelets in order to know that the cube is solved.

FUNCTION CheckSolved()

cubesolved 🡨 True

faceletnumber 🡨 0

WHILE cubesolved = True

i 🡨 faceletnumber / 3

j 🡨 faceletnumber mod 3

IF FaceW.faceletsArray(i, j).BackColor <> White THEN

cubesolved 🡨 False

ELSEIF FaceY.faceletsArray(i, j).BackColor <> Yellow THEN

cubesolved 🡨 False

ELSEIF FaceB.faceletsArray(i, j).BackColor <> Blue THEN

cubesolved 🡨 False

ELSEIF FaceG.faceletsArray(i, j).BackColor <> Green THEN

cubesolved 🡨 False

ELSEIF FaceR.faceletsArray(i, j).BackColor <> Red THEN

cubesolved 🡨 False

ELSEIF FaceO.faceletsArray(i, j).BackColor <> Orange THEN

cubesolved 🡨 False

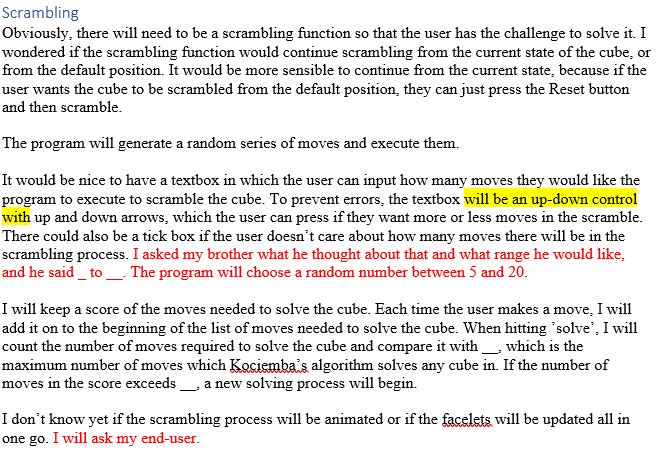
ENDIF

faceletnumber 🡨 faceletnumber + 1

ENDWHILE

RETURN cubesolved

### Scrambling Algorithm



There will be an up-down control in which the user can enter how many moves they would like to be executed by the program to scramble the cube. It is an up-down control instead of a normal textbox to prevent errors form occuring if the user types in text. I found that the NumericUpDown control in the Visual Studio Windows Forms Application has a maximum limit of 100. Therefore, I have decided to put my own maximum limit of 30 so that my program doesn’t have to do lots of moves if the user puts a large number in. It is also possible to enter a numeric value manually.

For the scramble function, the program will generate a random series of moves and execute them on the current cube’s state. I will also need to keep track of the permutations of the cubies since the start of the scrambling process.

I have realised with the way the subroutines are ranked, I will have to implement this subroutine in the Form’s code instead of in the Moves module, as the moves subroutines are privately declared in the Form.

It will keep a score of the moves needed to solve the cube. Each time the user makes a move, it will be added on to the beginning of the list of moves needed to solve the cube. When hitting ’solve’, I will count the number of moves required to solve the cube and compare it with \_\_, which is the maximum number of moves which Kociemba’s algorithm solves any cube in. If the number of moves in the score exceeds \_\_, a new solving process will begin.

[FLOWCHART]

### Solving Algorithm (Kociemba’s Algorithm)

The solving process will use Kociemba’s algorithm. When the solve button is pressed, a function containing Kociemba’s algorithm will be called. This function will be contained in the Moves module.

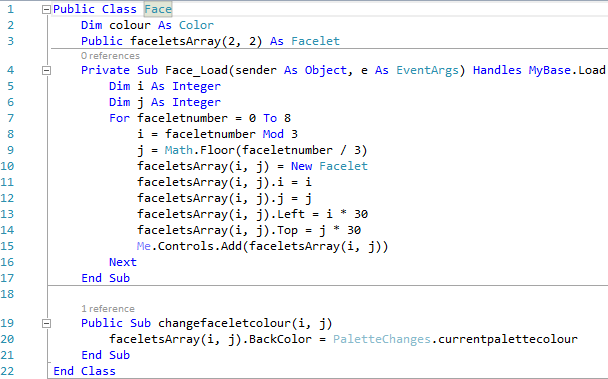
When solving, I will need to store the cube’s state at the start of being solved, and not update what is being outputted to the screen as the moves in the solving process are being explored. If the state is in G1, it will check if the cube is compatible with G2. If it is in G2, it will call the normal CheckSolved function.

There is going to be a read-only solve box which will output which moves are needed to solve the cube. The moves in the solve box will be separated by spaces, and will be outputted into the box using an iteration loop which adds a space after each move. It doesn’t matter if there is a space after the last move as it will not be seen, because the solve box will be read-only. If the next move executed by the user matches the first move in the solve box, the move will be erased from the solve box along with the space that follows it. If the user goes off track, the solve box will reset its contents and go empty.

# Implementation

I will first put the code for the Face, Facelet and Cubie classes, and then the Moves and PaletteChanges modules, and finally for the Form.

## Face

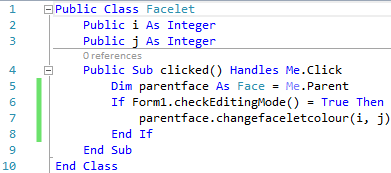


Line 10: The facelets are only instantiated when the face is instantiated (Group A).

Line 3: This is an example of composition, which is a Group A skill.

Line 5 makes use of object-oriented programming and gets the instance of the parent class which the current facelet instance is part of (Group A).

## Facelet

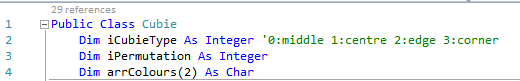


This IF statement going from lines 6 to 8 checks if the Editing Mode tickbox has been ticked before changing the facelet’s colour

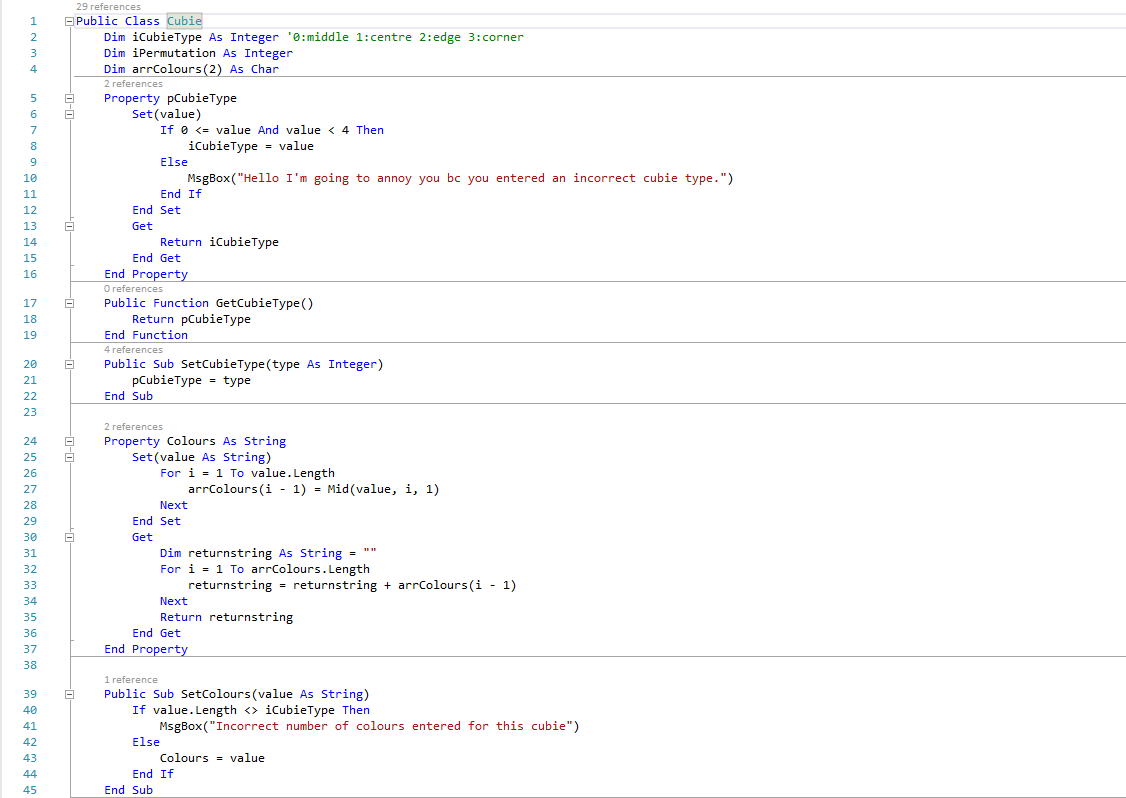
(OOP Group A)

## Cubie

This class makes excellent use of defensive programming (Excellent Coding style), as inputs are passed into a subroutine, then the property, then the variable, instead of directly into the variable. This is to check that the input meets the standards.



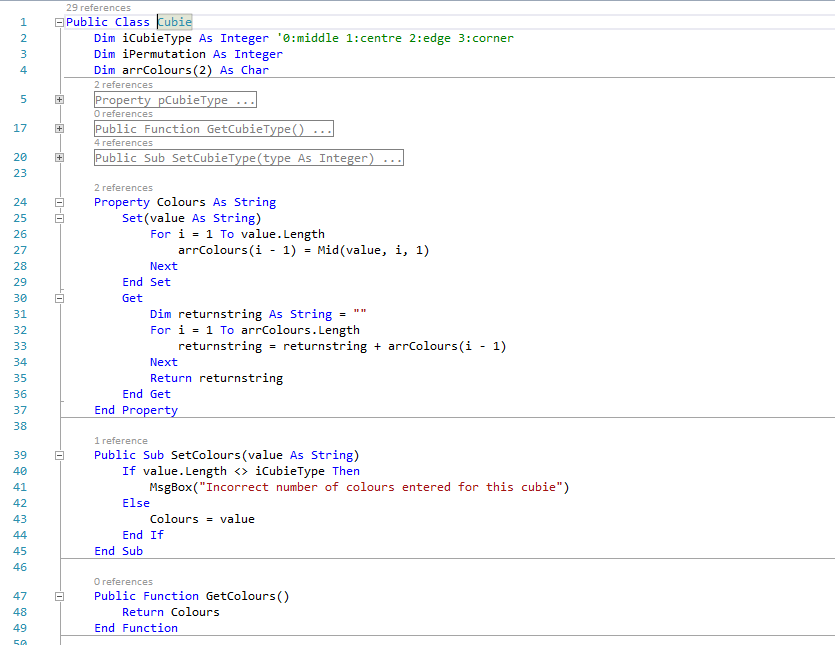
### Property CubieType



This is an example of defensive programming, as a cubietype can only be one of three types.

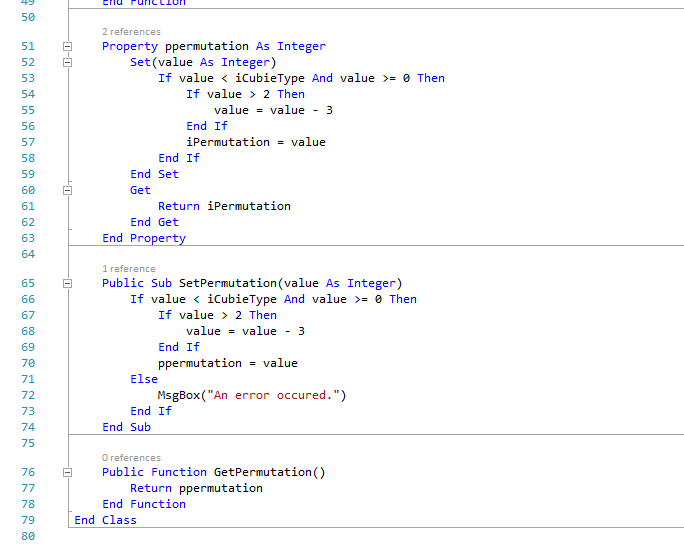
The array arrColours receives the number of colours that is passed into the property.

### Property Colours



This returns the colours stored in the arrColours array.

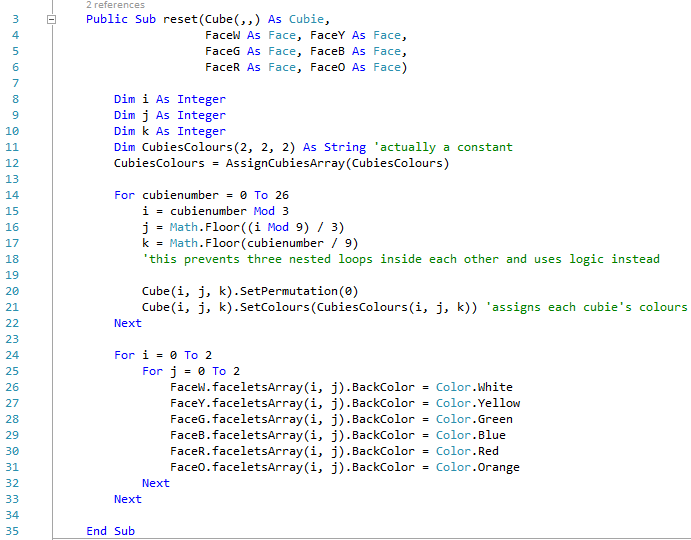
### Property Permutation



## Moves

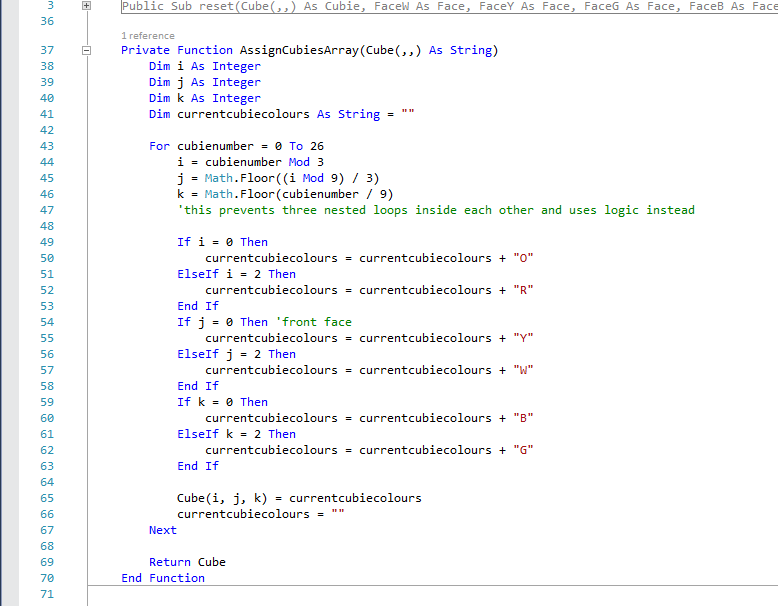
The majority of subroutines in this module involve complicated user-made algorithms (Group A), which required a lot of visualization to implement them.

### reset

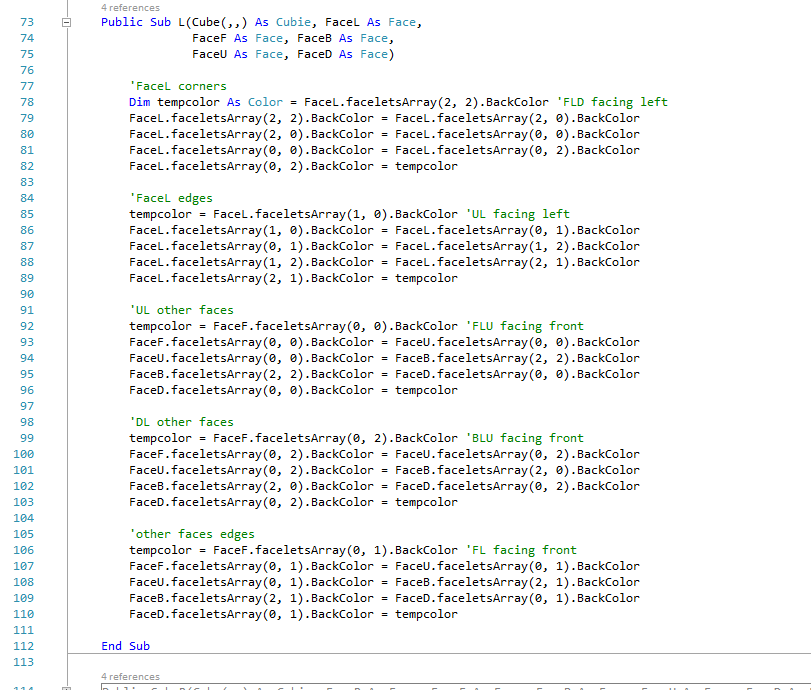


This calls a subroutine so that the array CubiesColours can be populated with colours by position.

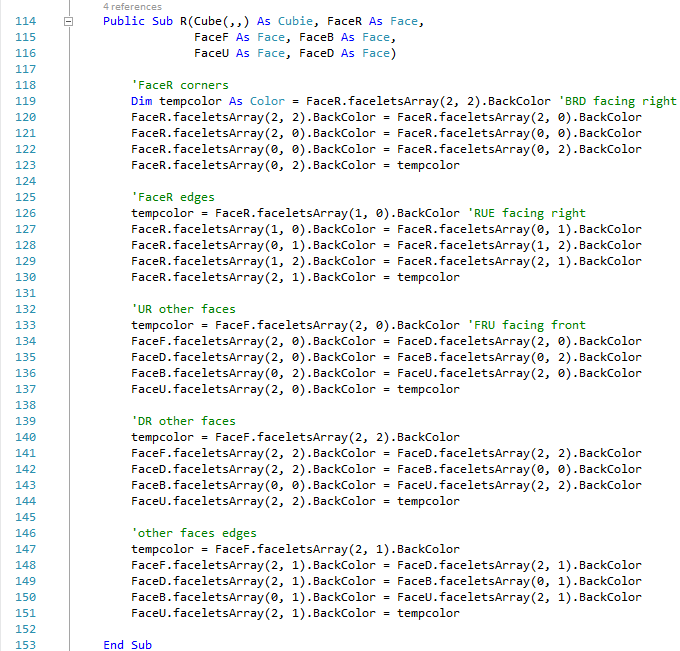
### AssignCubiesArray



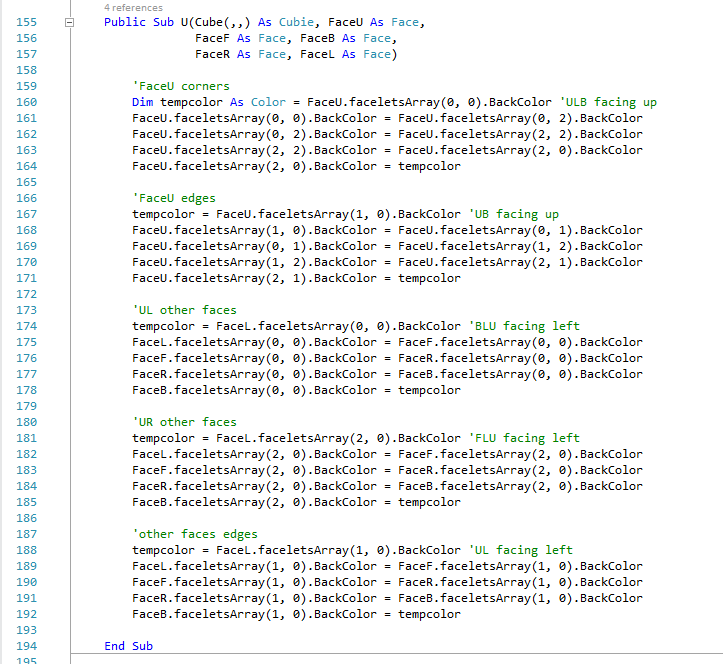
### L



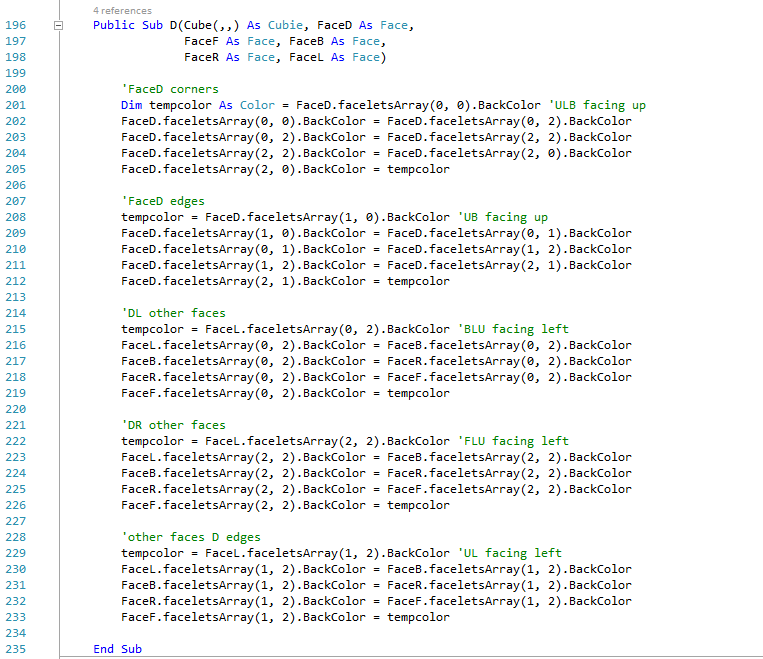
### R



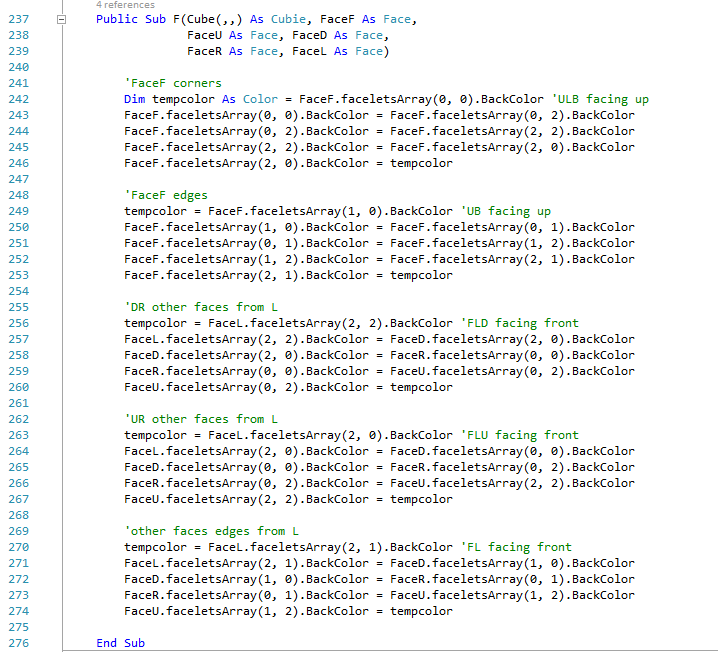
### U



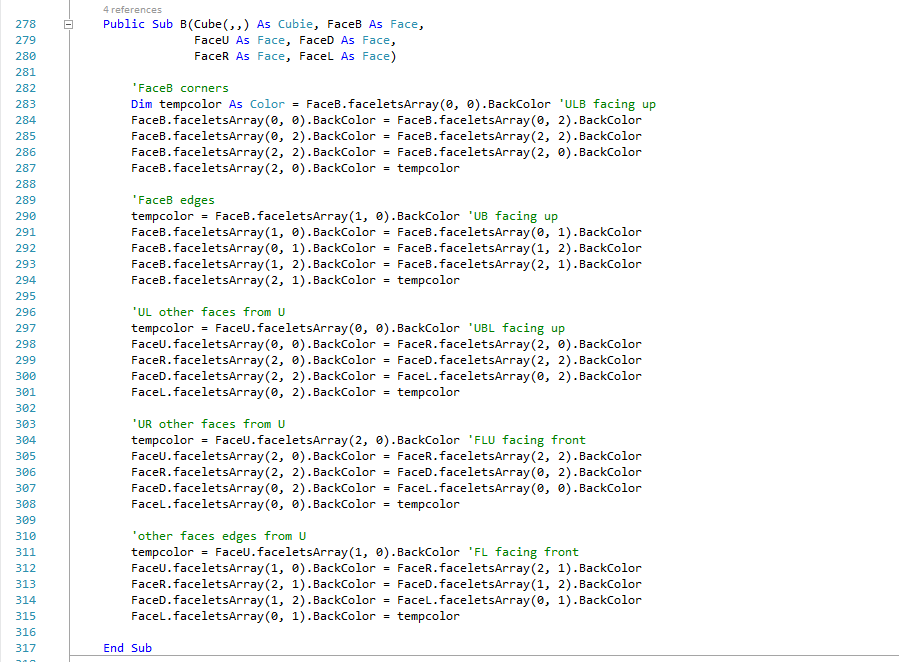
### D



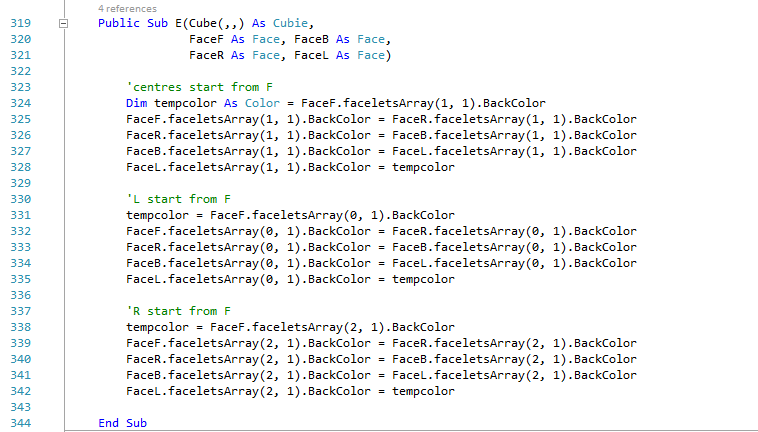
### F



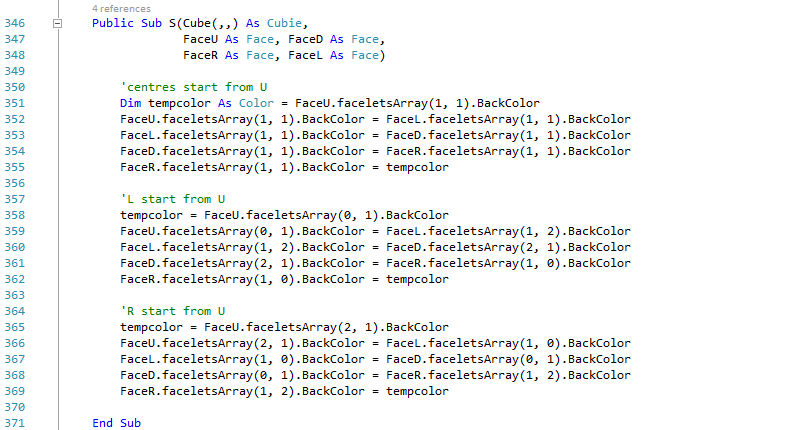
### B



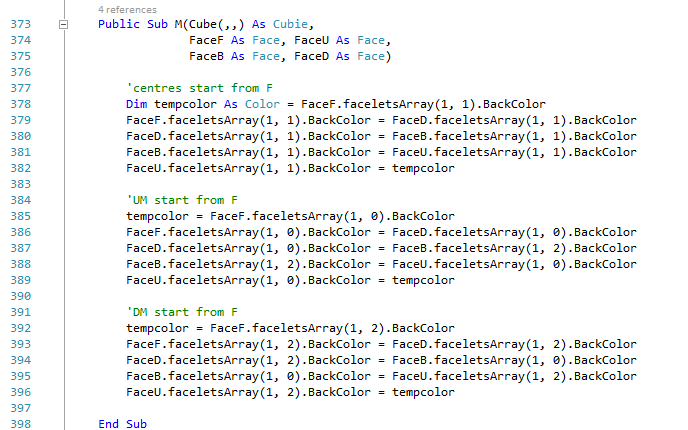
### E



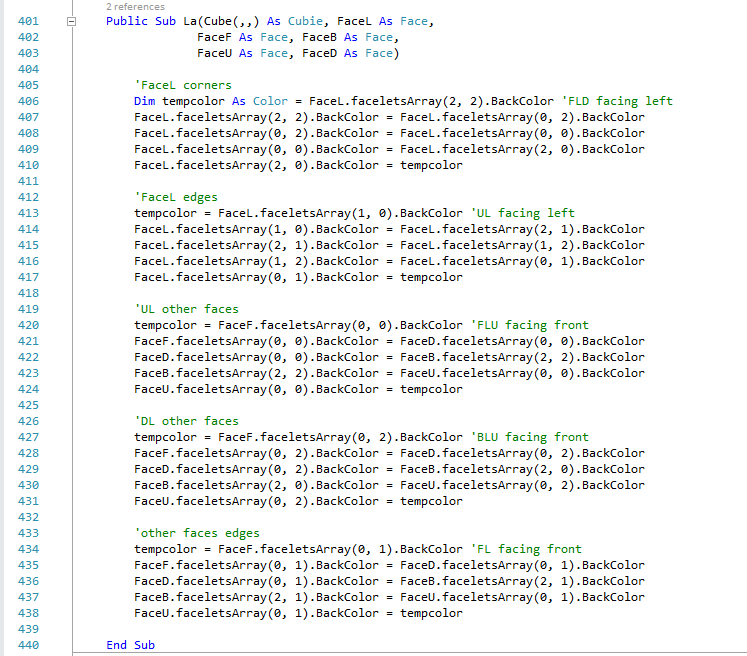
### S



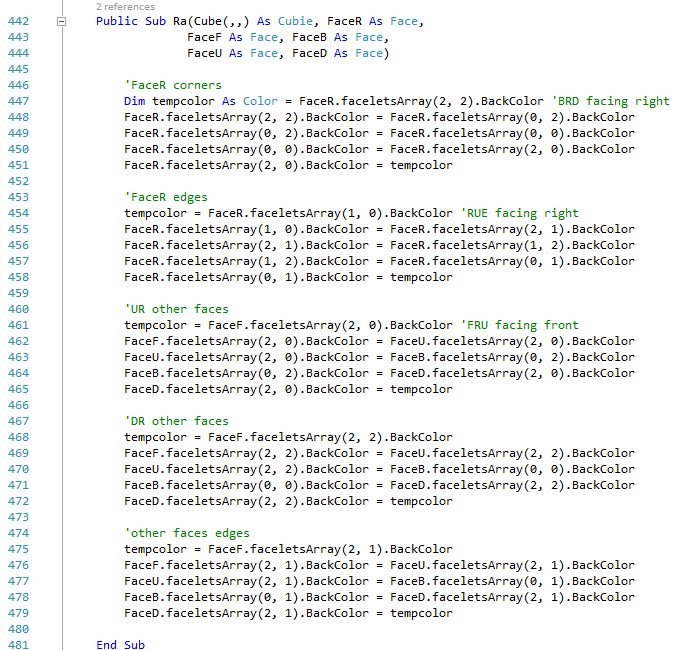
### M



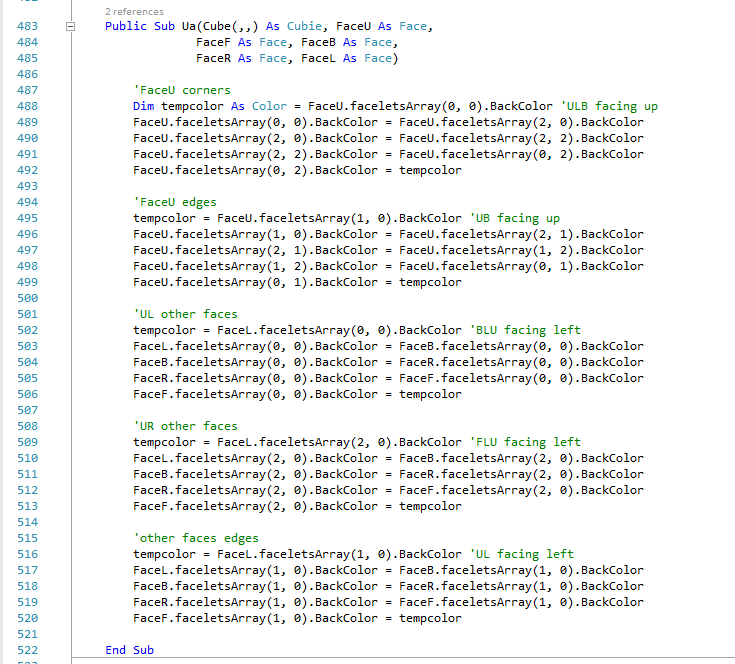
### La



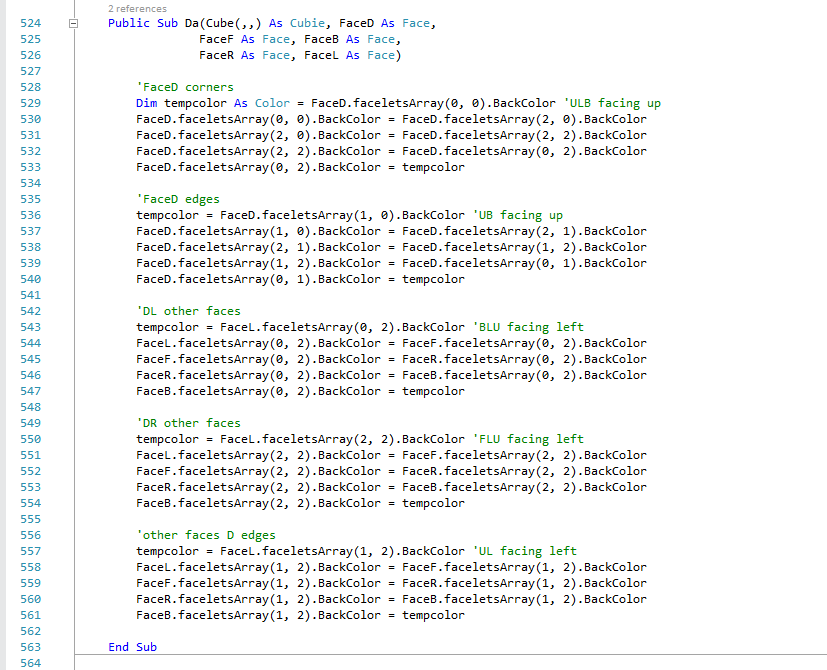
### Ra



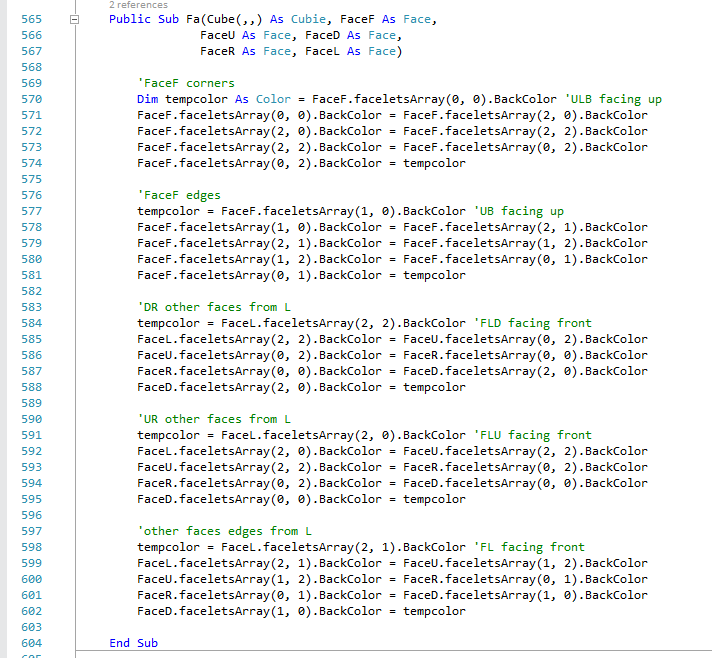
### Ua



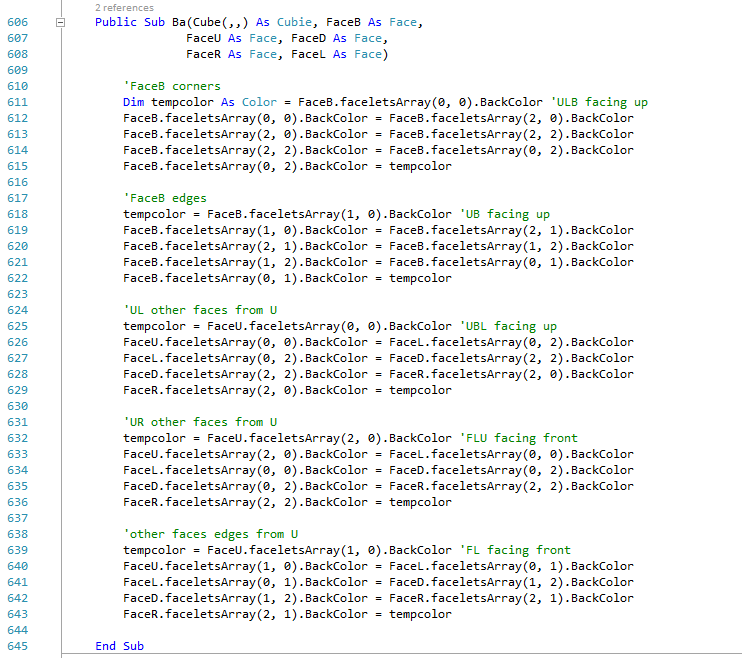
### Da



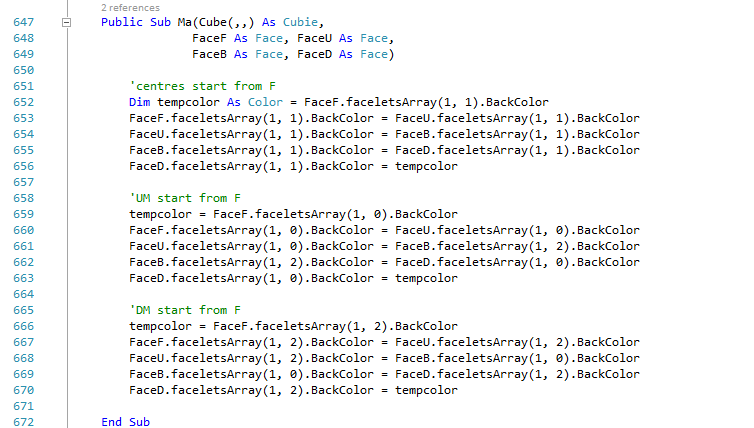
### Fa



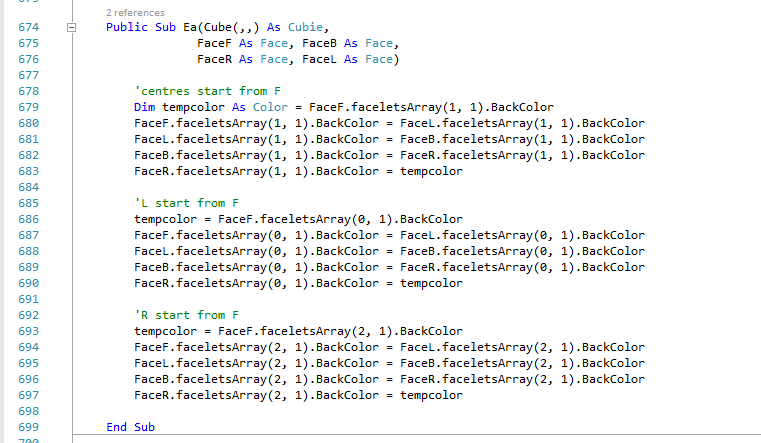
### Ba



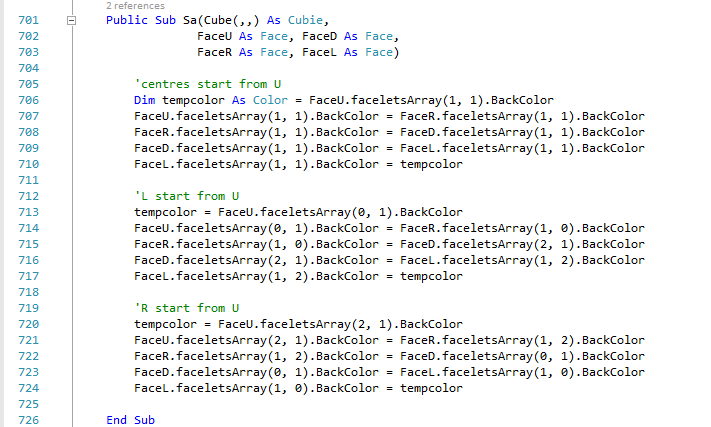
### Ma



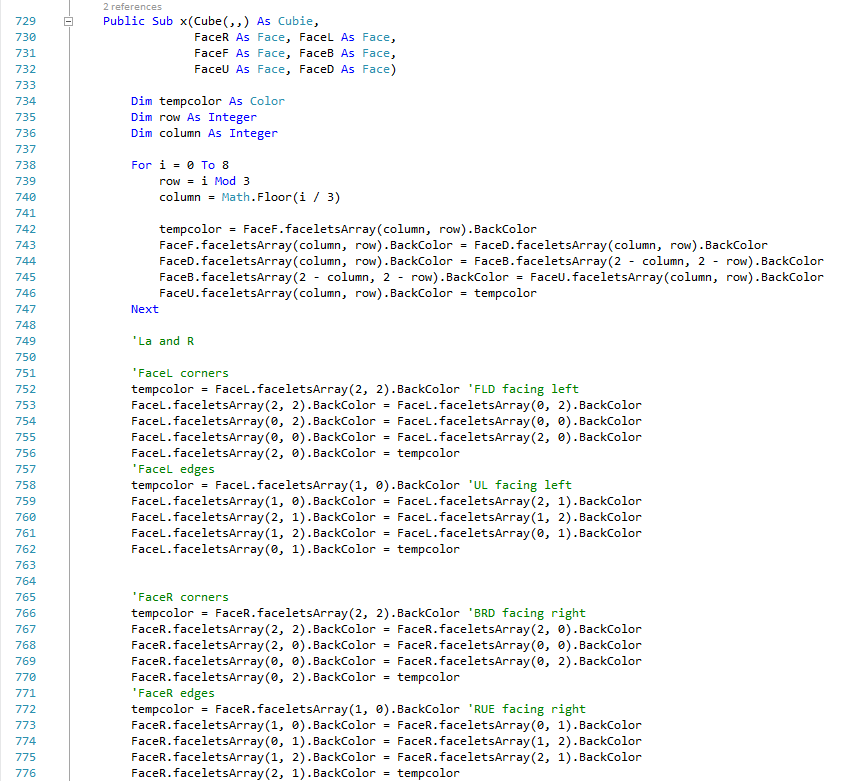
### Ea

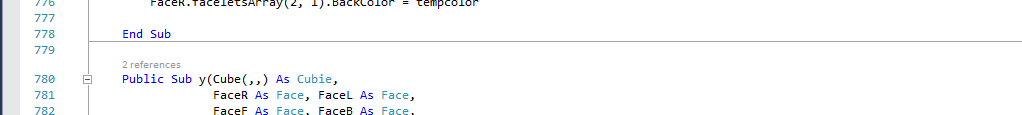


### Sa

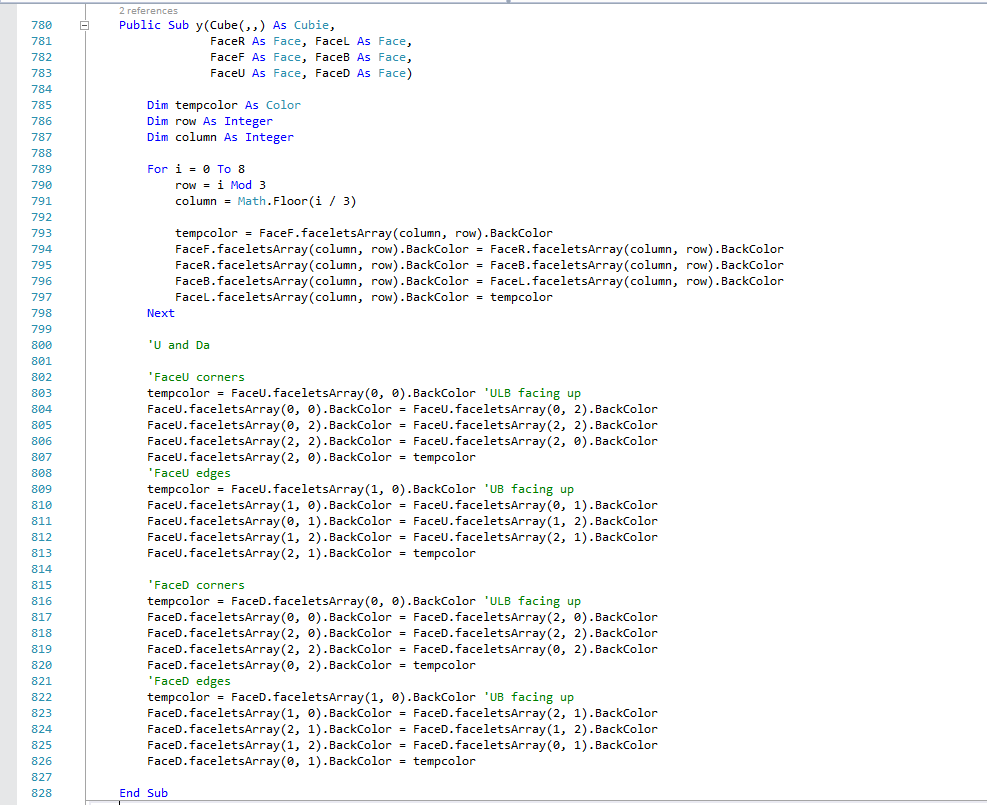


### x

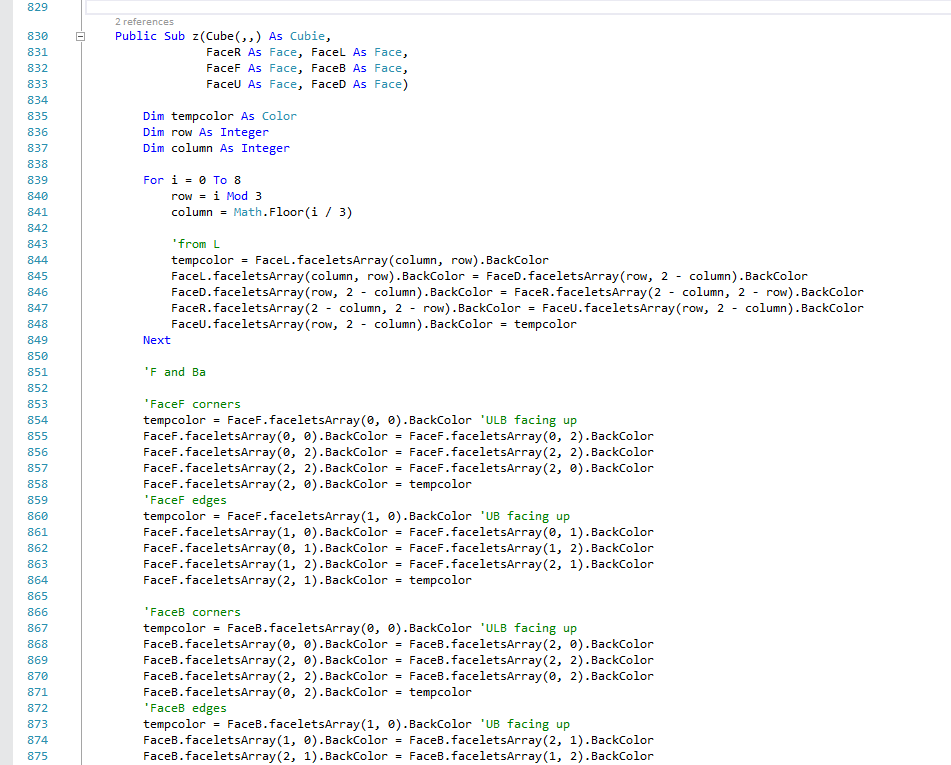


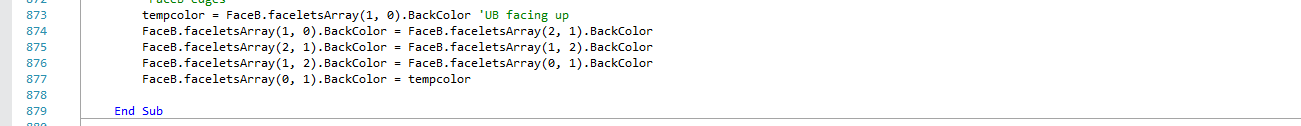


### y

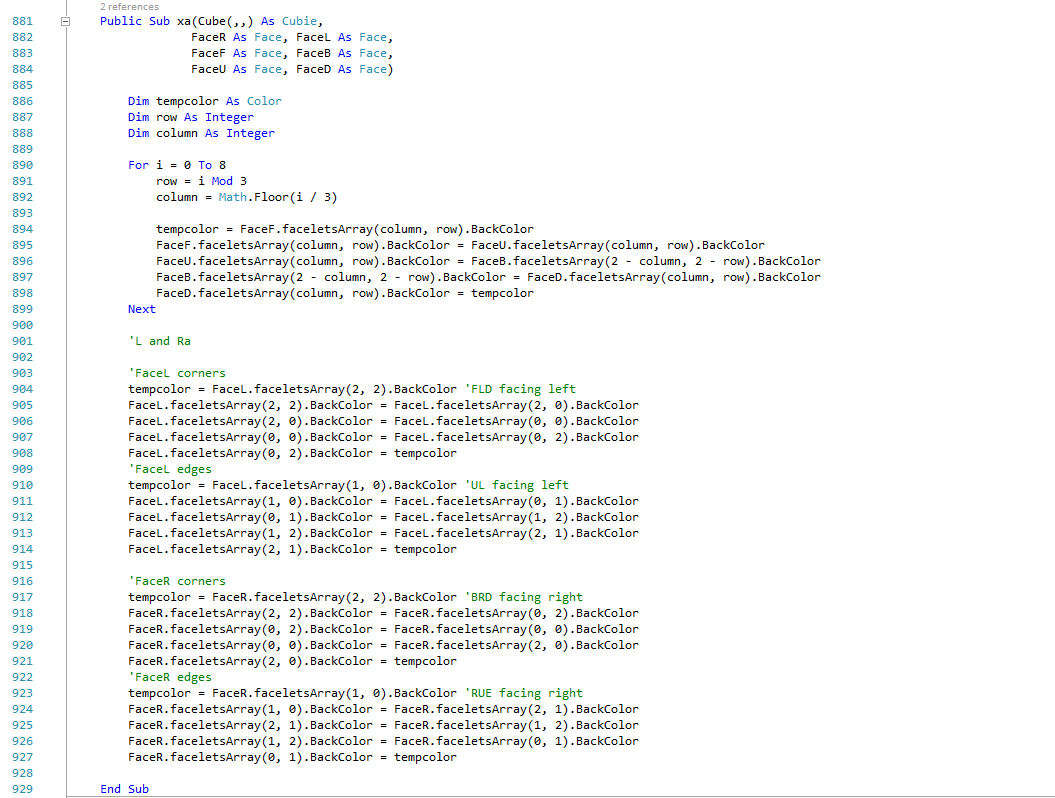


### z

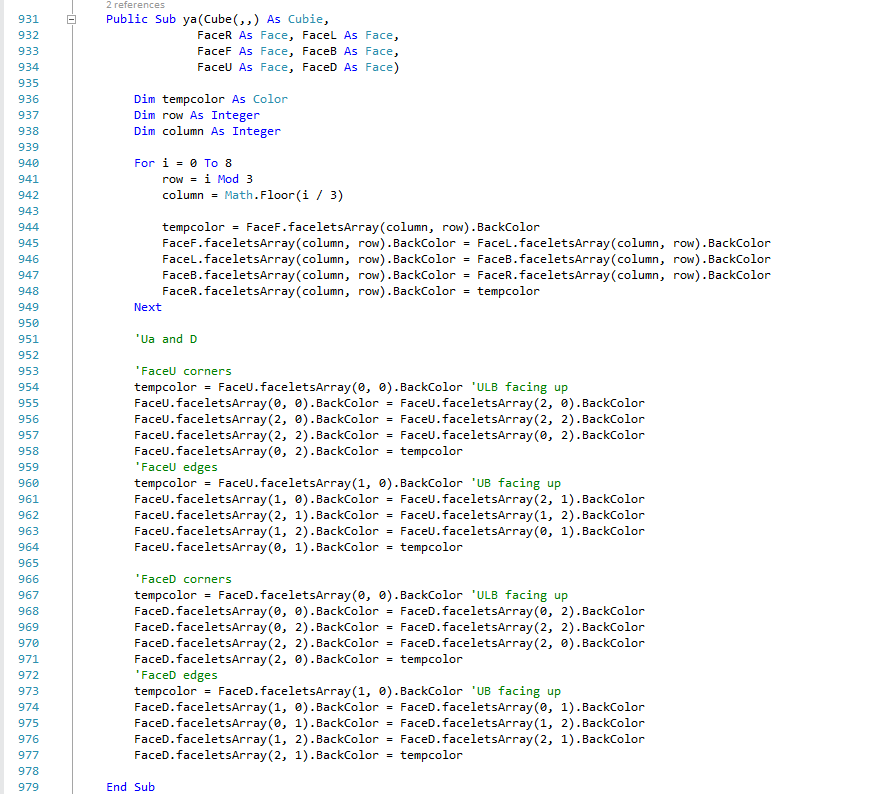




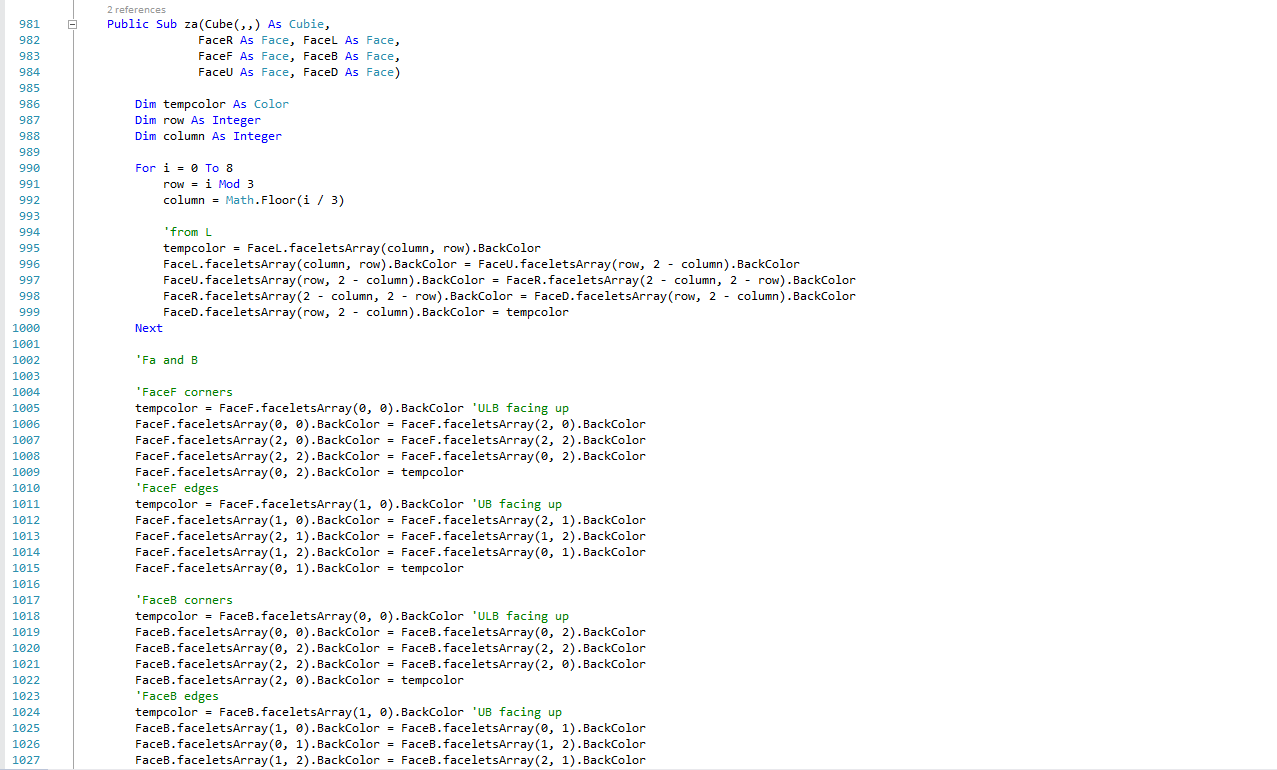
### xa

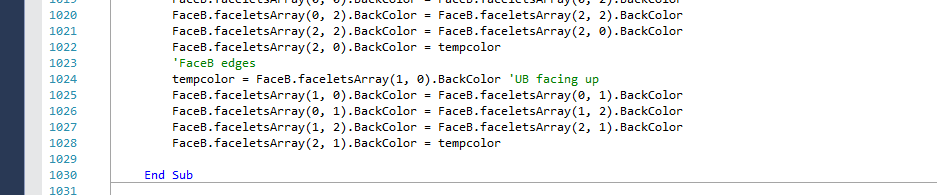


### ya

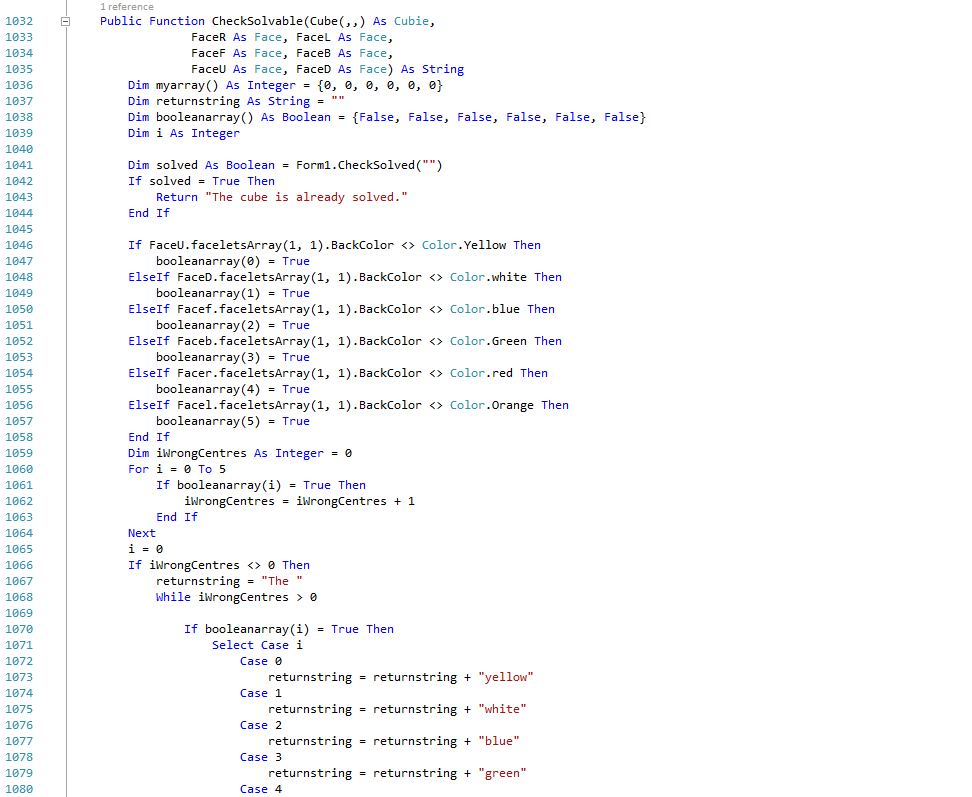
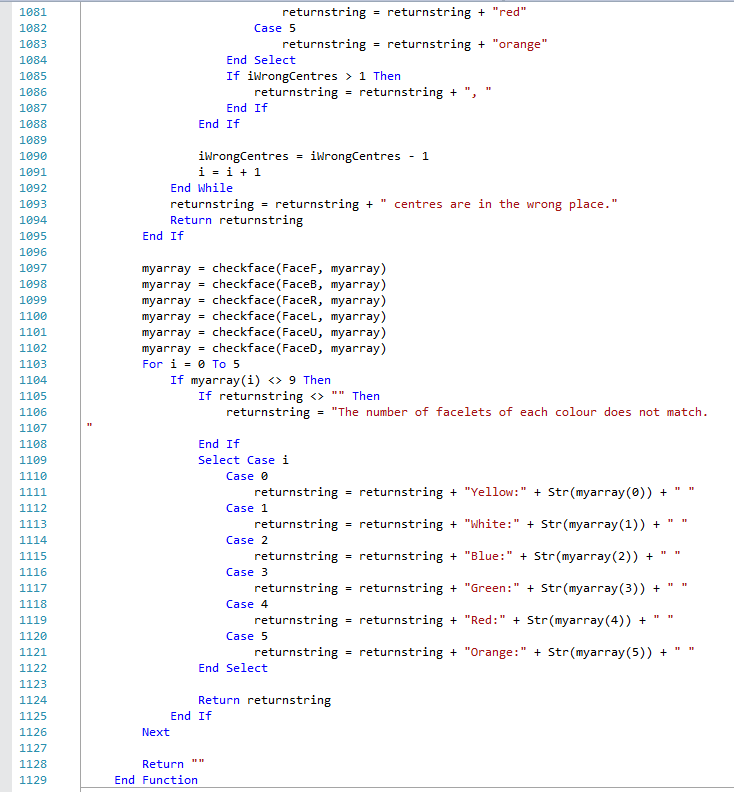


### za





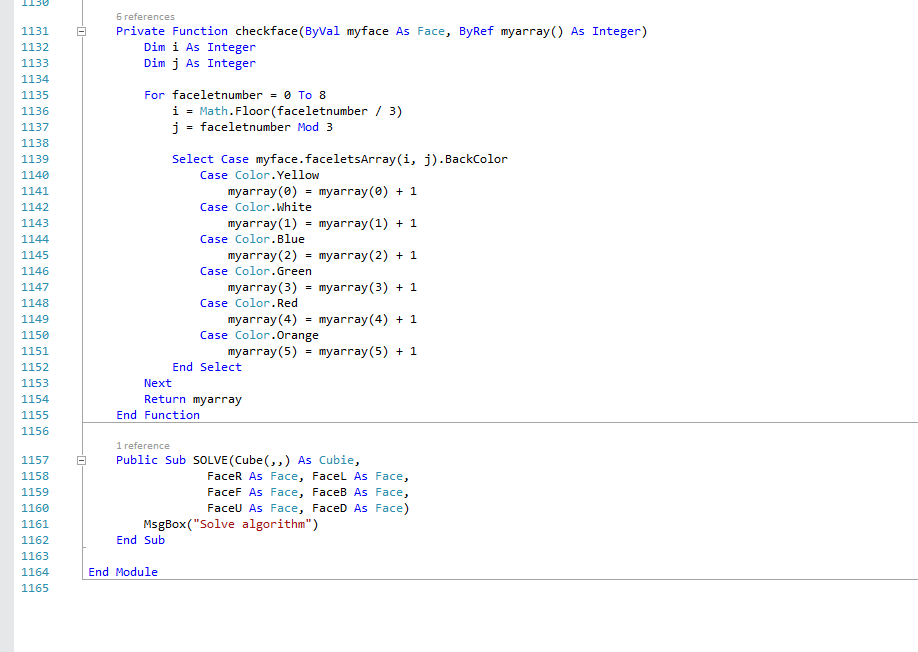
### CheckSolvable

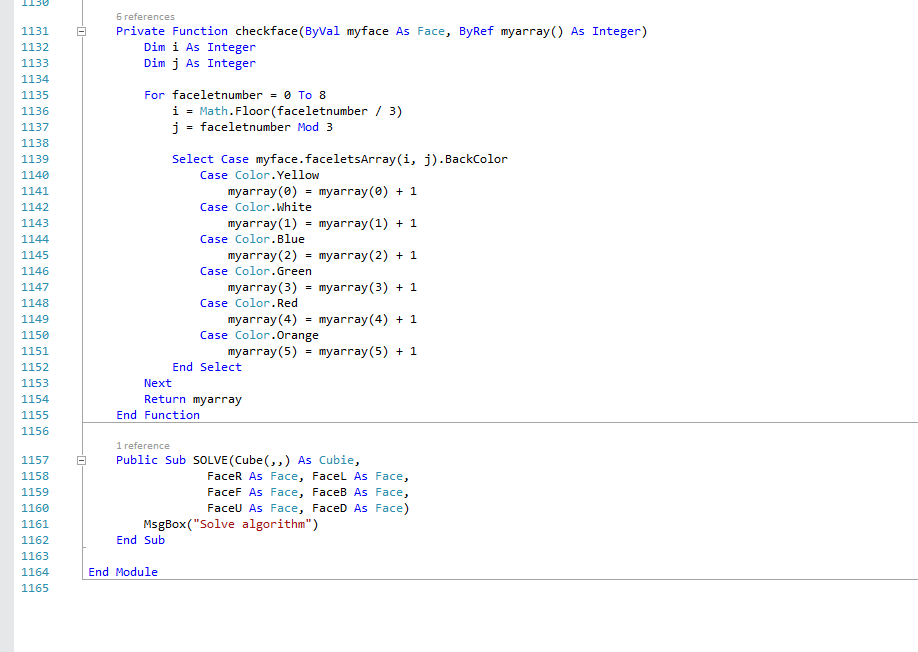
The returns on lines 1043 and 1094 go back to the previous subroutine so that time is not wasted in the next steps of checking if the cube is solvable.

The returns on lines 1043 and 1094 go back to the previous subroutine so that time is not wasted in the next steps of checking if the cube is solvable.

### checkface



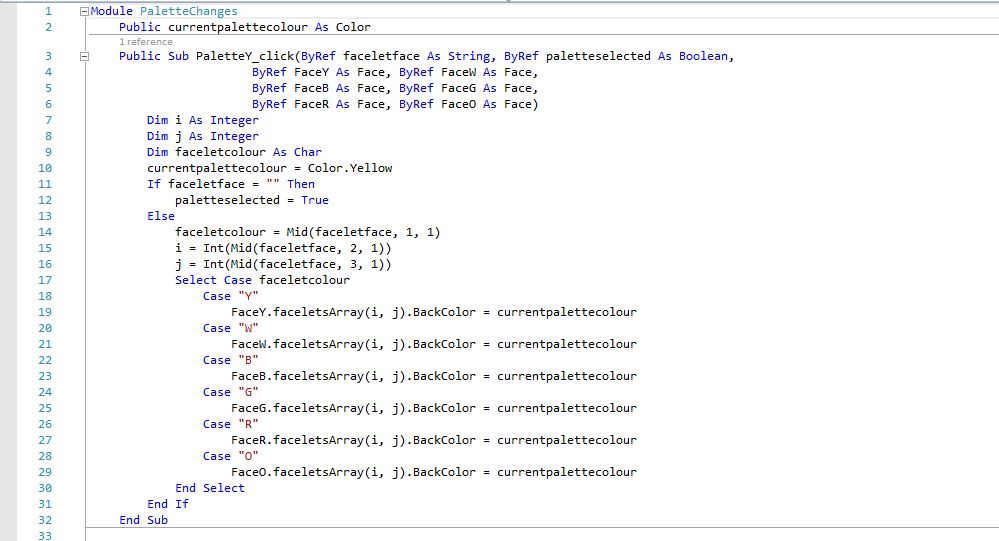
### SOLVE



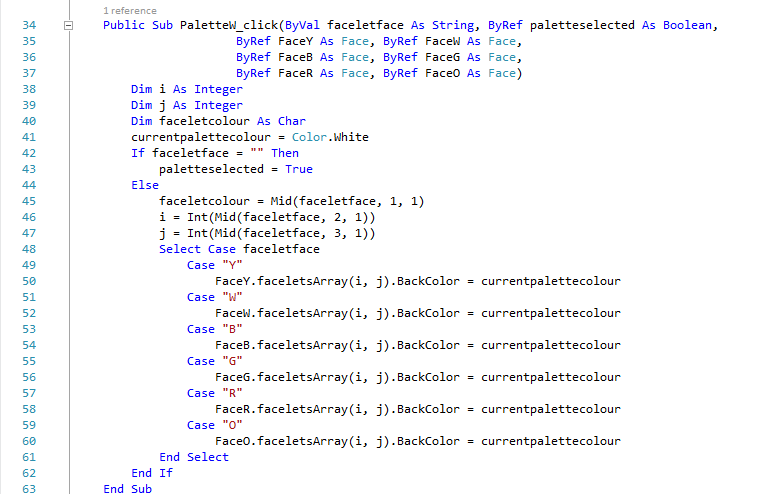
## PaletteChanges



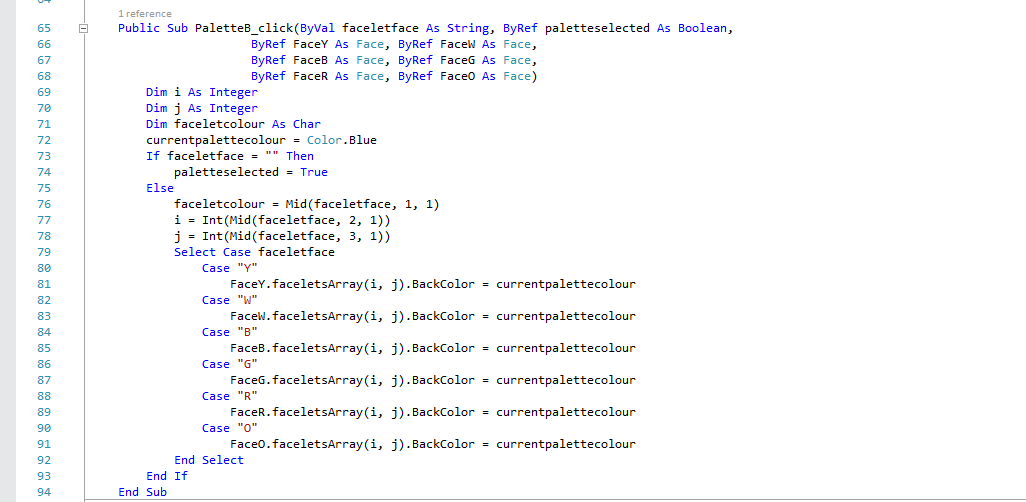
### PaletteY\_click



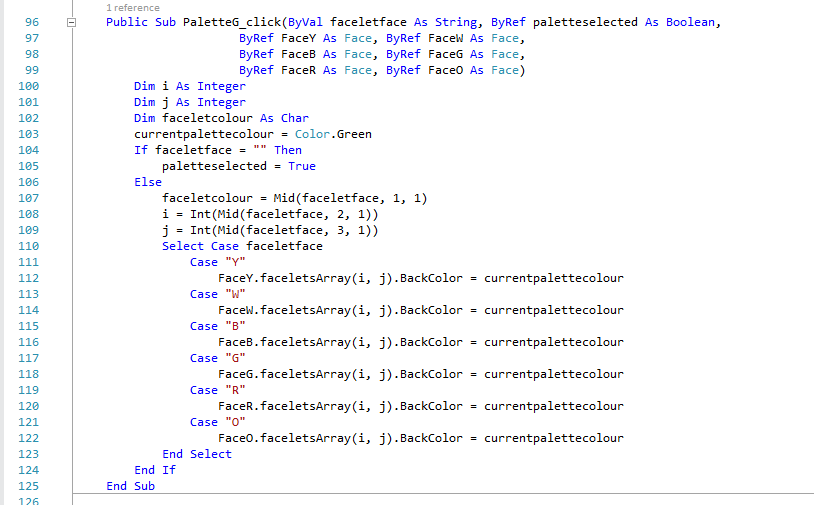
### PaletteW\_click



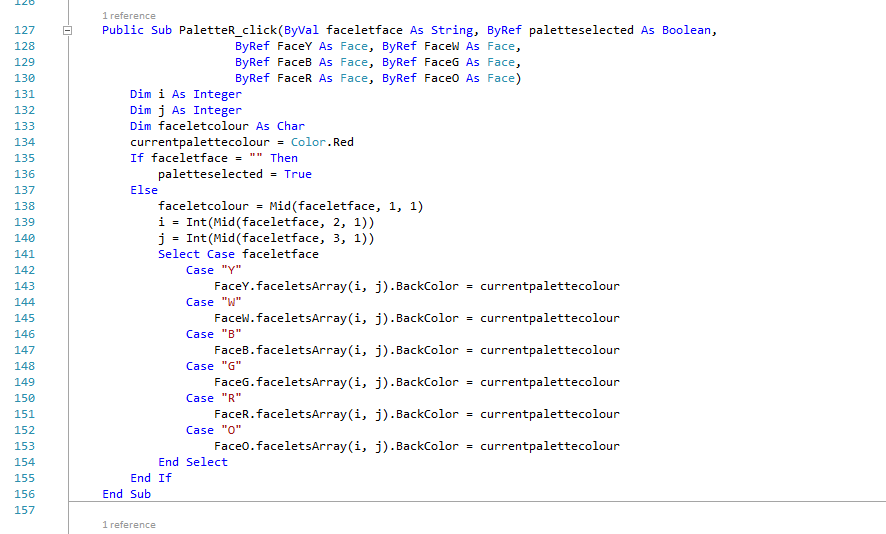
### PaletteB\_click



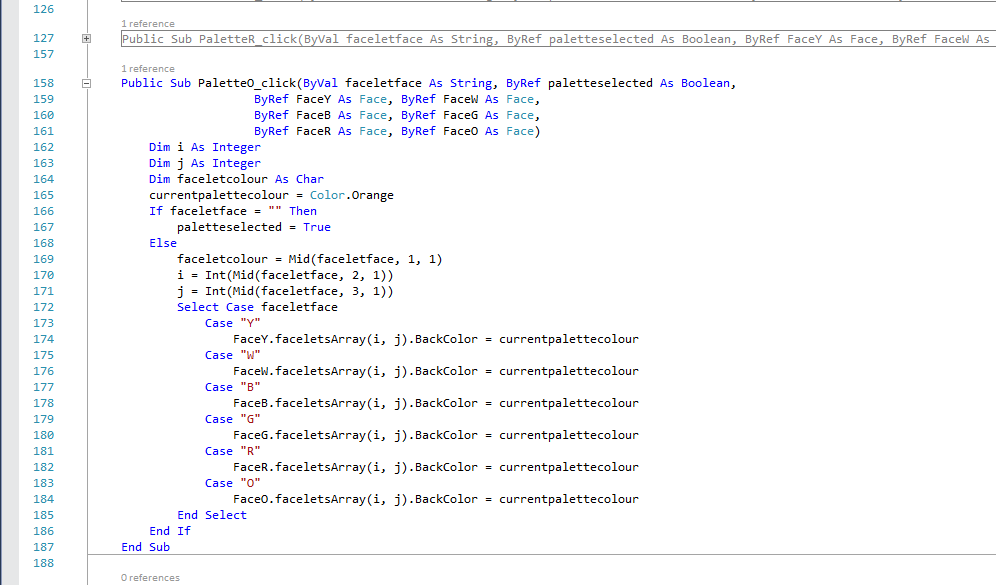
### PaletteG\_click



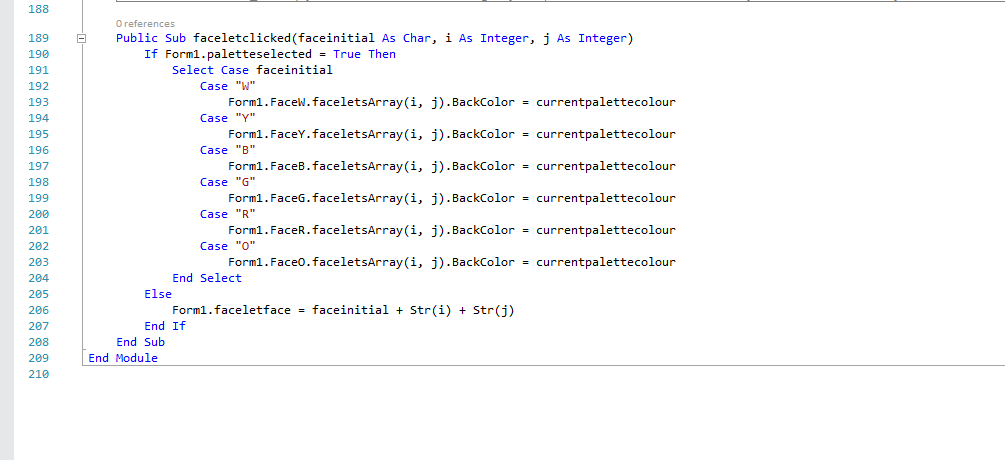
### PaletteR\_click



### PaletteO\_click

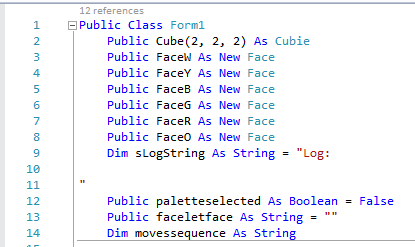


### faceletclicked



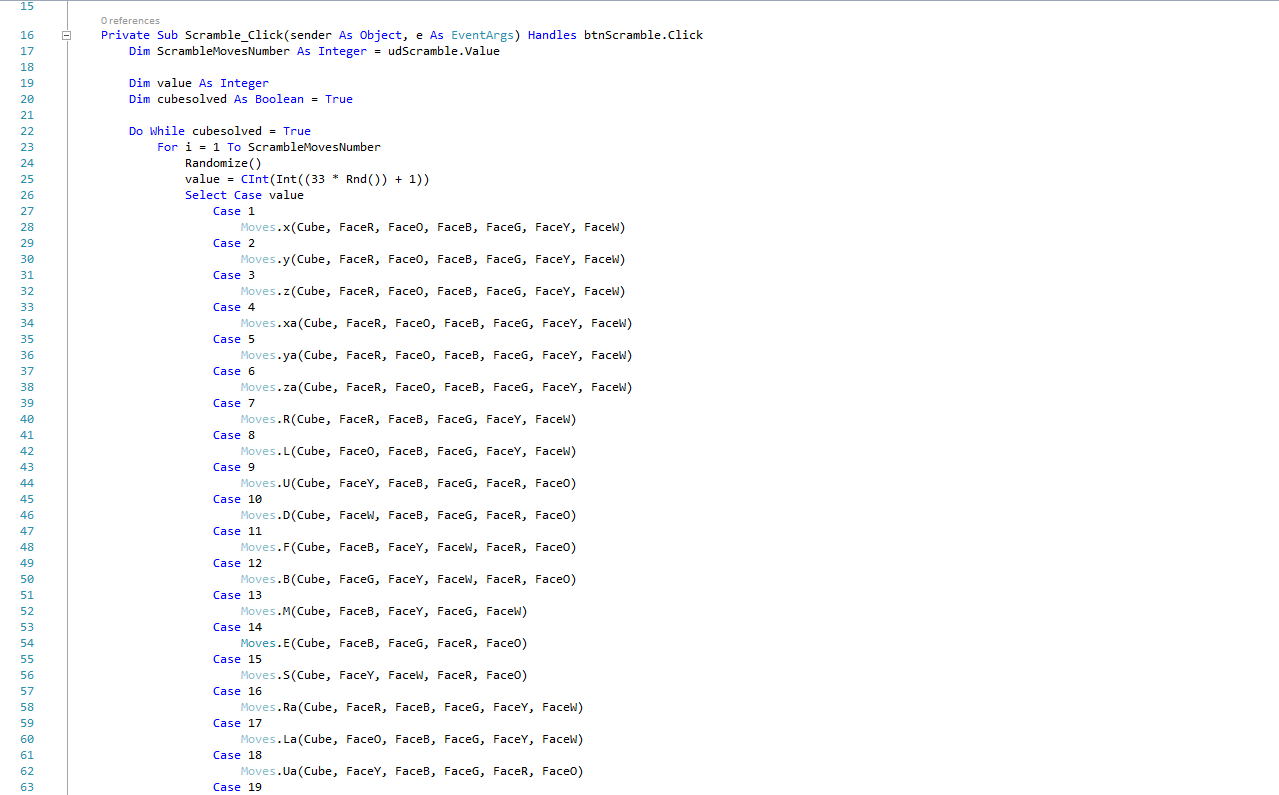
## Form1

Throughout the subroutines in Form1, they contribute to sLogString, which is displayed when the user clicks on the “Log2 button.

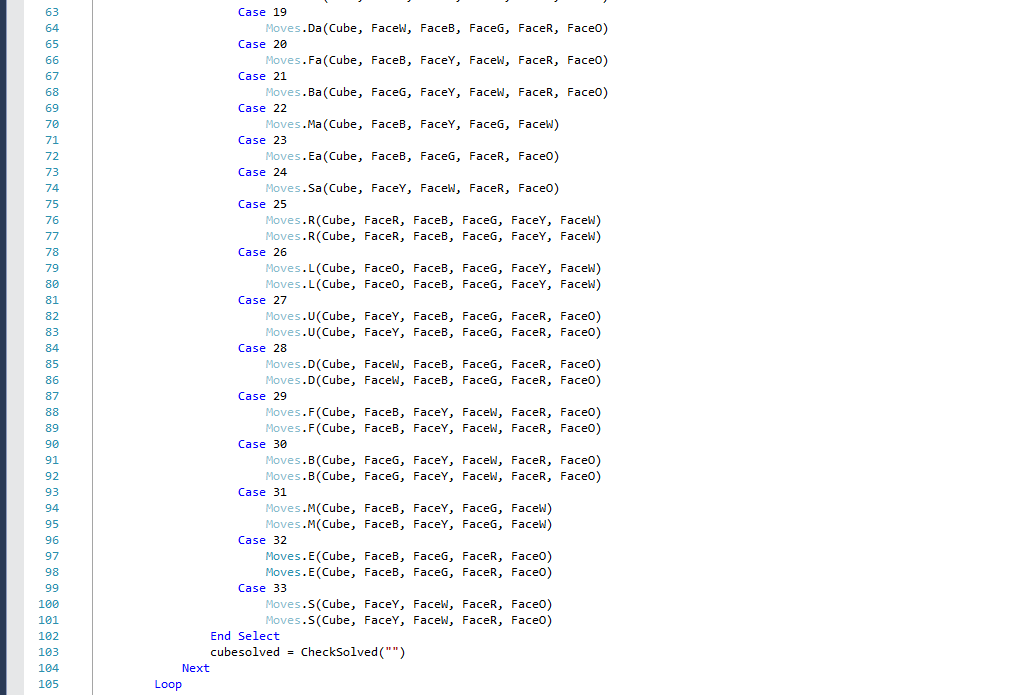


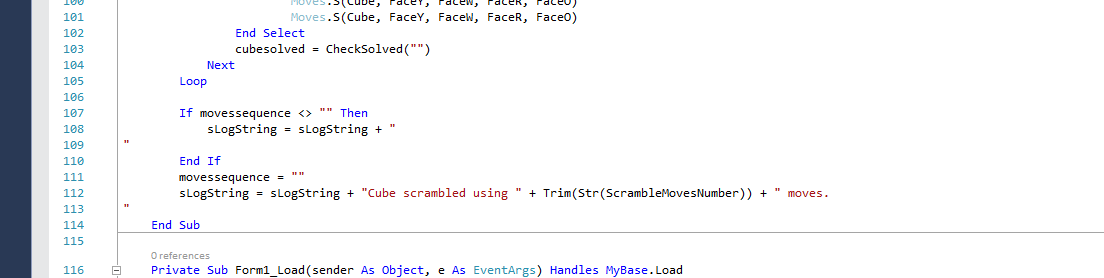
On line 2, the Cube is instantiated at run-time and makes use of OOP composition (Group A).

### Scramble\_Click

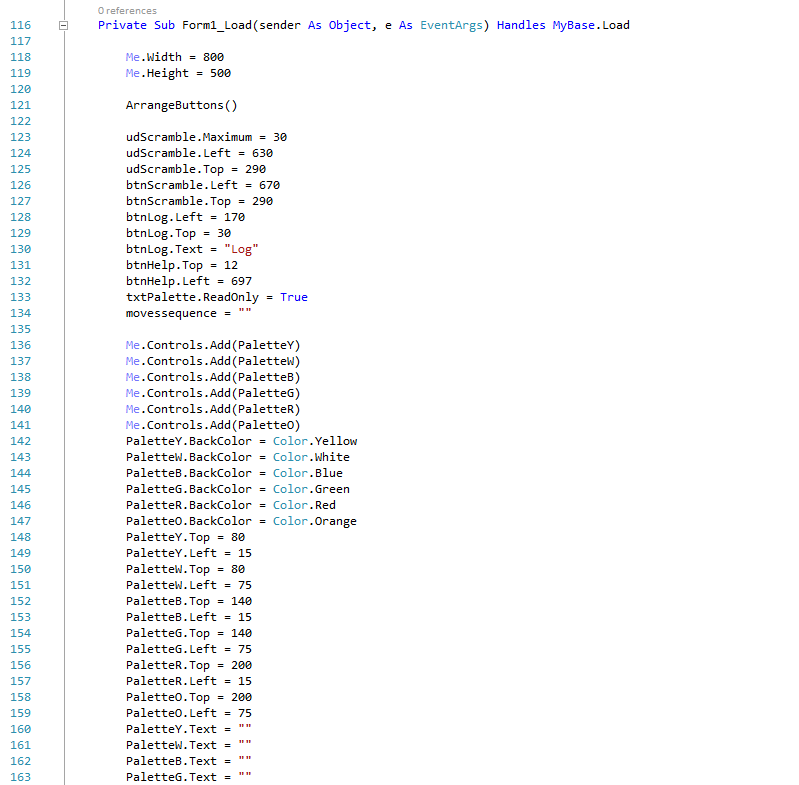


Line 17 uses OOP to get the integer value in the up-down control.



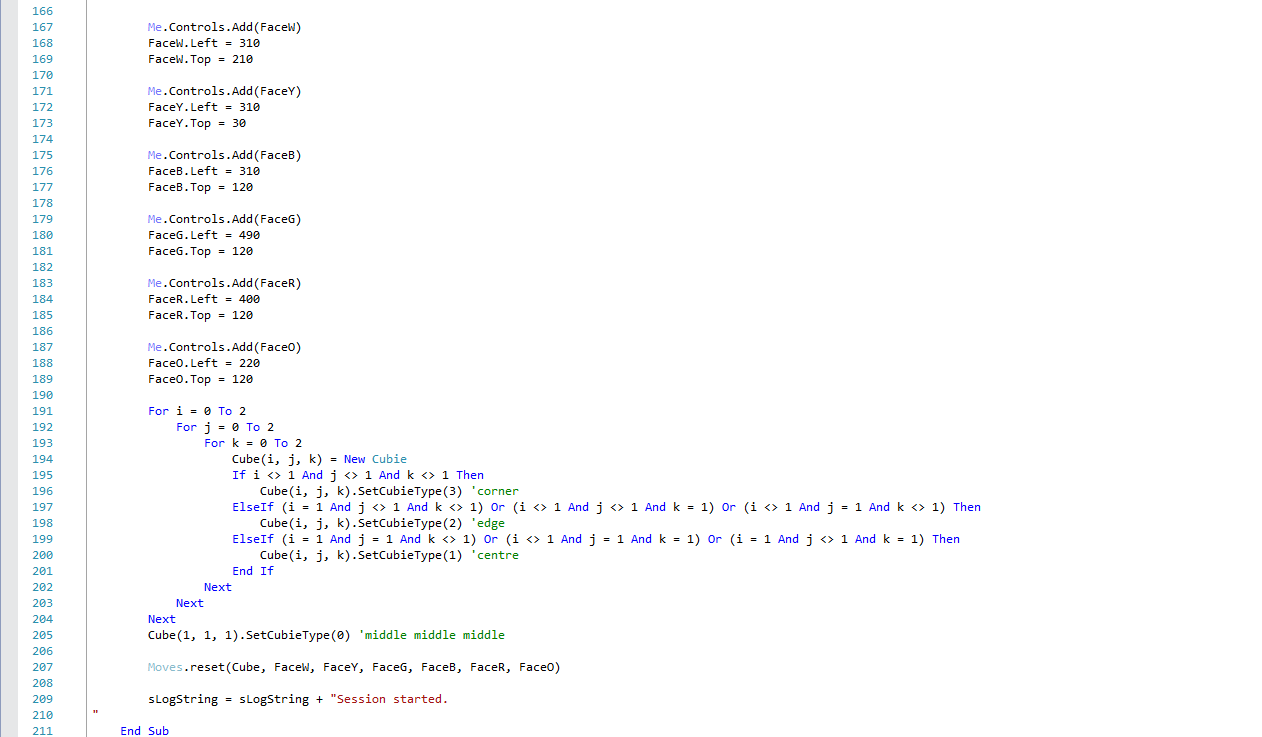


### Form1\_Load



Although the Form1\_Load subroutine is already pretty crowded with code, I think making a separate subroutine for the moves buttons was worth it, and prevents more code being in the Form1\_Load subroutine..





### btnReset\_Click



### L\_Click



### R\_Click



### U\_Click



### D\_Click



### F\_Click



### B\_Click



### M\_Click



### E\_Click



### S\_Click



### btnLa\_Click



### btnRa\_Click



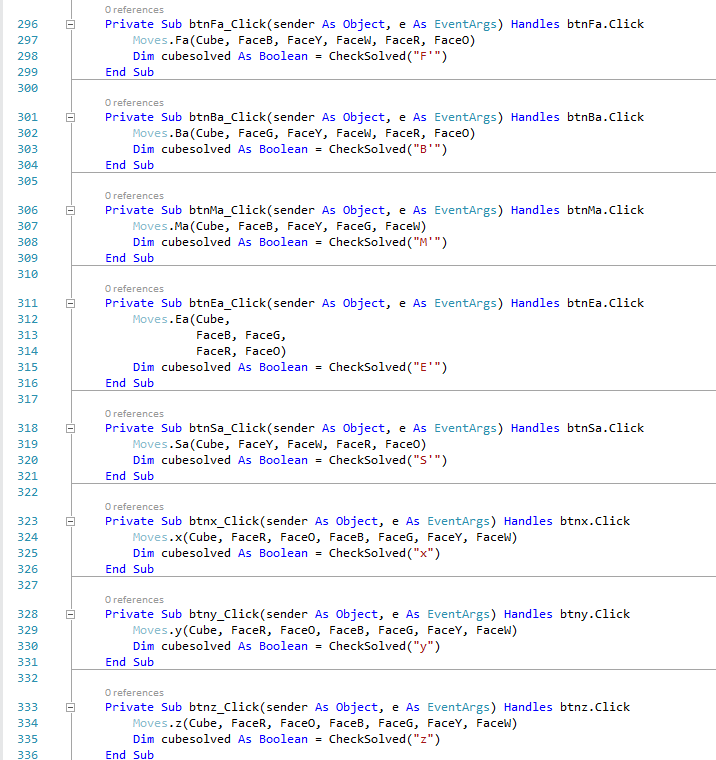
### btnUa\_Click



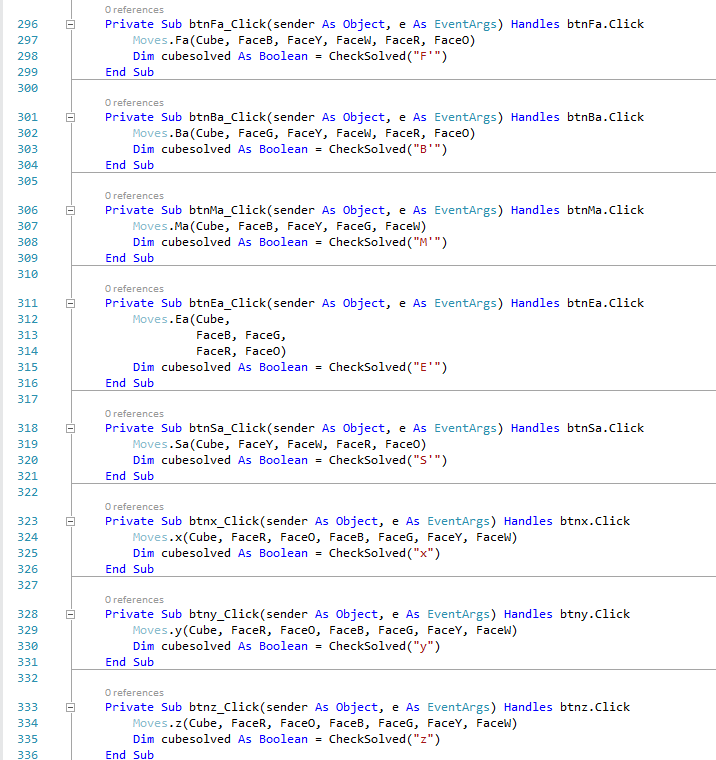
### btnDa\_Click



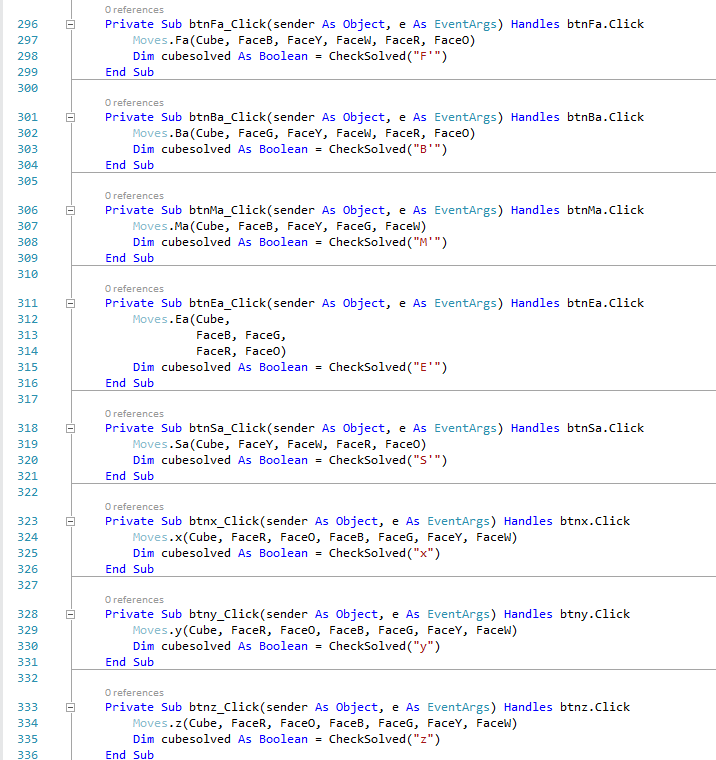
### btnFa\_Click



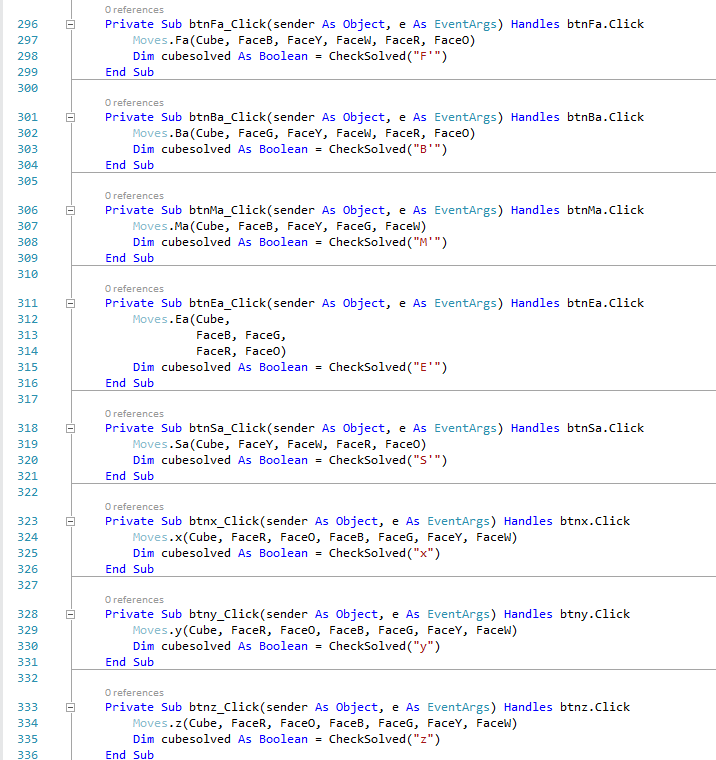
### btnBa\_Click



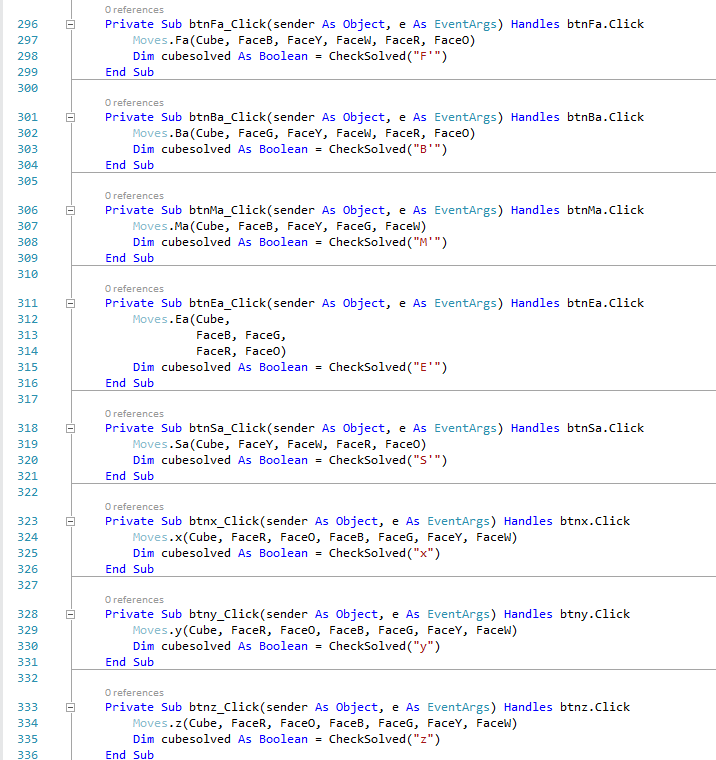
### btnMa\_Click



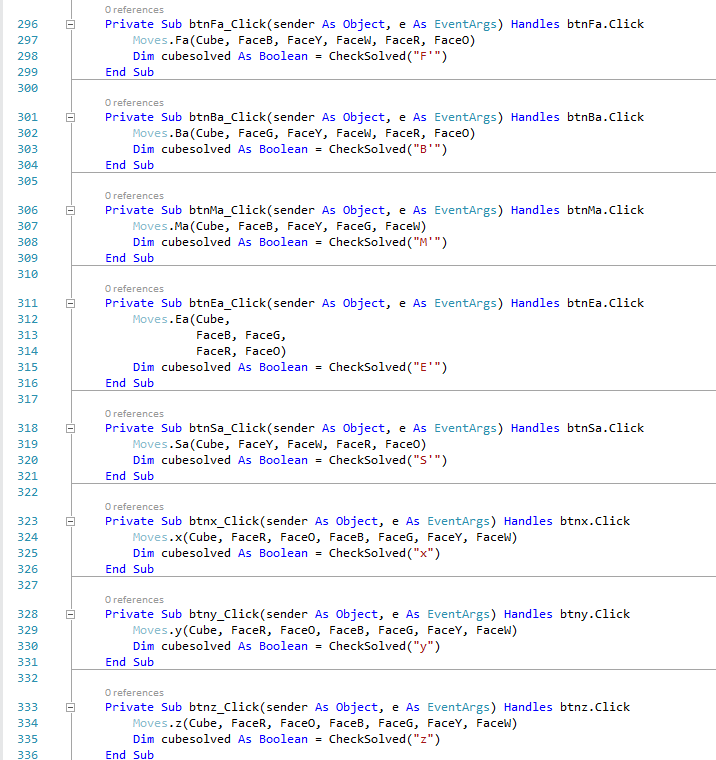
### btnEa\_Click



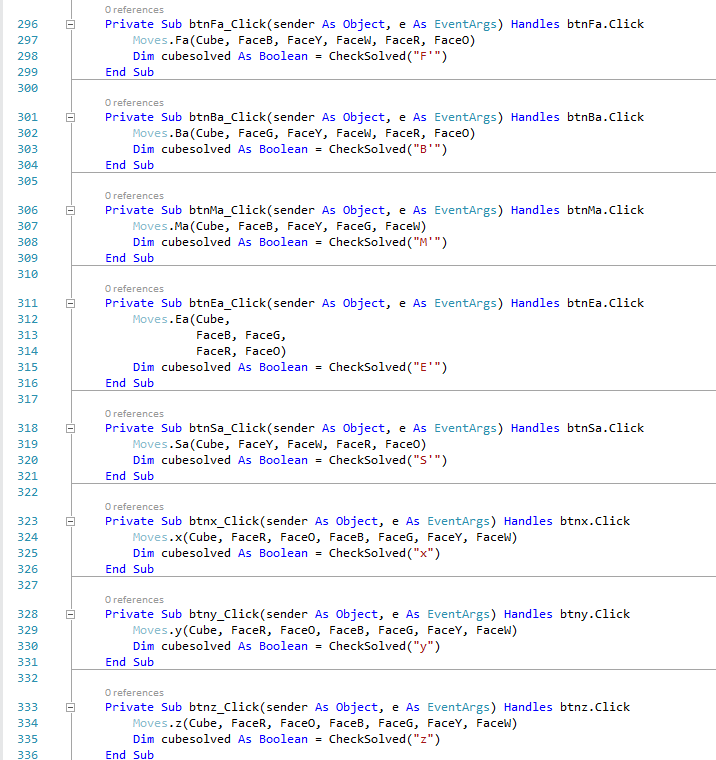
### btnSa\_Click



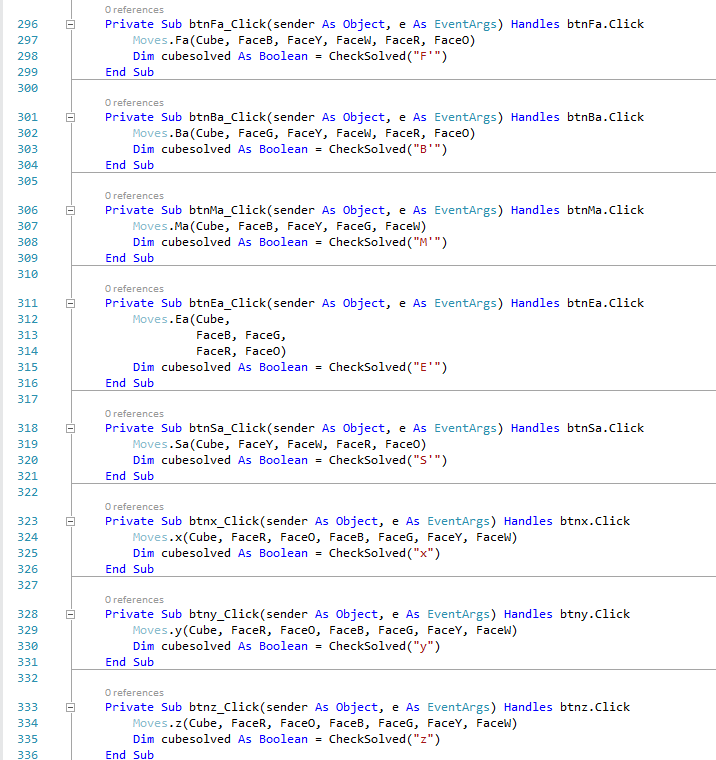
### btnx\_Click



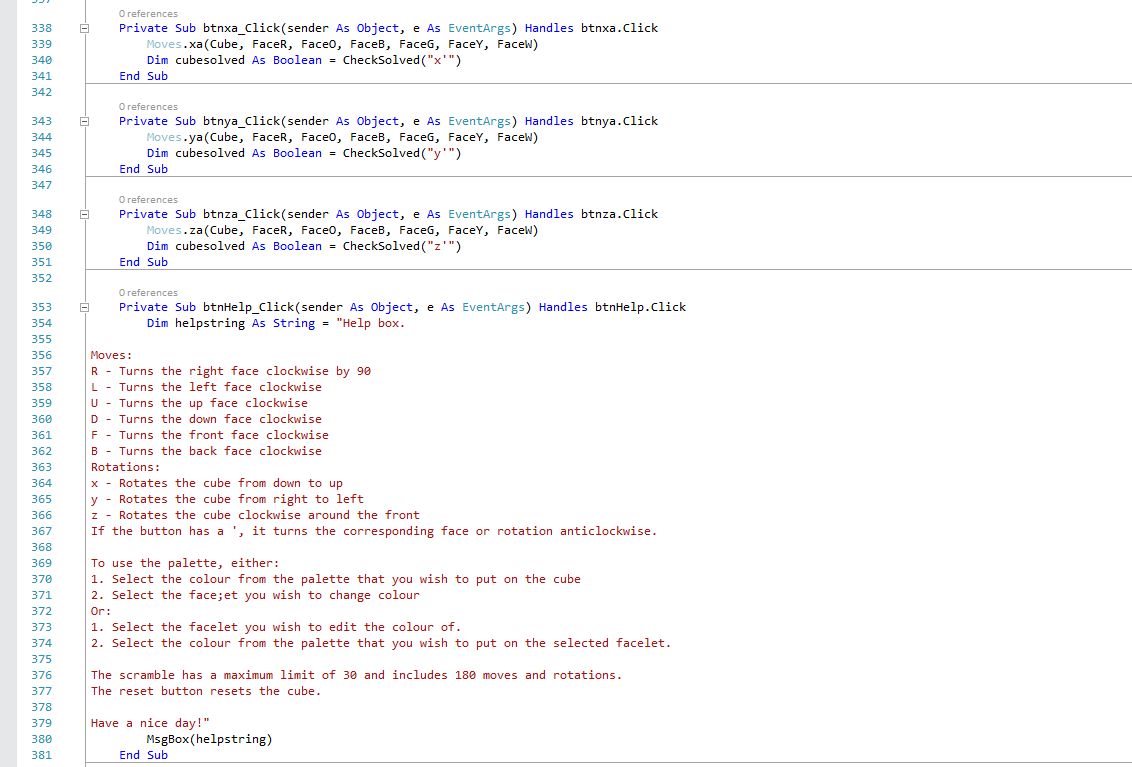
### btny\_Click



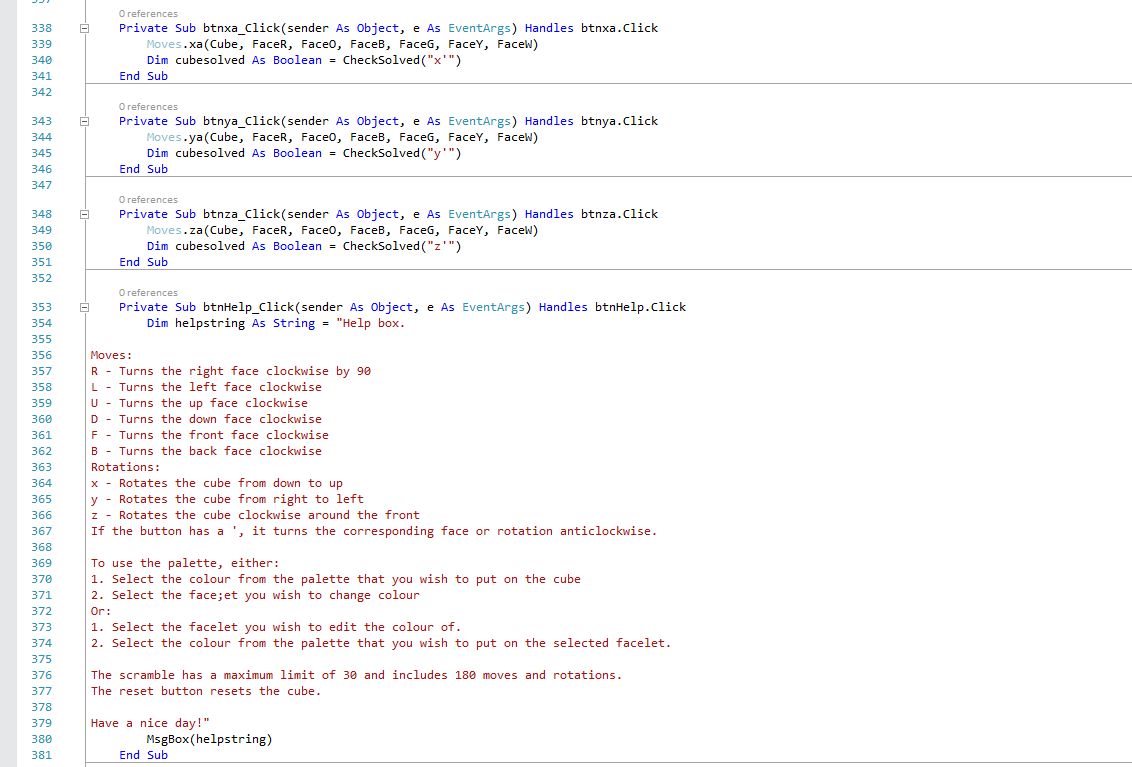
### btnz\_Click



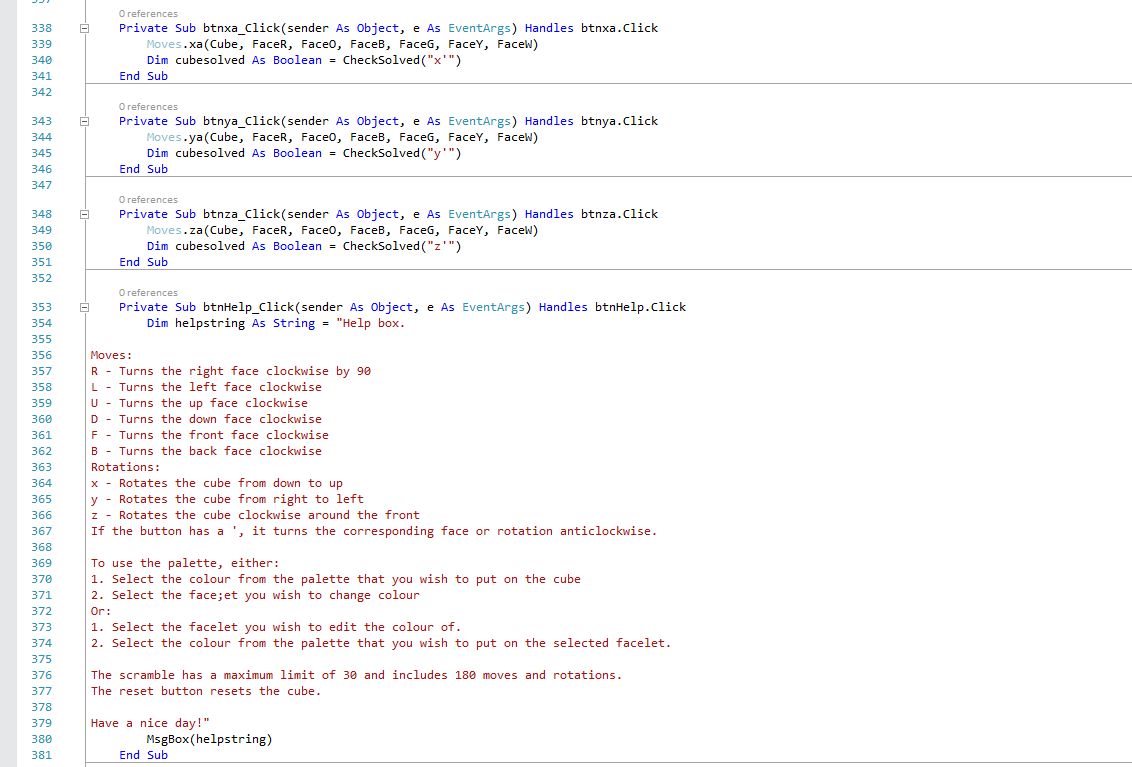
### btnxa\_Click



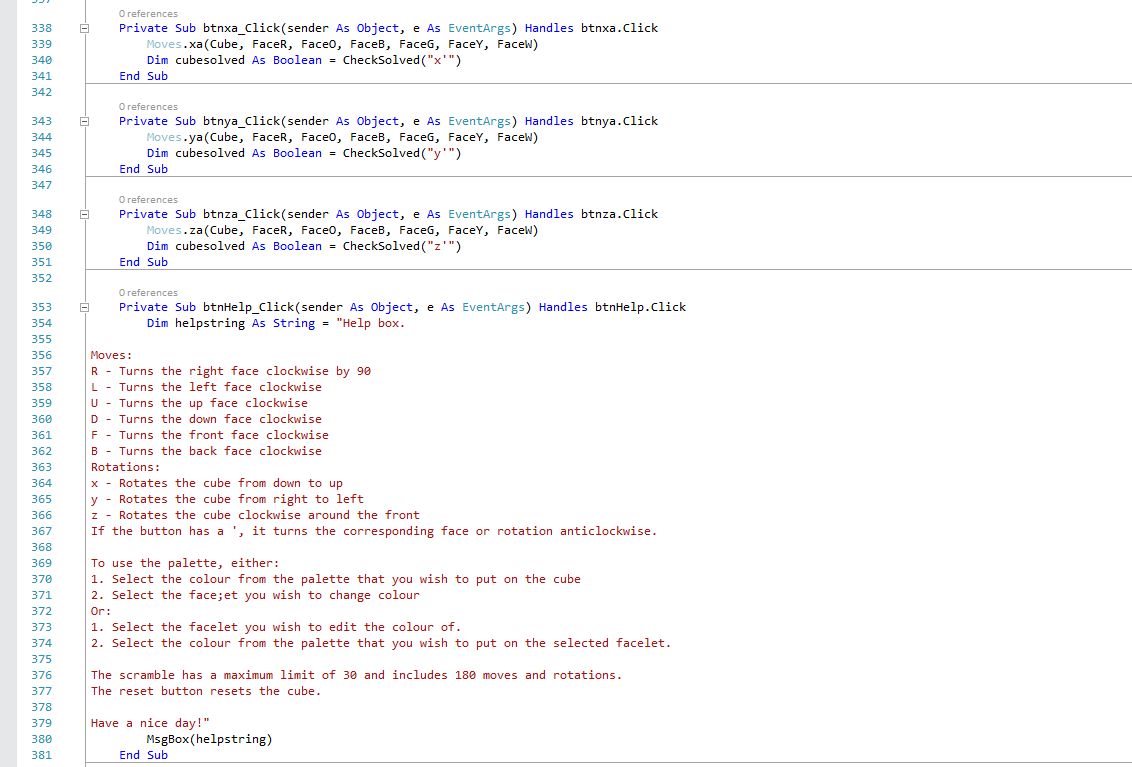
### btnya\_Click



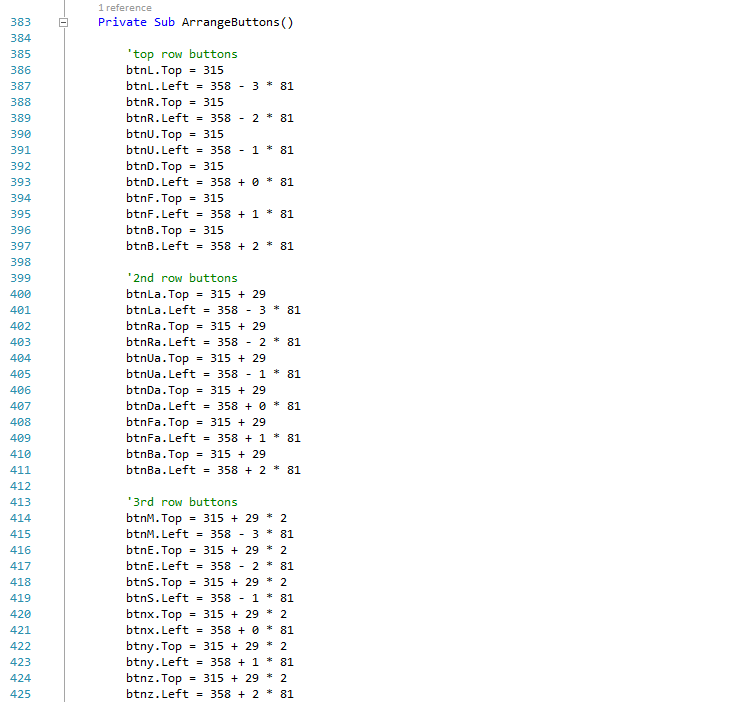
### btnza\_Click

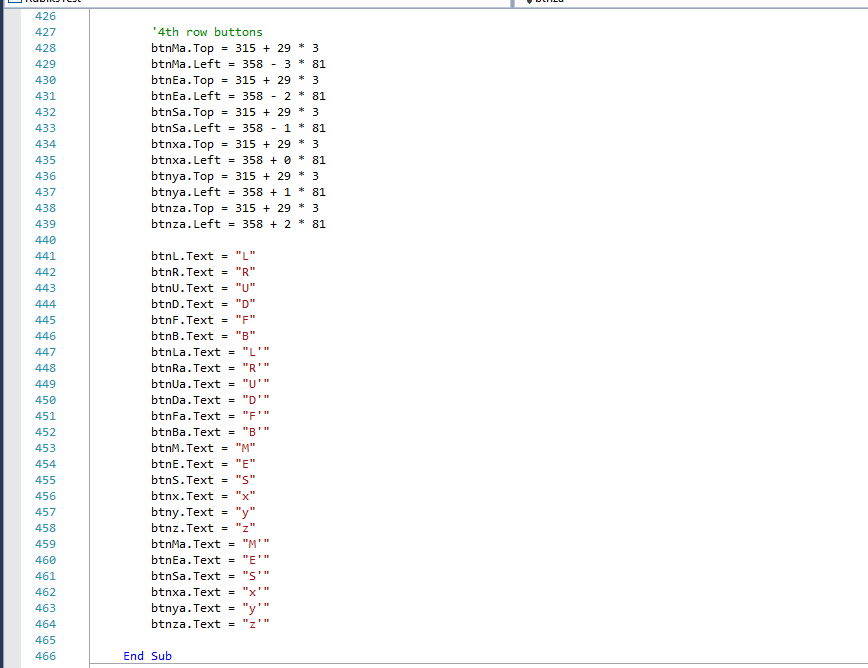


### btnHelp\_Click



### ArrangeButtons



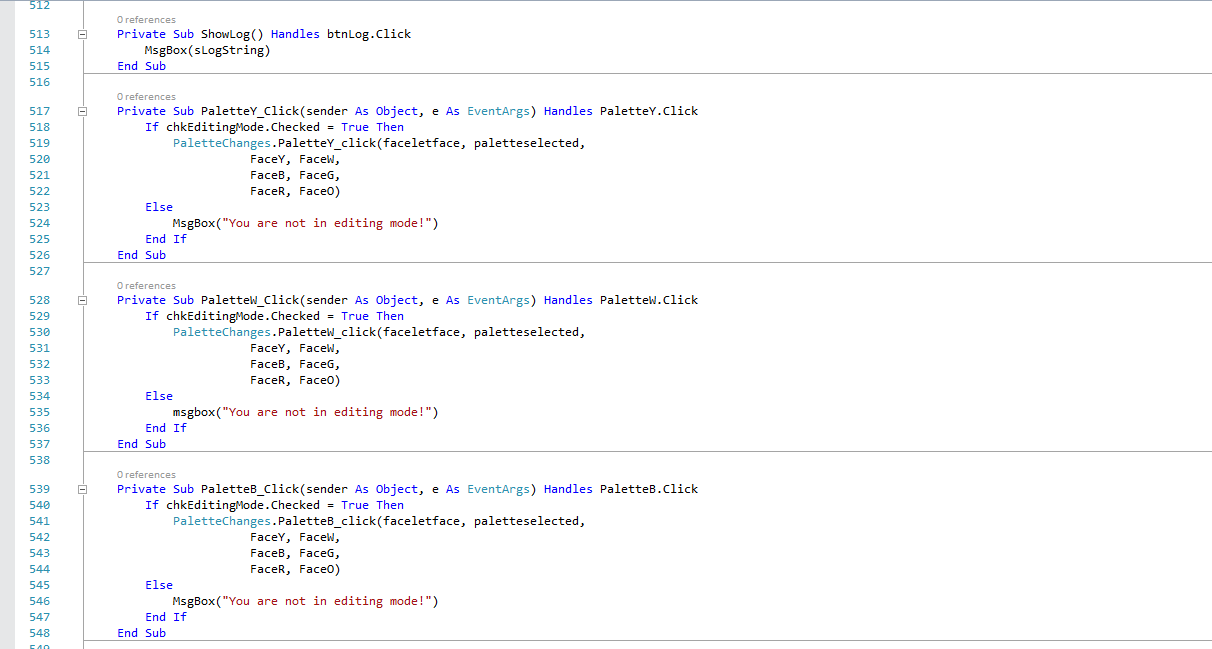


### CheckSolved

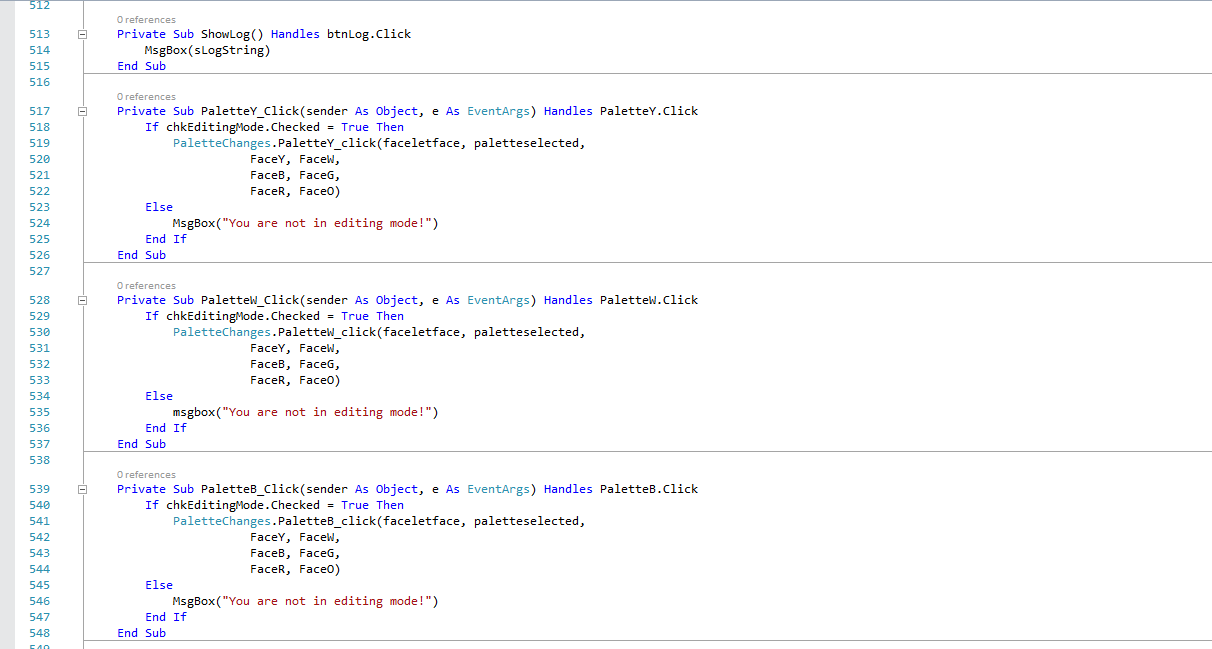


Lines 493 to 508 deals with putting moves onto the log in a suitable manner.

### ShowLog

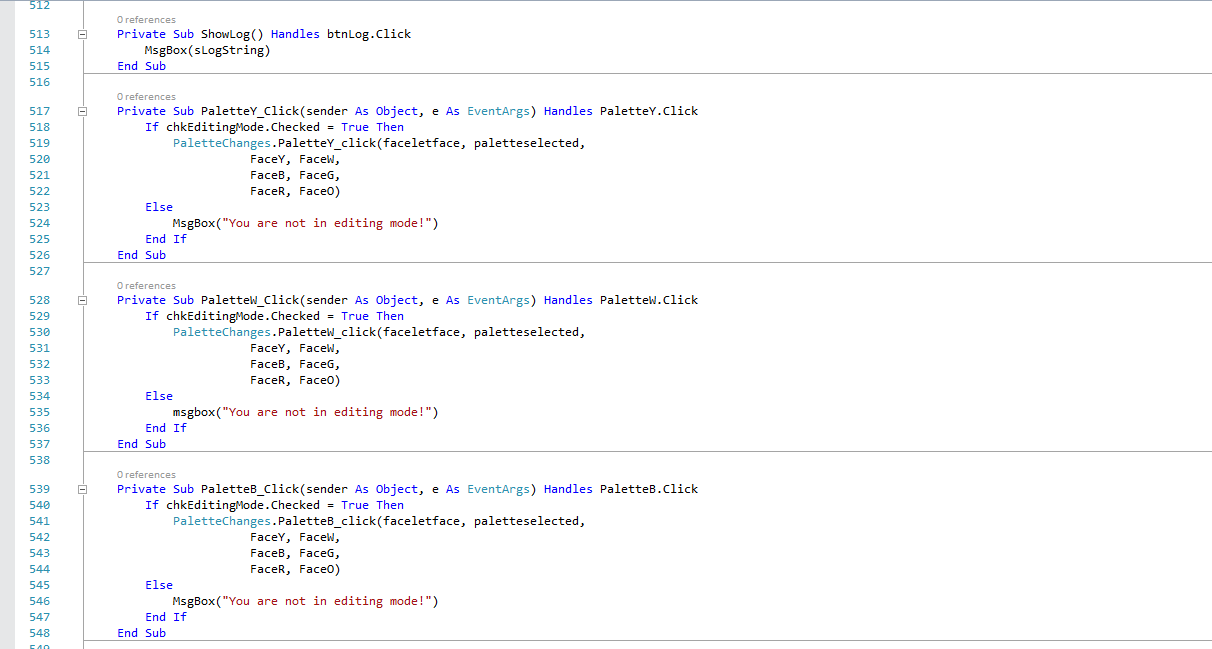


### PaletteY\_Click

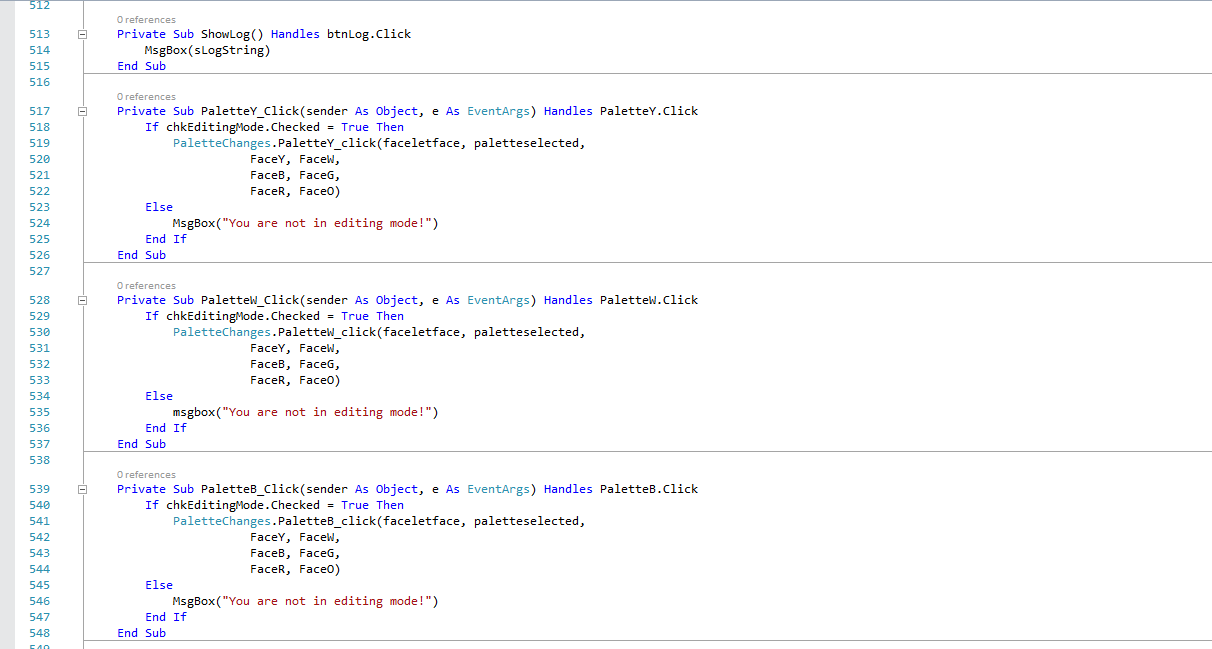


Line 518 uses OOP again (Group A).

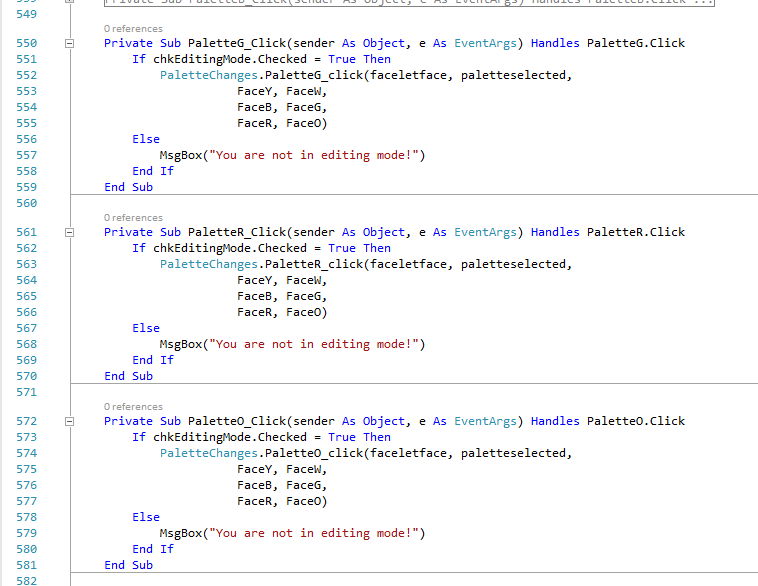
### PaletteW\_Click



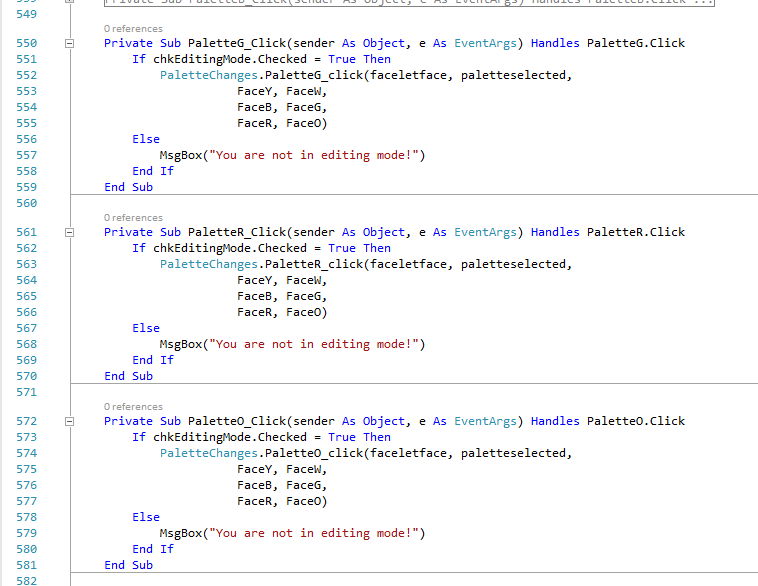
### PaletteB\_Click



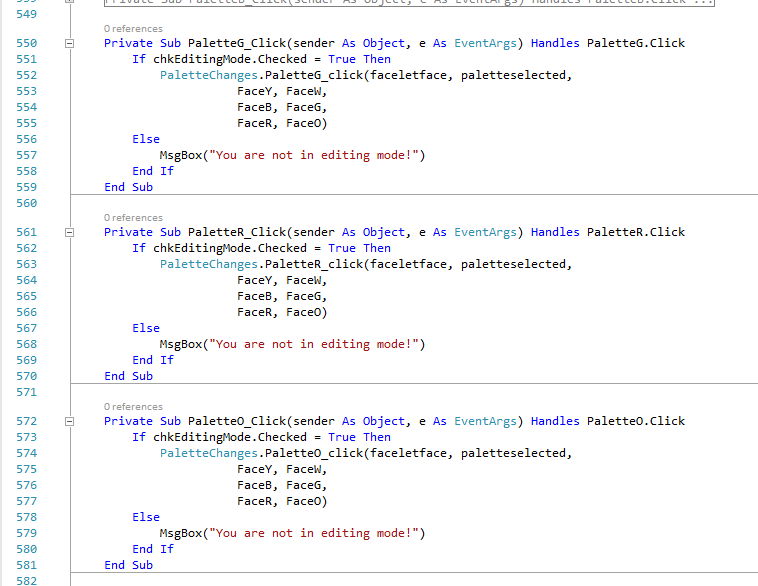
### PaletteG\_Click



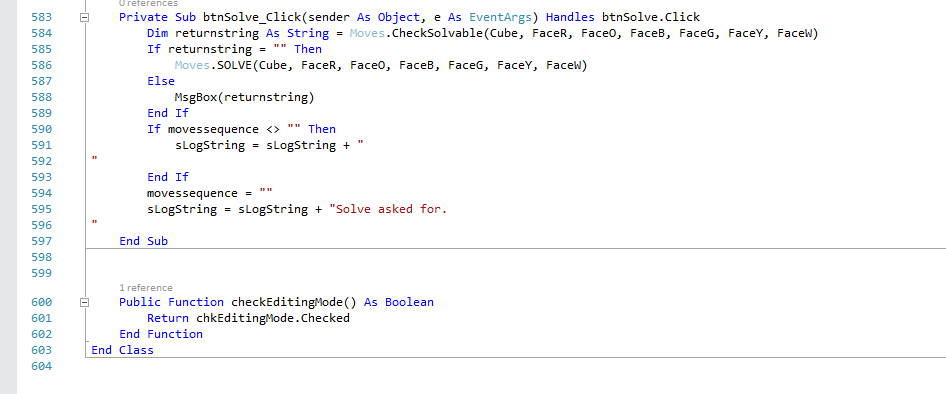
### PaletteR\_Click



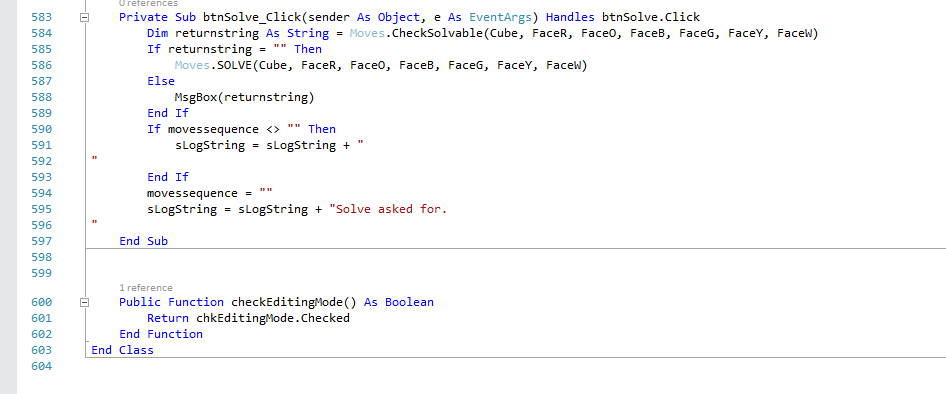
### PaletteO\_Click



### btnSolve\_Click



### checkEditingMode



# Testing

## Test Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No** | **Description** | **Test Data** | **Expected Results** | **Pass / Actual Results** |
| 1 | Check if the face shows up with 9 facelets. | N/A |  | Pass |
| 2 | The 9 facelets on Face1 change colours between white, yellow, blue, green, red and orange when “Start” button is pressed. | Press “Start” button |  | Pass |
| 3 | The nine faces to align correctly in a net. It looks like I miscalculated, and thought the faces were 50 pixels wide instead of 75. | N/A |  |  |
| 4 | See Test 3. | N/A |  | Pass |
| 5 | When I click on the “Reset” button, the faces should colour themselves by their respective colours. | Press “Reset” button |  | Array Boundary Error |
| 6 | See Test 5. I changed a line: | Press “Reset” button |  | Pass |
| 7 | When I press the “R” button, the facelets should mirror how an R move would look like on a 3D cube. | Press “R” button |  |  |
| 8 | See Test 7. | Press “R” button |  | Pass |
| 9 | When pressing the “L” button, it should show the equivalent of turning the left face clockwise. Looks like I called the R subroutine from the form. | Press “L” button |  |  |
| 10 | See Test 9. | Press “L” button |  | Pass |
| 11 | When pressing the “U” button, it should show the equivalent of turning the up face clockwise. I didn’t call the subroutine from the form. | Press “U” button |  |  |
| 12 | See Test 11. To deal with this, I will comment every line and decomment them. I have realised that I took the cubies’ positions relative to how they are seen when looking at the face. | Press “U” button |  |  |
| 13 | See Test 12. I don’t know what happened. | Press “U” button |  |  |
| 14 | See Test 13. I realised I got the rows and columns mixed up so I changed the lines in the moves subroutines and in the Face class. | Press “U” button |  |  |
| 15 | See Test 14. I changed back the lines in the Face class as the rows and columns were being swapped twice. We’re getting closer. I think the problem is in the “UL other faces” section. | Press “U” button |  |  |
| 16 | See Test 15. | Press “U” button |  | Pass |
| 17 | I realised from previous tests that the current L subroutine is modelling the anticlockwise move. Therefore, I will change that subroutine. | Press “L” button |  | Pass |
| 18 | When pressing the “D” button, it should show the equivalent of turning the down face clockwise. I have seen this before, so I will change the “DL other faces” section. | Press “D” button |  |  |
| 19 | See Test 18. | Press “D” button |  | Pass |
| 19 | To see if the D move works with the down face, I did R then D. I compared my result with the one from a website and it matches. | Press “R” button then “D” button |  | Pass |
| 20 | To see if the U move works with the up face, I did L then U. I compared my result with the one from a website and it matches. | Press “L” button then “U” button |  | Pass |
| 21 | To see if the L move works with the left face, I did U then L. I compared my result with the one from a website and it matches. | Press “U” button then “L” button |  | Pass |
| 22 | To see if the R move works with the right face, I did D then R. I compared my result with the one from a website and it matches. | Press “D” button then “R” button |  | Pass |
| 23 | When pressing the “F” button, it should show the equivalent of turning the front face clockwise. | Press “F” button |  | Pass |
| 24 | To see if the F move works with the front face, I will do R then F. I will compare my result with the one from a website and it matches. | Press “R” button then “F” button |  | Pass |
| 25 | When pressing the “B” button, it should show the equivalent of turning the back face clockwise. I called the F subroutine from the form when the B button was clicked. | Press “B” button |  |  |
| 26 | See Test 25. | Press “B” button |  | Pass |
| 27 | To see if the B move works with the back face, I will do R then B. I will compare my result with the one from a website and it matches. | Press “R” button then “B” button |  | Pass |
| 28 | When pressing the “M” button, it should show the equivalent of turning the middle column clockwise. | Press “M” button |  | Pass |
| 29 | When clicking on the up arrow on the up-down Scramble control, it should stop at 30. | Press up button on said control |  | Pass |
| 30 | See if I can assign a string array to the properties of another with the same size |  | Form1 pops up. | Array bounds error |
| 31 | When doing integer division, it looks like the program rounded to the nearest whole number. I added Math.Floor around all integer divisions in my program. |  | Form1 pops up. | Pass. |
| 32 | To test the R’ subroutine, I will execute R then R’ and the cube should be as its default position. | Press R button then R’ button |  | Pass |
| 33 | To test the facelets on the right face, I will do U, R, R’ which should end up showing the move U. | Press U button, then R button, then R’ button |  | Pass |
| 34 | To test the M’ subroutine, I will execute M then M’ and the cube should be as its default position. | Press M button then M’ button |  | Pass |
| 35 | To test the facelets on the up face, I will do R, U, U’ which should end up showing the move R. | Press R button, then U button, then U’ button |  | Pass |
| 36 | To test the facelets on the down face, I will do R, D, D’ which should end up showing the move R. | Press R button, then D button, then D’ button |  |  |
| 37 | To test the facelets on the front face, I will do R, F, F’ which should end up showing the move R. It looks like the problem was in the “UR other faces” section. | Press R button, then F button, then F’ button |  |  |
| 38 | See test 37. | Press R button, then F button, then F’ button |  | Pass |
| 39 | To test the facelets on the back face, I will do R, B, B’ which should end up showing the move R. It looks like it was turning the face clockwise. | Press R button, then B button, then B’ button |  |  |
| 40 | See Test 39. | Press R button, then B button, then B’ button |  | Pass |
| 41 | To see if the M move works, I executed U then M | Press U button, then M button |  | Pass |
| 42 | See Test 41. When pressing M’, it should show the U move | Press M’ button |  | Pass |
| 43 | Test the x rotation subroutine | Press “x” button |  | Pass |
| 44 | To test if the x subroutine works for all facelets, I will do a random series of moves (R, U, L, B, D) and then x to see if the result matches the one from a website. I also counted the number of facelets of each colour to check if they add up. | Press “R”, “U”, “L”, “B” and “D” buttons |  | Pass |
| 45 | I changed the faces’ coordinates and sizes so that it matches the design UI. |  |  |  |
| 46 | See Test 45. |  |  | Pass |
| 47 | When pressing the “Help” button, a message box should appear with a predefined string. | Press “Help” button |  | There are quotation marks around the text, but this is easily resolvable. |
| 48 | After some messing around, I realised that the x or x’ subroutines don’t work. | Mess around |  |  |
| 49 | To see if the y subroutine works, I did L then y. | Press “L” button then “y” button |  |  |
| 50 | See Test 49. |  |  | Pass |
| 51 | To see if the x subroutine works, I will do U then x. | Press “U” button then “x” button |  | Pass |
| 52 | To see if the z subroutine works, I will do M then z. | Press “M” button then “z” button |  |  |
| 53 | See Test 52. | Press “M” button then “z” button |  | Pass |
| 54 | To see if the x’ subroutine works, I will do U then x’. | Press “U” button then “x’” button |  |  |
| 55 | See Test 54. | Press “U” button then “x’” button |  | Pass |
| 56 | To see if the z’ subroutine works, I will do M then z’. | Press “M” button then “z’” button |  | Pass |
| 57 | To see if the y’ subroutine works, I will do L then y’. | Press “L” button then “y’” button |  | Pass |
| 58 | To see if the CheckSolved function works, I pressed a random move. | Press any move button |  |  |
| 59 | See Test 58. The message box pops up even when the cube is not solved. | Press any move button |  |  |
| 60 | See Test 59. | Press any move button |  | Pass |
| 61 | See Test 60. | Press Test 60’s move’s inverse’s button |  | Pass |
| 62 | Test if the log works with the scramble subroutine. It looks like it is adding an integer with a string which it doesn’t like. | Press “Scramble” button |  |  |
| 63 | Test the scramble subroutine | Press “Scramble” button |  | Pass |
| 64 | Test if the log works. I pressed some random moves, then did a scramble, then reset the cube. | Press moves buttons, the scramble button |  | Fail |
| 65 | I found out that the palette textbox is writable, but it would be preferable if not. | Delete character from Palette textbox |  |  |
| 66 | See Test 65. | Try to delete character from Palette textbox |  | Pass |
| 67 | When clicking on a palette and then a facelet, it appears that it only works for the colour yellow. |  |  |  |
| 68 | When clicking on the blue palette and then on an orange facelet, it puts it as grey. | Click on blue palette |  |  |
| 69 | I put in a line so that the program reminds the user that they are not in Editing Mode if the tickbox is not ticked. | Press any palette colour |  | Pass |
| 70 | I tested each of the palette colours with all the faces. |  |  | Pass |
| 71 | Check if the solve button checks if the cube is solvable and calls the solve subroutine, which for the moment only outputs a textbox saying “Solve algorithm” to say that the subroutine has been called. | Press “Solve” button |  |  |
| 72 | See Test 71 | Press “Solve” button |  | Pass |
| 73 | Test to see if the E subroutine works. | Press “E” button |  | Pass |
| 74 | Test to see if the E’ subroutine works. | Press “E’” button |  | Pass |
| 75 | Test to see if the S subroutine works. | Press “S” button |  | Pass |
| 76 | Test to see if the S’ subroutine works. | Press “S’” button |  | Pass |
| 77 | To see if the E subroutine properly works, I did M, F then E and compared against a website | Press “M”, “F”, then “E” buttons |  | Pass |
| 78 | To see if the E subroutine properly works, I copied Test 77 then did E’ and compared against a website. | Press “M”, “F”, “E” then “E’” buttons |  | Pass |
| 79 | To see if the E’ subroutine properly works, I did M, U then S and compared against a website. | Press “M”, “U”, then “S” buttons |  | Pass |
| 80 | To see if the S’ subroutine properly works, I copied Test 79 then did S’ and compared against a website. | Press “M”, “U”, “S” then “S’” buttons |  | Pass |
| 81 | When scrambling and then solving the cube, the log should show the scramble, moves made, and the fact that the cube has been solved. | Start a scramble, solve the cube, press “Log” button |  |  |
| 82 | Check if the program knows that the cube is solved when the form first pops up. | Press “Solve” button when the form loads |  | Pass |
| 83 | See if the log works with the moves. | Press any move button |  |  |
| 84 | See Test 83. | Press some moves buttons |  | Pass |
| 85 | To see if the program knows if the cube has been solved. | Press “U” then “U’” buttons |  |  |
| 86 | To see if the program outputs the last move before the cube is solved. | Press “U” then “U’” buttons |  |  |
| 87 | See Test 87. | Press “U” then “U’” buttons |  | Pass |
| 88 | See Test 81. | Start a scramble, solve the cube, press “Log” button |  | Pass |
| 89 | Test to see if the program checks if the user is in editing mode when clicking on a facelet. | Click on a facelet. |  |  |
| 90 | See Test 89. | Click on a facelet. |  | Pass |

## Objectives Testing

This table is to show which objectives have been met and have passed their tests.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objective No** | **Test Description** | **Test Data** | **Expected Results** | **Pass / Actual Results** |
| 1 | This objective does not need testing as it was worked on and was not met. | N/a | N/a | n/a |
| 2 | Test every move in a specific order, then compare against a website. | L, R, U, D, F, B, L’, R’, U’, D’, F’, B’, M, E, S, x, y, z, M’, E’, S’, x’, y’, z’ |  | Pass |
| 3 and 5.7 | When the form loads, the buttons should be neatly arranged and centered, as per the design UI. I have decided to make all of the buttons the same size. | n/a |  | Pass |
| 4 | Test each of the six rotations. | n/a | n/a | n/a |
| 4 (x) | Check if the x subroutine works. | Press “x” button |  | Pass |
| 4 (y) | Check if the y subroutine works. | Press “y” button |  | Pass |
| 4 (z) | Check if the z subroutine works. | Press “z” button |  | Pass |
| 4 (x’) | Check if the xa subroutine works. | Press “xa” button |  | Pass |
| 4 (y’) | Check if the ya subroutine works. | Press “ya” button |  | Pass |
| 4 (z’) | Check if the za subroutine works. | Press “za” button |  | Pass |
| 5 | See if the form loads and shows the UI as it is meant to be in the design UI. | n/a |  | Pass |
| 5.1 | This objective works if objective 5.2 works. | n/a | n/a | Pass |
| 5.2 | When the form loads, the six faces should arrange themselves as it would look on a normal cube net. | n/a |  | Pass |
| 5.2.1 | When the form loads and the six faces show, yellow should be on top and white should be at the bottom. | N/a |  | Pass |
| 5.2.2 and 5.2.3 | I did not have enough time to meet these objectives so they will not be tested. |  |  |  |
| 5.3 | Press any move and press Reset button. This objective works in conjunction with objective 6. | Press any move button, then “Reset” button |  | Pass |
| 5.4 and 5.5 | I did not animate so these have failed. | n/a |  | Fail |
| 5.6 | See if facelets can change colour when clicked on and tickbox is ticked. | n/a |  | Pass |
| 5.6.1 | See if the facelets don’t change colour when clicked and tickbox is not ticked. | n/a |  | Pass |
| 6 | This objective works in conjunction with objective 5.3. | n/a | The Design UI did not include this button. | Pass |
| 7 | Scramble. | Press “Scramble” button with 3 moves. |  | Pass |
| 8 | The “History” button got renamed “Log” because I thought it sounded more convenient. | n/a | The Design UI did not include this button. | Pass |
| 8.1 | A new window appears when the “Log” button is pressed. | Do a scramble, solve it manually, do more moves, reset. Press “Log” button. |  | Pass |
| 8.2 and 8.3 | I did not feel that the user would like to access previous sessions. |  |  |  |
| 9 | “Help” button |  | The “Help” button is in a different place on the form than on the Design UI. | Pass |
| 9.1 | A new window appears when “Help” button is pressed. | Press “Help” button. |  | Pass |
| 9.2 |  |  |  | Pass |
| 10 | I did not have enough time to make this, in my opinion. | Press “SOLVE” button. |  | Fail |

# Evaluation – TO BE FINISHED BY 19/12/23

1. - - problems/changes
2. - - how well ?

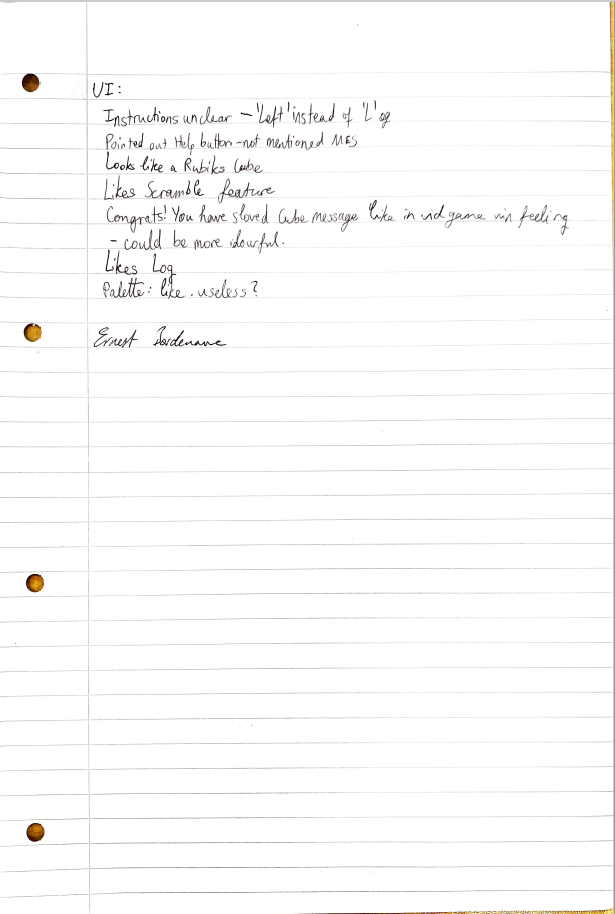
## Evaluating objectives

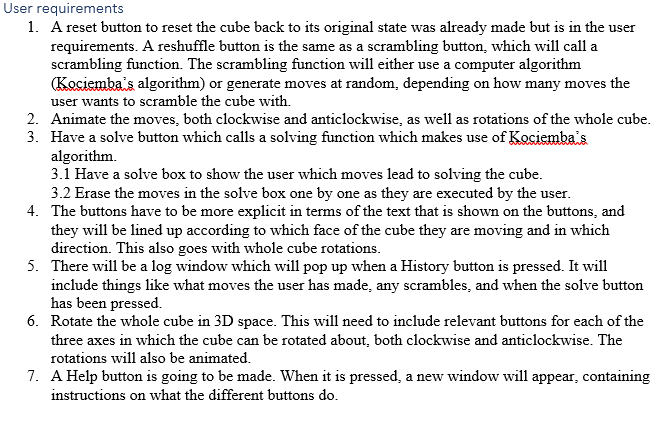
In this table, I list the objectives and evaluate each one on how well I met it and if there were any issues.

|  |  |
| --- | --- |
| **Objective** | **Evaluation** |
| 1. The underlying cube | This is a big flop, as I was mainly working on outputting the facelets onscreen, as they are more important, rather than updating the underlying Cube. |
| 1.1 3x3x3 array of cubies | It has been created but wasn’t used much, if at all. |
| 1.1.1 Make a Cubie class with a CubieType property. Make sure each Cubie has the right number of colours for its CubieType. Preferably instantiate the cubies at run-time. | Met. I have also instantiated the cubies at run-time so this is a success. |
| 1.1.2 Store the cubies’ permutations. | Met. |
| 1.1.3 Make a private Cube made of 27 Cubies and preferably instantiate at run time. | Met. However, it was not instantiated at run time, as it needed to be passed in as a parameter to subroutines in the Moves module, which seems to not be possible with instances/objects not created at design-time. |
| 1.2 6x3x3 array of faces | Failed as not created. I think I didn’t have enough time. |
| 2. Create the clockwise and anticlockwise moves as functions and return the new cube state, which will prevent the Cube from being public. | I made all the moves as subroutines, however I did not return the cube’s state as I was mainly working on outputting the facelets onscreen than updating the underlying Cube. |
| 3. The buttons on the form have to be neatly arranged and explicit, and there will be a button for each clockwise and anticlockwise move function. | There is indeed a button for each clockwise and anticlockwise move, so this objective has been met. In my opinion, the buttons are neatly arranged, however I will need to get feedback from my end-user. |
| 4. Make six rotation functions, a clockwise and an anticlockwise rotation for each of the three axes. | Met. |
| 4.1 The underlying cube will be updated first before any animation happens. | Fail as I did not focus on the underlying cube, and I ended up doing no animation whatsoever. |
| 5.UI | The general UI follows the design UI, so this objective has been met. |
| 5.1 Make a Face class, which will be composed of 9 Facelets in a 3x3 2D array. | Met. |
| 5.2 Display six Faces in a classic net on a Windows Form. | Met. |
| 5.2.1 Make sure that yellow is above and white is on the bottom, so as to not confuse anyone. | Met. |
| 5.2.2 (Optional) Create the faces at run-time. | Not met. The faces needed to be passed in as parameters from subroutines in the Moves module, which seems to not be possible with instances/objects not created at design-time. |
| 5.2.3 (Optional) Recreate Option 2 from ruwix.com. | Not met, as I didn’t have enough time. |
| 5.3 Make a default position where each face is only one colour and show it when the Reset button is clicked. | Met. This objective works in conjunction with objective 6. |
| 5.4 Animate each move, both clockwise and anticlockwise. For 180° rotations, I will call and animate the clockwise move twice. Use the sleep subroutine when animating moves. | Not met as I didn’t have enough time. Also, there were no buttons for 180° moves or rotations. |
| 5.5 Animate the rotations. The facelets on the far side will be updated first, and the wave will continue in the direction that the cube is being rotated in. Any faces on the axis of rotation will be animated in a clockwise or anticlockwise manner, and will be animated along with the rest of the cube as it rotates. | Not met as I didn’t have enough time. |
| 5.6 Make a colour palette so that the user can input individual colours. | Met. This took quite some time to get it to work, but I managed it in the end. |
| 5.6.1 Have a tickbox so the user can say that they are in Editing Mode. | Met. |
| 5.7 Have the move buttons in neat order. Align the clockwise and anticlockwise move buttons with their respective partner. | I have realised that this is repeat of objective 3, so in my opinion it has been met. |
| 6. Make a Reset button which will call an algorithm that resets the state of the cube and facelets. | Met. This objective works in conjunction with objective 5.3. |
| 7. Make a scramble function. It will work in conjunction with a textbox(?) in which the user can input how many moves they want the scrambling function to scramble with. | Met. The textbox ended up being an up-down control, which I didn’t know the name of when writing the objectives. This is some defensive programming. |
| 8. A History button will be made. | Met, however it ended up being called the “Log” button, as I felt that that was a more sensible name. |
| 8.1 When it is pressed a new window will appear, which will contain a log of everything that has happened with the cube so far in the session. | Met. |
| 8.2 A text file is to be created to keep track of what has happened. | Not met, as I didn’t have the time, and I thought it as too unnecessary compared to other things, such as the palette. |
| 8.3 An integer variable will contain the line number of the file which starts the current session. | Not met as objective 8.2 was not met so the textfile was not made, which meant that no integer variable held the line number. |
| 9. A Help button will be created. | Met. |
| 9.1 When it is pressed, a new window will appear containing instructions on how each button works and describe their relevant moves. | Met. |
| 9.2 The text string in the Help window will be stored in a constant variable. | Met. |
| 10. The algorithm to solve the cube (Kociemba’s algorithm) | Not met as I decided that it would take too long. |

## Feedback from user

When the program was finished, I sat down with my end-user and I showed him my program.





- Link to user requirements

- Ask for improvements

- summarise

-signed

* Off first impressions, he liked it.
* User requirement 4: He said that the instructions are unclear. I didn’t understand what he was saying as I liked my layout, and he said, for example, for the Left button, it should say “Left” instead of “L”.
* He didn’t know what E and S were, so I pointed out the Help button (user requirement 7), and we realised that it didn’t cover E and S, whereas it covered all the others.
* He said that it looks like a Rubik’s Cube. He understood the need for the Reset button (user requirement 1) but didn’t express an opinion on it.
* He liked the scramble feature.
* After experimenting with the scramble feature, he said he liked the “Congratulations! You have solved the cube” pop-up message. It was like the winning message in a video game and made him feel powerful. He said that it could be made more colourful, because it looks like an error message.
* I told him to click on the Log button (user requirement 5), and he said that he liked it. He asked if he could see what I wrote on the first interview to remind himself of what we discussed.

I told him to experiment with the palette and I told him about my box-around-the-palette idea, and he said that he likes the palette but that it was a bit useless.

* I asked him to sign it, to prove that he is the same person as my end-user at the beginning.

## Improvements

1. – 3+ suggestions
2. - – rough idea how?
3. - - how it would add to the solution
4. 4. ^ should link to user feedback

This is a list of improvements that could make the program better:

INCLUDE ONE LINKING TO USER FEEDBACK

### Actually use the underlying cube (objectives 1 and 4.1)

This was, in my opinion, the main challenge in this project. However, I didn’t get round to it, and it didn’t help that much. It would, however, be very beneficial in the solving algorithm, which wasn’t made in the end. In order to implement this, I would edit the moves subroutines so that they also deal with the underlying cube as well as updating the facelets on screen.

I also didn’t try out the 6x3x3 array of faces, however upon further reflection I have realised that it would be pretty useless anyway and it was only included for experimenting.

Upon further reflection, I don’t think the underlying cube was very important, as its main purpose was to satisfy the solving algorithm. I have realised that I could have just checked the facelets to see if the cube was in the G1 state. Checking the underlying cube to check if the algorithm was in G1 would have required some research into the most effective way to search through a 3D array.

### Arrangement of buttons (objectives 3 and 5.7)

Doing this in a separate subroutine was smart, in my opinion, as I would only have had to edit that subroutine to change the buttons’ positions, given that the subroutine was in Form1, so the buttons didn’t need to be public.

However, my end-user was not very pleased with the arrangement of the buttons. I think I should have asked him to elaborate more.

### Box around palette reflects the selected colour

This box around the palette was in rubiks-cube-solver.com, and I wanted to replicate that if I had the time. However, I found it pretty difficult to place the box behind the palette colours, so I decided to not do it at all. It would have been nice to have it, so that the user doesn’t have to remember which colour they last selected, however the selected palette colour is forgotten when a move happens, so it doesn’t matter a lot. I spent quite a lot of time on this, which I could have used for doing other things, but it was nice to experiment.

Any palette edits are not put on the Log, and upon reflection I have realised that it would be nice to have a message like for example: “4 facelets have been changed to green.”

### Animation (objectives 5.4 and 5.5)

I didn’t have enough time to do the animation. When I first made the moves subroutines, I decided to make the animation later, as I was mainly focusing on actually making the facelets move onscreen when a move button was pressed. Animation wouldn’t have really added anything to the properties of the program, but it was in the user requirements when I first interviewed my user.

### The algorithm to solve the cube (Kociemba’s algorithm) (objective 10)

I did not have enough time to do this. The first step in the algorithm, G1, wouldn’t have required me to update the underlying cube, as I could have just checked the facelets. This is also in the user requirements from when I first interviewed my end-user. I would have to refresh my mind on how the algorithm works.

### Checking if the cube is solved

Upon further refection, I have realised that my cube is only solved when the cube is in the Reset position. This means that if the user solves the cube but another face other than blue is facing front or blue is facing front but yellow is not on top, then the program will not recognise it as solved and will not congratulate the user. This also means that if the user wants to see that message, they will have to do some rotations which feels unnecessary.

To deal with this, I could check each face if all the facelets on it are of the same colour. I would first see what colour the centre facelet is and check if the other 8 facelets on the face are the same colour as the centre facelet.

# Links

Information/Description:

* [Writing Code to Solve a Rubik’s Cube | by Brad Hodkinson | Medium](https://medium.com/@brad.hodkinson2/writing-code-to-solve-a-rubiks-cube-7bf9c08de01f)

Analysis:

* <https://www.youtube.com/watch?v=ZtlMkzix7Bw&feature=youtu.be>

Design:

* [data structures - How would you represent a Rubik's Cube in code? - Stack Overflow](https://stackoverflow.com/questions/500221/how-would-you-represent-a-rubiks-cube-in-code)
* [language agnostic - How to represent a Rubik's cube - Stack Overflow](https://stackoverflow.com/questions/9762162/how-to-represent-a-rubiks-cube?rq=1)
* [Representing rubik's cube in APL (acm.org)](https://dl.acm.org/doi/pdf/10.1145/384283.801107)

[Computer Puzzling (jaapsch.net)](https://www.jaapsch.net/puzzles/compcube.htm)