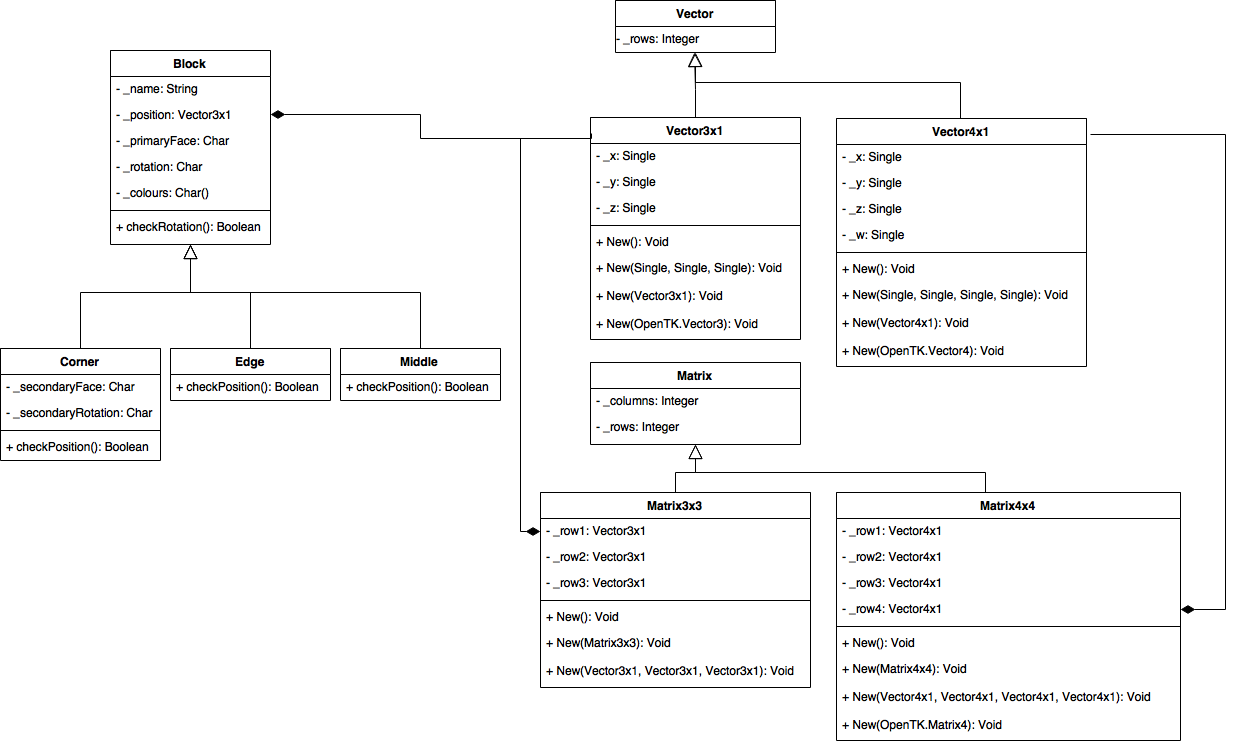
Rubik’s Cube Solver Code – 05/11/17

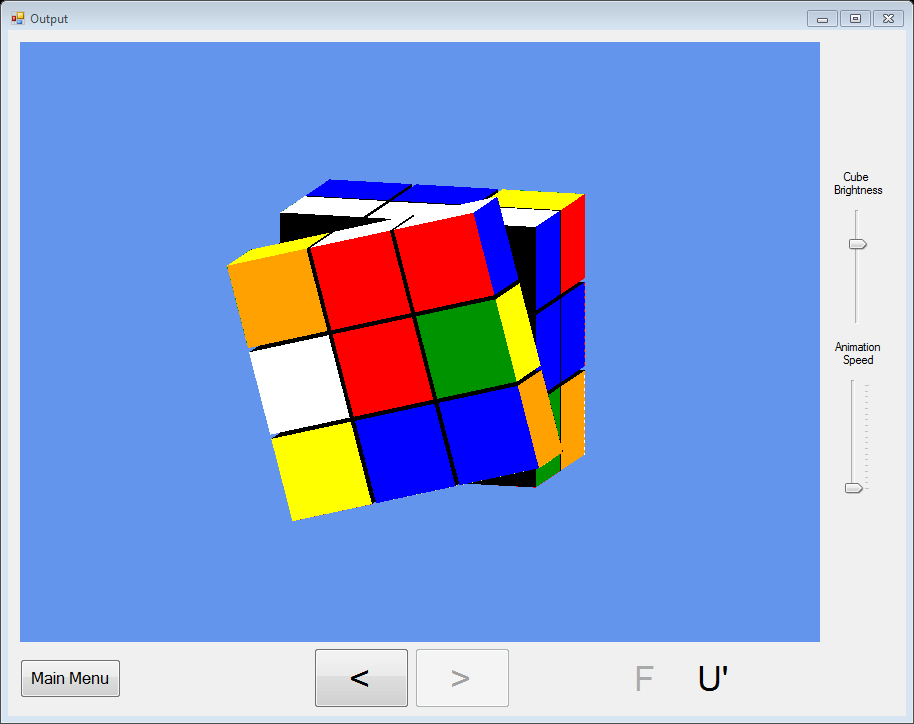


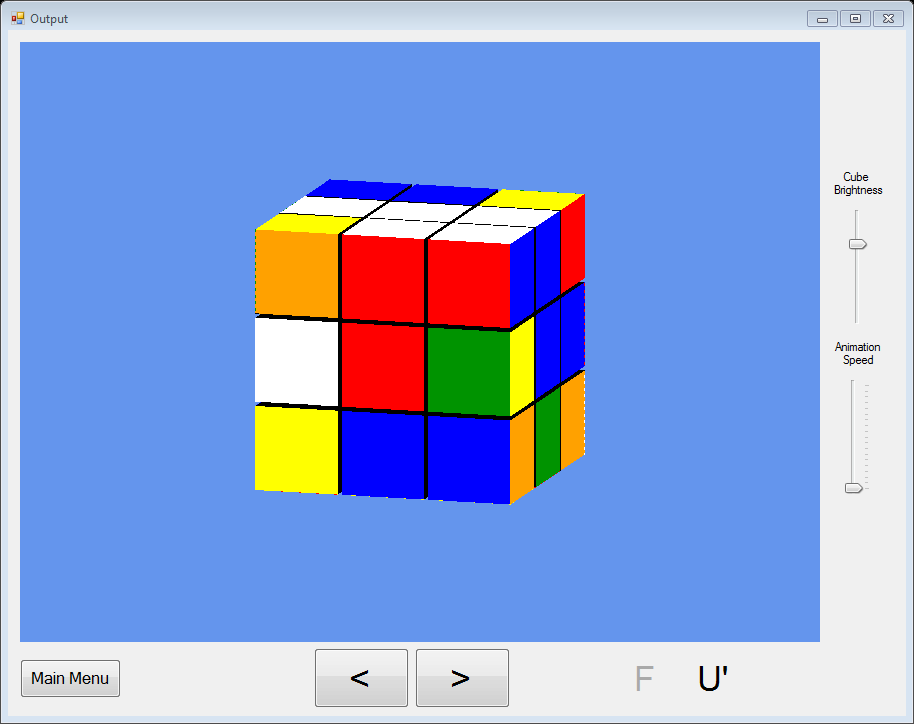
# Screenshots

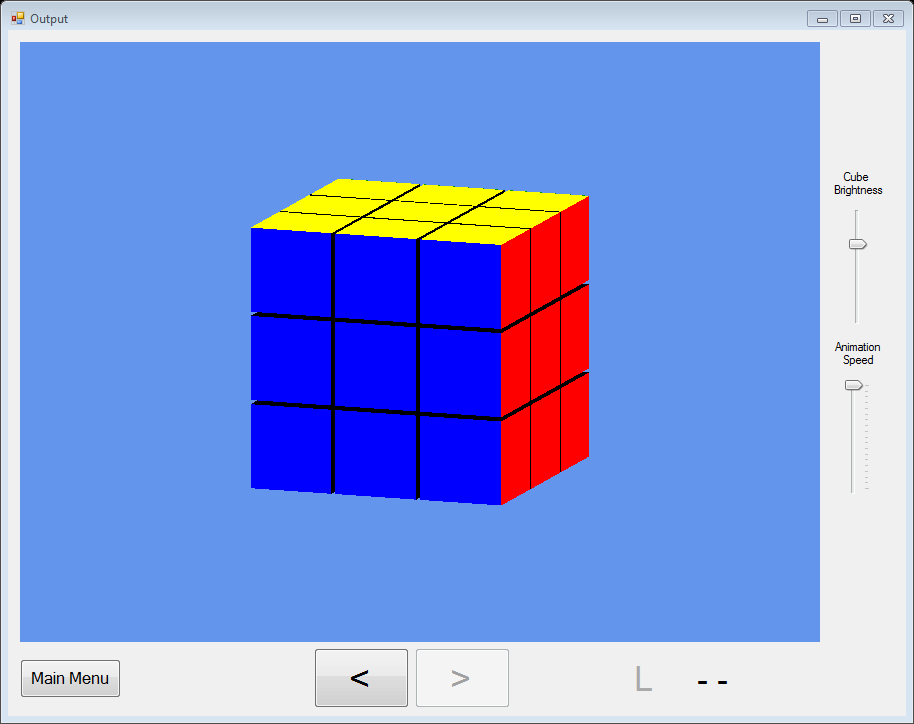
## Input Screens

## 

## Output Screen







# Input

Imports System.IO

Public Class Input

Dim clicked(0 To 5, 0 To 8) As Boolean

Dim stickers(0 To 5, 0 To 8) As Char

Private Sub Input\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

ClearCheckedArray(clicked)

ClearStickersArray(stickers)

End Sub

''' <summary> Handles click events from any of the colour selection boxes </summary>

Private Sub Colour\_Change\_Click(sender As Object, e As EventArgs) Handles a1.Click, b1.Click, c1.Click, d1.Click, f1.Click, g1.Click, h1.Click, i1.Click, a2.Click, b2.Click, c2.Click, d2.Click, f2.Click, g2.Click, h2.Click, i2.Click, a3.Click, b3.Click, c3.Click, d3.Click, f3.Click, g3.Click, h3.Click, i3.Click, a4.Click, b4.Click, c4.Click, d4.Click, f4.Click, g4.Click, h4.Click, i4.Click, a5.Click, b5.Click, c5.Click, d5.Click, f5.Click, g5.Click, h5.Click, i5.Click, a6.Click, b6.Click, c6.Click, d6.Click, f6.Click, g6.Click, h6.Click, i6.Click

Dim boxName As String

Dim faceNumber As Integer

Dim stickerLetter As String

Dim stickerNumber As Integer

Dim clickedBefore As Boolean

boxName = sender.Name

stickerLetter = Mid(boxName, 1, 1) 'e.g. a5 -> a

stickerNumber = Asc(stickerLetter) - 97

faceNumber = Mid(boxName, 2, 1) - 1 'e.g. a5 -> 5

clickedBefore = clicked(faceNumber, stickerNumber)

clicked(faceNumber, stickerNumber) = True

SetButtonColour(sender, faceNumber, stickerNumber, clickedBefore, clicked)

End Sub

Private Sub ResetBtn\_Click(sender As Object, e As EventArgs) Handles ResetBtn.Click

ClearBoxColours(Controls)

ClearCheckedArray(clicked)

ClearStickersArray(stickers)

End Sub

Private Sub AnalyseBtn\_Click(sender As Object, e As EventArgs) Handles AnalyseBtn.Click

If Not AllBoxesClicked(clicked) Then

MsgBox("Please Fill In All The Boxes.")

Return

End If

ResetBtn.Enabled = False

Dim stickersArray As Char(,)

stickersArray = StickerAssign(stickers, Controls)

Dim cubeBlocks(2)() As Block

cubeBlocks = Analyse.AnalyseCube(stickersArray)

If cubeBlocks Is Nothing Then

MsgBox("Cube Not Possible")

Return

End If

Dim corners(7) As Corner

Dim edges(11) As Edge

corners = cubeBlocks(0)

edges = cubeBlocks(1)

Dim processingWindow As New Processing(corners, edges)

processingWindow.Show()

Me.Close()

End Sub

Private Sub btnTest\_Click(sender As Object, e As EventArgs) Handles btnTest.Click

Dim fileName As String = InputBox("Input name of file (without path/extention)")

If fileName = Nothing Then

Exit Sub

End If

ReadTestCube("testCubes\" & fileName & ".txt", stickers)

ResetBtn.Enabled = False

Dim corners(7) As Corner

Dim edges(11) As Edge

Dim cubeBlocks(2)() As Block

cubeBlocks = Analyse.AnalyseCube(stickers)

If cubeBlocks Is Nothing Then 'If the inputted cube is not possible:

MsgBox("Cube Not Possible")

Exit Sub

Else

corners = cubeBlocks(0)

edges = cubeBlocks(1)

Console.WriteLine("Cube Possible")

Console.WriteLine(vbNewLine)

Dim processingWindow As New Processing(corners, edges)

processingWindow.Show() 'Show the processing window

Close() 'Close the cube input window

End If

End Sub

Private Sub ReadTestCube(ByVal filePath As String, ByRef stickersArray(,) As Char)

Dim fReader As StreamReader

Try

fReader = New StreamReader(filePath)

Catch ex As FileNotFoundException

Dim fileName As String = InputBox("Input name of file (without path/extention)")

If fileName = Nothing Then

Exit Sub

End If

ReadTestCube("testCubes\" & fileName & ".txt", stickersArray)

Return

End Try

Dim faceString As String

For face = 0 To 5

faceString = fReader.ReadLine

For sticker = 0 To 8

stickersArray(face, sticker) = faceString(sticker)

Next

Next

fReader.Close()

End Sub

End Class

# Input Helpers

Imports PublicValues = RubiksCubeSolver\_v2\_0.MyPublic.PublicVars

Module InputHelpers

Public Function AllBoxesClicked(ByVal clickedArray(,) As Boolean) As Boolean

'should I use 'for each' or 'nested for(i,j)' loops?

For Each boxClicked In clickedArray

If Not boxClicked Then Return False

Next

Return True

End Function

'''<summary> Assigns values based on the colour of the buttons </summary>

Public Function StickerAssign(ByRef stickersArray(,) As Char, ByVal controls As Control.ControlCollection) As Char(,)

For faceNumber = 0 To 5

For Each stickerLetter In PublicValues.FACE\_NO\_MIDDLE.ToCharArray()

Dim stickerNumber As Integer

Dim colourChar As Char

colourChar = GetButtonColour(stickerLetter, faceNumber, controls)

stickerNumber = (InStr(PublicValues.FACE\_WITH\_MIDDLE, stickerLetter) - 1) 'returns (position of i in "abcdefghi" - 1) == y-coordinate of sticker

stickersArray(faceNumber, stickerNumber) = colourChar

Next

Next

Return stickersArray

End Function

Private Function GetButtonColour(ByVal stickerLetter As Char, ByVal faceNumber As Integer, ByVal controls As Control.ControlCollection) As Char

Dim buttonName As String

Dim box As Button

Dim colourChar As Char

buttonName = stickerLetter & (faceNumber + 1) 'e.g. a2

box = controls.Find(buttonName, True)(0)

colourChar = Mid(box.BackColor.ToString(), 8, 1) 'if box.BackColor.ToString = "Colour [White]", Mid(8, 1) = "W"

If colourChar = "D" Then

colourChar = "O" 'converts DarkOrange to "O"(for Orange)

ElseIf colourChar = "L" Then

colourChar = "G" 'converts Lime to "G"(for Green)

End If

Return colourChar

End Function

Public Sub SetButtonColour(ByRef box As Button, ByVal face As Integer, ByVal sticker As Integer, ByVal clickedbefore As Boolean, ByRef clickedArray(,) As Boolean)

Dim colour As Color

colour = GetUserColour()

If colour = Nothing Then

If Not clickedbefore Then

clickedArray(face, sticker) = False

End If

Return

End If

box.BackColor = colour

End Sub

Private Function GetUserColour() As Color

Dim colourPickerObject As New ColourPicker

colourPickerObject.ShowDialog()

If colourPickerObject.colour = Color.Black Then 'Black = default colour ==> no change

colourPickerObject.Close()

Return Nothing

End If

colourPickerObject.Close()

Return colourPickerObject.colour

End Function

Public Sub ClearCheckedArray(ByRef clickedArray(,) As Boolean)

For i = 0 To 5

For j = 0 To 8

If j = 4 Then

clickedArray(i, j) = True

Else

clickedArray(i, j) = False

End If

Next

Next

End Sub

Public Sub ClearBoxColours(ByVal controls As Control.ControlCollection)

For Each faceChar As Char In PublicValues.FACE\_NO\_MIDDLE.ToCharArray()

For buttonNumber = 1 To 6

Dim buttonName As String

buttonName = faceChar.ToString() & buttonNumber.ToString() 'e.g. b5

Dim box As Button = controls.Find(buttonName, True)(0)

box.BackColor = Color.LightGray

Next

Next

End Sub

Public Sub ClearStickersArray(ByRef stickersArray(,) As Char)

stickersArray(0, 4) = "W"

stickersArray(1, 4) = "G"

stickersArray(2, 4) = "R"

stickersArray(3, 4) = "B"

stickersArray(4, 4) = "Y"

stickersArray(5, 4) = "O"

For i = 0 To 5

For j = 0 To 8

If j <> 4 Then

stickersArray(i, j) = vbNullChar

End If

Next

Next

End Sub

End Module

# ColourPicker

Public Class ColourPicker

Public colour As Color

'On Confirm button click

Private Sub ConfirmColour\_Click(sender As Object, e As EventArgs) Handles ConfirmColour.Click

Select Case True

Case Red.Checked

colour = Color.Red

Case Blue.Checked

colour = Color.Blue

Case Yellow.Checked

colour = Color.Yellow

Case Orange.Checked

colour = Color.DarkOrange

Case Green.Checked

colour = Color.Lime

Case White.Checked

colour = Color.White

Case Else

MsgBox("Something Broke")

colour = Color.LightGray

End Select

Me.Close()

End Sub

'On form load

Private Sub ColourPicker\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

'Deafult colour - causes no change on main form

colour = Color.Black

ConfirmColour.Focus()

End Sub

'On Cancel button Click

Private Sub CancelColour\_Click(sender As Object, e As EventArgs) Handles CancelColour.Click

Me.Close()

End Sub

End Class

# Analyse

Imports PublicFuncs = RubiksCubeSolver\_v2\_0.MyPublic.PublicFunctions

Public Class Analyse

Public Shared Function AnalyseCube(ByVal Cube(,) As Char) As Block()()

If Not StickerCheck(Cube) Then

Return Nothing

End If

Dim cubeBlocks(2)() As Block

cubeBlocks = PublicFuncs.CreateBlocks(Cube)

Return cubeBlocks

End Function

'Returns True if all 'stickers' CAN be in the inputted positions (i.e. if the cube stored in Stickers() is a possible cube)

Shared Function StickerCheck(ByVal Stickers(,) As Char)

If Not checkNumber(Stickers) Then

Return False

ElseIf Not checkEdges(Stickers) Then

Return False

ElseIf Not checkCorners(Stickers) Then

Return False

Else

Return checkCornerRotation(Stickers)

End If

End Function

''' <summary> Checks there are 9 of each colour in the array </summary>

''' <returns> true if there are 9 of each </returns>

Shared Function checkNumber(ByVal Stickers(,) As Char) As Boolean

If PublicFuncs.OtherIn1DIntArray(PublicFuncs.NumberInCharArray(Stickers), 9) Then

MsgBox("You have not entered the correct number of each colour. Are you sure you entered all the squares correctly?")

Return False

End If

Return True

End Function

'Checks adjacent middle edge cube != opposite colour or same colour - adjacent across 2 faces, not the same face

Shared Function checkEdges(ByVal Stickers(,) As Char) As Boolean

Dim oppColour As Char

For faceNumber = 0 To 5

For edgeNumber = 1 To 7 Step 2

Dim adjacentFaceNumber, adjacentEdgeNumber As Integer

Dim adjacentCubeCoordinate(1) As Integer

adjacentCubeCoordinate = PublicFuncs.AdjacentEdge(faceNumber, edgeNumber)

adjacentFaceNumber = adjacentCubeCoordinate(0)

adjacentEdgeNumber = adjacentCubeCoordinate(1)

oppColour = PublicFuncs.Opposite(Stickers(faceNumber, edgeNumber)).ToString()

'if the adjacent sticker is the same or opposite colour return false

If Stickers(adjacentFaceNumber, adjacentEdgeNumber) = Stickers(faceNumber, edgeNumber) Or

Stickers(adjacentFaceNumber, adjacentEdgeNumber) = oppColour Then

Dim wrongFace1, wrongFace2 As String

wrongFace1 = PublicFuncs.ColourChar2Word(Stickers(faceNumber, 4)).ToLower()

wrongFace2 = PublicFuncs.ColourChar2Word(Stickers(adjacentFaceNumber, 4)).ToLower()

MsgBox("The middle cube on the " & wrongFace1 & "/" & wrongFace2 & " edge can't be that. Are you sure you entered it correctly?")

Return False

End If

Next edgeNumber

Next faceNumber

Return True

End Function

'Check adjacent 2 corner stickers != opposite color, checking all corner stickers checks each corner cube - adjacent across 2 faces, not the same face

Shared Function checkCorners(ByVal Stickers(,) As Char) As Boolean

Dim oppColour As Char

For faceNumber = 0 To 5

For cornerNumber = 0 To 8 Step 2

If cornerNumber <> 4 Then

Dim coordinateArray(3) As Integer

coordinateArray = PublicFuncs.AdjacentConers(faceNumber, cornerNumber)

Dim x1, y1, x2, y2 As Integer

x1 = coordinateArray(0)

y1 = coordinateArray(1)

x2 = coordinateArray(2)

y2 = coordinateArray(3)

oppColour = PublicFuncs.Opposite(Stickers(faceNumber, cornerNumber)).ToString()

'if either adjacent corner sticker is the same or opposite colour return false

If Stickers(x1, y1) = Stickers(faceNumber, cornerNumber) Or

Stickers(x2, y2) = Stickers(faceNumber, cornerNumber) Or

Stickers(x1, y1) = oppColour Or Stickers(x2, y2) = oppColour Then

Dim wrongFace1, wrongFace2, wrongFace3 As String

wrongFace1 = PublicFuncs.ColourChar2Word(Stickers(faceNumber, 4)).ToLower()

wrongFace2 = PublicFuncs.ColourChar2Word(Stickers(x1, 4)).ToLower()

wrongFace3 = PublicFuncs.ColourChar2Word(Stickers(x2, 4)).ToLower()

MsgBox("The cube on the " & wrongFace1 & "/" & wrongFace2 & "/" & wrongFace3 & " corner can't be that. Are you sure you entered it correctly?")

Return False

End If

End If

Next cornerNumber

Next faceNumber

Return True

End Function

'Checks if corners are in right rotation (i.e. stickers on cube) in right order - (i.e. if 3rd sticker clockwise = what it should be based on the 1st and 2nd)

Shared Function checkCornerRotation(ByVal Stickers(,) As Char) As Boolean

For faceNumber = 0 To 5

For cornerNumber = 0 To 8 Step 2

If cornerNumber = 4 Then

Continue For

End If

Dim coordinateArray(3) As Integer

coordinateArray = PublicFuncs.AdjacentConers(faceNumber, cornerNumber)

Dim x1, y1, x2, y2 As Integer

x1 = coordinateArray(0)

y1 = coordinateArray(1)

x2 = coordinateArray(2)

y2 = coordinateArray(3)

Dim firstStickerColour As Char = Stickers(faceNumber, cornerNumber)

Dim secondStickerColour As Char = Stickers(x1, y1)

Dim thirdStickerColour As Char = Stickers(x2, y2)

If firstStickerColour <> "W" And firstStickerColour <> "Y" Then

Continue For

End If

Dim correctThirdStickerColour As Char

If firstStickerColour = "W" Then

Select Case secondStickerColour

Case "R" : correctThirdStickerColour = "G"

Case "O" : correctThirdStickerColour = "B"

Case "G" : correctThirdStickerColour = "O"

Case "B" : correctThirdStickerColour = "R"

End Select

ElseIf firstStickerColour = "Y" Then

Select Case secondStickerColour

Case "R" : correctThirdStickerColour = "B"

Case "O" : correctThirdStickerColour = "G"

Case "G" : correctThirdStickerColour = "R"

Case "B" : correctThirdStickerColour = "O"

End Select

End If

If thirdStickerColour = correctThirdStickerColour Then

Continue For

End If

Dim wrongFace1, wrongFace2, wrongFace3 As String

wrongFace1 = PublicFuncs.ColourChar2Word(Stickers(faceNumber, 4)).ToLower()

wrongFace2 = PublicFuncs.ColourChar2Word(Stickers(x1, 4)).ToLower()

wrongFace3 = PublicFuncs.ColourChar2Word(Stickers(x2, 4)).ToLower()

MsgBox("The cube on the " & wrongFace1 & "/" & wrongFace2 & "/" & wrongFace3 & " corner can't be that. Are you sure you entered it correctly?")

Return False

Next cornerNumber

Next faceNumber

Return True

End Function

End Class

# MyPublic

Namespace MyPublic

Module PublicVars

Public Const FACE\_NO\_MIDDLE As String = "abcdfghi"

Public Const FACE\_WITH\_MIDDLE As String = "abcdefghi"

Public Const FACE\_COLOURS As String = "bgorwy"

Public ReadOnly CORNER\_NAMES(0 To 7) As String

Public ReadOnly EDGE\_NAMES(0 To 11) As String

'ToDo: Pass Around

'Stickers with respect to top, i.e. changes based on 'top' face

Public StickersWRTTop(0 To 4, 0 To 8) As Char

Public Enum Faces

W

G

R

B

Y

O

End Enum

End Module

'TODO SPLIT FURTHER INTO SECTIONS

Module PublicFunctions

'Returns the number of each letter in a 2d char. array(,) as an array(0 To 5) i.e. ("bgorwy") b:0, g:1 etc...

Public Function NumberInCharArray(ByVal chararray(,) As Char) As Integer()

If chararray.Length < 1 Then

Return Nothing

End If

Dim number(0 To 5) As Integer

number = SetArrayToZero(number)

For i = 0 To UBound(chararray, 1)

For j = 0 To UBound(chararray, 2)

number(InStr(FACE\_COLOURS.ToUpper(), chararray(i, j)) - 1) += 1

Next

Next

Return number

End Function

Private Function SetArrayToZero(ByRef array() As Integer) As Integer()

For i = 0 To UBound(array)

array(i) = 0

Next

Return array

End Function

'Returns location of value in 1d array()

Public Function LinerarSearch(ByVal array() As Char, ByVal value As Char) As Integer

For i = 0 To UBound(array)

If array(i) = value Then

Return i

End If

Next

Return -1

End Function

'Returns True if anything other than value is IN 1d array() of integers

Public Function OtherIn1DIntArray(ByVal array() As Integer, ByVal value As Integer)

For i = 0 To UBound(array)

If array(i) <> value Then

Return True

Exit Function

End If

Next

Return False

End Function

'Converts colour character to word

Public Function ColourChar2Word(ByVal colourchar As Char)

Dim colourword As String

Select Case colourchar

Case "W"

colourword = "White"

Case "R"

colourword = "Red"

Case "B"

colourword = "Blue"

Case "Y"

colourword = "Yellow"

Case "O"

colourword = "Orange"

Case "G"

colourword = "Green"

Case Else

colourword = "ERROR"

End Select

Return colourword

End Function

'Converts colour char to face number

Public Function ColourChar2FaceNumber(ByVal colourchar As Char)

Dim x As Integer

Select Case colourchar

Case "W"

x = 0

Case "G"

x = 1

Case "R"

x = 2

Case "B"

x = 3

Case "Y"

x = 4

Case "O"

x = 5

End Select

Return x

End Function

'Converts face number to face colour char

Public Function FaceNumber2ColourChar(ByVal number As Integer)

Dim x As Char

Select Case number

Case 0

x = "W"

Case 1

x = "G"

Case 2

x = "R"

Case 3

x = "B"

Case 4

x = "Y"

Case 5

x = "O"

End Select

Return x

End Function

Public Sub InstantiateBlocks(ByRef corners() As Corner, ByRef edges() As Edge)

FillNameArrays()

' Dim corners(7) As Corner

For i = 0 To 7

corners(i) = New Corner

corners(i).Name = CORNER\_NAMES(i)

Next

'Dim edges(11) As Edge

For i = 0 To 11

edges(i) = New Edge

edges(i).Name = EDGE\_NAMES(i)

Next

'Return {corners, edges}

End Sub

Private Sub FillNameArrays()

FillCorners()

FillEdges()

End Sub

Private Sub FillCorners()

CORNER\_NAMES(0) = "WRB"

CORNER\_NAMES(1) = "WOB"

CORNER\_NAMES(2) = "WOG"

CORNER\_NAMES(3) = "WRG"

CORNER\_NAMES(4) = "YRB"

CORNER\_NAMES(5) = "YRG"

CORNER\_NAMES(6) = "YOG"

CORNER\_NAMES(7) = "YOB"

End Sub

Private Sub FillEdges()

EDGE\_NAMES(0) = "WR"

EDGE\_NAMES(1) = "WB"

EDGE\_NAMES(2) = "WO"

EDGE\_NAMES(3) = "WG"

EDGE\_NAMES(4) = "RB"

EDGE\_NAMES(5) = "OB"

EDGE\_NAMES(6) = "OG"

EDGE\_NAMES(7) = "RG"

EDGE\_NAMES(8) = "YR"

EDGE\_NAMES(9) = "YG"

EDGE\_NAMES(10) = "YO"

EDGE\_NAMES(11) = "YB"

End Sub

'TODO Split up these conversion functions

Private Function ConvertCorners(ByVal Cube(,) As Char, ByVal corners() As Corner, ByVal top As Char, ByVal front As Char, ByRef stickerArray(,) As Char) As Corner()

'Dim corners(7) As Corner

'corners = InstantiateBlocks()(0)

Dim sticker As Char

Dim faceColour As Char

'Converts top then bottom

For i = 0 To 1

' for each the top and side four faces

For face = 0 To 4

' for each corner sticker

For cornerSticker = 0 To 8 Step 2

' not the middle sticker

If cornerSticker <> 4 Then

' searches the top half of the cube only

If face = 0 Or (face > 0 And cornerSticker < 4) Then

' gets the sticker colour

sticker = StickersWRTTop(face, cornerSticker)

If sticker = "W" Or sticker = "Y" Then

Dim cornerTri As New CornerTriplet

'sets the primary sticker of cornerTri

cornerTri.Corners(0, "x") = face : cornerTri.Corners(0, "y") = cornerSticker

'gets the colour of the cube face that the primary sticker is on

faceColour = StickersWRTTop(cornerTri.Corners(0, "x"), 4)

' Returns a CornerTriplet Object

cornerTri = AdjacentCornersWR(cornerTri.Corners(0))

Dim cornerString As String = ""

Dim cornerColours(2) As Char

Dim secondaryFaceColour, secondaryFaceRotation As Char

For cornerBlockSticker = 0 To 2

Dim colourStore As Char

colourStore = StickersWRTTop(cornerTri.Corners(cornerBlockSticker, "x"), cornerTri.Corners(cornerBlockSticker, "y"))

If colourStore = "W" Or colourStore = "Y" Then

cornerColours(0) = colourStore

ElseIf colourStore = "R" Or colourStore = "O" Then

secondaryFaceColour = colourStore

secondaryFaceRotation = StickersWRTTop(cornerTri.Corners(cornerBlockSticker, "x"), 4)

cornerColours(1) = colourStore

ElseIf colourStore = "G" Or colourStore = "B" Then

cornerColours(2) = colourStore

Else

MsgBox("ERROR: " + cornerBlockSticker.ToString() + ", " + colourStore.ToString())

End If

Next cornerBlockSticker

'needed so that they are set in the correct order

For k = 0 To 2

cornerString += cornerColours(k)

Next k

For l = 0 To UBound(CORNER\_NAMES)

If CORNER\_NAMES(l) = cornerString Then

corners(l).PrimaryFace = sticker

corners(l).SecondaryFace = secondaryFaceColour

If i = 0 Then

Select Case face

Case 0 ' Top

Select Case cornerSticker

Case 0, 6

corners(l).Position.x = -1

Case 2, 8

corners(l).Position.x = 1

End Select

Select Case cornerSticker

Case 0, 2

corners(l).Position.z = -1

Case 6, 8

corners(l).Position.z = 1

End Select

Case 1 ' Left

corners(l).Position.x = -1

corners(l).Position.z = 1 - Math.Abs(cornerSticker - 2)

Case 2 ' Back

corners(l).Position.x = Math.Abs(cornerSticker - 2) - 1

corners(l).Position.z = -1

Case 3 ' Right

corners(l).Position.x = 1

corners(l).Position.z = 1 - cornerSticker

Case 4 ' Front

corners(l).Position.x = cornerSticker - 1

corners(l).Position.z = 1

End Select

' Only searching top half, & it's a corner therefore:

corners(l).Position.y = 1

Else

Select Case face

Case 0 ' Top

Select Case cornerSticker

Case 0, 6

corners(l).Position.x = 1

Case 2, 8

corners(l).Position.x = -1

End Select

Select Case cornerSticker

Case 0, 2

corners(l).Position.z = -1

Case 6, 8

corners(l).Position.z = 1

End Select

Case 1 ' Left

corners(l).Position.x = 1

corners(l).Position.z = 1 - Math.Abs(cornerSticker - 2)

Case 2 ' Back

corners(l).Position.x = cornerSticker - 1

corners(l).Position.z = -1

Case 3 ' Right

corners(l).Position.x = -1

corners(l).Position.z = 1 - cornerSticker

Case 4 ' Front

corners(l).Position.x = Math.Abs(cornerSticker - 2) - 1

corners(l).Position.z = 1

End Select

' Only searching bottom half, & it's a corner therefore:

corners(l).Position.y = -1

End If

'For x = 0 To 2

' Console.WriteLine("L: {0}, Pos{1}: {2}", l, x, corners(l).Position(x))

'Next

corners(l).Rotation = faceColour

corners(l).SecondaryRotation = secondaryFaceRotation

WriteCornersToFile(corners)

'Console.WriteLine("L: {0}, ROT: {1}", l, corners(l).Rotation)

For m = 0 To 2

corners(l).Colours(m) = cornerColours(m)

'Console.WriteLine("L: {0}, COl{2}: {1}", l, corners(l).Colours(m), m)

Next m

Exit For

End If

Next l

End If

End If

End If

Next cornerSticker

Next face

WriteCornersToFile(corners)

GiveRespect(stickerArray, Opposite(top), front)

Next

Return corners

End Function

Private Function ConvertEdges(ByVal Cube(,) As Char, ByVal edges() As Edge, ByVal top As Char, ByVal front As Char, ByRef stickerArray(,) As Char) As Edge()

'Dim edges(7) As Edge

'edges = InstantiateBlocks()(1)

Dim sticker As Char

Dim faceColour As Char

'Converts top then bottom

For i = 0 To 1

' for each the top and side four faces

For face = 0 To 4

' for each corner sticker

For edgeSticker = 1 To 7 Step 2

' searches the top half of the cube only

If face = 0 Or (face > 0 And edgeSticker < 4) Then

Dim edgePair As New EdgePair

' gets the sticker colour

sticker = StickersWRTTop(face, edgeSticker)

edgePair.Edges(0, "x") = face : edgePair.Edges(0, "y") = edgeSticker

edgePair = AdjacentEdgesWR(edgePair.Edges(0))

Dim secondSticker As Char = StickersWRTTop(edgePair.Edges(1, "x"), edgePair.Edges(1, "y"))

If (sticker = "W" Or sticker = "Y") Or ((sticker = "R" Or sticker = "O") And (secondSticker <> "W" And secondSticker <> "Y")) Then

'sets the primary sticker of edgePair

edgePair.Edges(0, "x") = face : edgePair.Edges(0, "y") = edgeSticker

'gets the colour of the cube face that the primary sticker is on

faceColour = StickersWRTTop(edgePair.Edges(0, "x"), 4)

edgePair = AdjacentEdgesWR(edgePair.Edges(0))

Dim edgeString As String = ""

Dim edgeColours(1) As Char

Dim colourStore, colourStore2 As Char

colourStore = StickersWRTTop(edgePair.Edges(0, "x"), edgePair.Edges(0, "y"))

colourStore2 = StickersWRTTop(edgePair.Edges(1, "x"), edgePair.Edges(1, "y"))

If (colourStore = "W" Or colourStore = "Y") Then

edgeColours(0) = colourStore

edgeColours(1) = colourStore2

ElseIf (colourStore2 = "W" Or colourStore2 = "Y") Then

edgeColours(0) = colourStore2

edgeColours(1) = colourStore

ElseIf (colourStore = "R" Or colourStore = "O") Then

edgeColours(0) = colourStore

edgeColours(1) = colourStore2

ElseIf (colourStore2 = "R" Or colourStore2 = "O") Then

edgeColours(0) = colourStore2

edgeColours(1) = colourStore

End If

For k = 0 To 1

edgeString += edgeColours(k)

Next k

For l = 0 To UBound(EDGE\_NAMES)

If EDGE\_NAMES(l) = edgeString Then

edges(l).PrimaryFace = sticker

If i = 0 Then

Select Case face

Case 0 ' Top

edges(l).Position.y = 1

Select Case edgeSticker

Case 1

edges(l).Position.x = 0

edges(l).Position.z = -1

Case 3

edges(l).Position.x = -1

edges(l).Position.z = 0

Case 5

edges(l).Position.x = 1

edges(l).Position.z = 0

Case 7

edges(l).Position.x = 0

edges(l).Position.z = 1

End Select

Case 1 ' Left

edges(l).Position.x = -1

If edgeSticker = 1 Then

edges(l).Position.y = 1

edges(l).Position.z = 0

ElseIf edgeSticker = 3 Then

edges(l).Position.y = 0

edges(l).Position.z = -1

End If

Case 2 ' Back

edges(l).Position.z = -1

If edgeSticker = 1 Then

edges(l).Position.x = 0

edges(l).Position.y = 1

ElseIf edgeSticker = 3 Then

edges(l).Position.x = 1

edges(l).Position.y = 0

End If

Case 3 ' Right

edges(l).Position.x = 1

If edgeSticker = 1 Then

edges(l).Position.y = 1

edges(l).Position.z = 0

ElseIf edgeSticker = 3 Then

edges(l).Position.y = 0

edges(l).Position.z = 1

End If

Case 4 ' Front

edges(l).Position.z = 1

If edgeSticker = 1 Then

edges(l).Position.x = 0

edges(l).Position.y = 1

ElseIf edgeSticker = 3 Then

edges(l).Position.x = -1

edges(l).Position.y = 0

End If

End Select

Else

'lower half

Select Case face

Case 0 ' Top

edges(l).Position.y = -1

Select Case edgeSticker

Case 1

edges(l).Position.x = 0

edges(l).Position.z = -1

Case 3

edges(l).Position.x = 1

edges(l).Position.z = 0

Case 5

edges(l).Position.x = -1

edges(l).Position.z = 0

Case 7

edges(l).Position.x = 0

edges(l).Position.z = 1

End Select

Case 1 ' Left

edges(l).Position.x = 1

If edgeSticker = 1 Then

edges(l).Position.y = -1

edges(l).Position.z = 0

ElseIf edgeSticker = 3 Then

edges(l).Position.y = 0

edges(l).Position.z = -1

End If

Case 2 ' Back

edges(l).Position.z = -1

If edgeSticker = 1 Then

edges(l).Position.x = 0

edges(l).Position.y = -1

ElseIf edgeSticker = 3 Then

edges(l).Position.x = -1

edges(l).Position.y = 0

End If

Case 3 ' Right

edges(l).Position.x = -1

If edgeSticker = 1 Then

edges(l).Position.y = -1

edges(l).Position.z = 0

ElseIf edgeSticker = 3 Then

edges(l).Position.y = 0

edges(l).Position.z = 1

End If

Case 4 ' Front

edges(l).Position.z = 1

If edgeSticker = 1 Then

edges(l).Position.x = 0

edges(l).Position.y = -1

ElseIf edgeSticker = 3 Then

edges(l).Position.x = 1

edges(l).Position.y = 0

End If

End Select

End If

edges(l).Rotation = faceColour

For m = 0 To 1

edges(l).Colours(m) = edgeColours(m)

'Console.WriteLine("L: {0}, COl{2}: {1}", l, corners(l).Colours(m), m)

Next m

Exit For

ElseIf l = UBound(EDGE\_NAMES) And EDGE\_NAMES(l) <> edgeString Then

MsgBox("EdgeString not recognised: " & edgeString)

End If

Next l

End If

End If

Next edgeSticker

Next face

WriteEdgesToFile(edges)

GiveRespect(stickerArray, Opposite(top), front)

Next

Return edges

End Function

'TODO: pass Cube(,) around; Todo split for edges and corners - wont be called twice

Public Function CreateBlocks(ByVal Cube(,) As Char, Optional top As Char = "W", Optional front As Char = "R") As Block()()

'include checks from classes

Dim corners(7) As Corner

Dim edges(11) As Edge

'TODO MAYBE REARRANGE SO GIVE RESPECT IS ONLY CALLED 2 NOT 4 TIMES

InstantiateBlocks(corners, edges)

GiveRespect(Cube, top, front)

corners = ConvertCorners(Cube, corners, top, front, Cube)

GiveRespect(Cube, top, front)

edges = ConvertEdges(Cube, edges, top, front, Cube)

For Each cornerCube In corners

If (Not cornerCube.checkPosition()) Or (Not cornerCube.checkPosition()) Then

MsgBox("Error, corner cube: " & cornerCube.Name)

End If

Next

For Each edgeCube In edges

If (Not edgeCube.checkPosition()) Or (Not edgeCube.checkPosition()) Then

MsgBox("Error, edge cube: " & edgeCube.Name)

End If

Next

Return {corners, edges}

End Function

Public Sub WriteCornersToFile(ByVal corners As Corner(), Optional filename As String = "CORNERS.txt")

FileOpen(1, filename, OpenMode.Output)

For i = 0 To corners.Length - 1

Print(1, corners(i).Name & ", ")

Print(1, corners(i).Colours(0))

Print(1, corners(i).Colours(1))

Print(1, corners(i).Colours(2) & ", ")

Print(1, corners(i).Position.x)

Print(1, corners(i).Position.y)

Print(1, corners(i).Position.z & ", 1st:")

Print(1, corners(i).PrimaryFace & "->")

Print(1, corners(i).Rotation & ", 2nd:")

Print(1, corners(i).SecondaryFace & "->")

Print(1, corners(i).SecondaryRotation)

PrintLine(1)

Next

FileClose(1)

End Sub

Public Sub WriteEdgesToFile(ByVal edges As Edge(), Optional filename As String = "EDGES.txt")

FileOpen(1, filename, OpenMode.Output)

For i = 0 To edges.Length - 1

Print(1, edges(i).Name & ", ")

Print(1, edges(i).Colours(0))

Print(1, edges(i).Colours(1) & ", ")

Print(1, edges(i).Position.x)

Print(1, edges(i).Position.y)

Print(1, edges(i).Position.z & ", ")

Print(1, edges(i).PrimaryFace & "->")

Print(1, edges(i).Rotation)

PrintLine(1)

Next

FileClose(1)

End Sub

'TODO hashtable

'Returns the opposte coloured face to the colour given as an argument

Public Function Opposite(ByVal colour As Char) As Char

'Opposites:

'W/Y, R/O, B/G

Dim opp As Char

Select Case colour

Case "W"

opp = "Y"

Case "R"

opp = "O"

Case "B"

opp = "G"

Case "Y"

opp = "W"

Case "O"

opp = "R"

Case "G"

opp = "B"

End Select

Return opp

End Function

'Returns associated face colours to face passed as an argument (0 = self, 1-4 = sides, 5 = opposite)

Public Function AssociatedFaces(face As Char) As Char()

Dim faces(0 To 5) As Char

Dim x As Integer = 1

faces(0) = face

faces(5) = Opposite(face)

For Each j In FACE\_COLOURS.ToUpper().ToCharArray()

If j <> faces(0) And j <> faces(5) Then

faces(x) = j

x += 1

End If

Next

Return faces

End Function

'Returns associated face colours to face passed as an argument (0 = self, 1-4 = sides clockwise based on front, 5 = opposite)

Public Function AssociatedFaces(top As Char, front As Char) As Char()

Dim faces(0 To 5) As Char

Dim x As Integer = 1

faces(0) = top

faces(5) = Opposite(top)

Dim sides(4) As Char

Select Case top

Case "W" : sides(0) = "R" : sides(1) = "G" : sides(2) = "O" : sides(3) = "B"

Case "Y" : sides(3) = "R" : sides(2) = "G" : sides(1) = "O" : sides(0) = "B"

Case "R" : sides(0) = "B" : sides(1) = "Y" : sides(2) = "G" : sides(3) = "W"

Case "O" : sides(3) = "B" : sides(2) = "Y" : sides(1) = "G" : sides(0) = "W"

Case "B" : sides(0) = "Y" : sides(1) = "R" : sides(2) = "W" : sides(3) = "O"

Case "G" : sides(3) = "Y" : sides(2) = "R" : sides(1) = "W" : sides(0) = "O"

End Select

Dim positionOfFront As Integer

positionOfFront = LinerarSearch(sides, front)

For i = 0 To 3

faces(i + 1) = sides((positionOfFront + i + 1) Mod 4)

Next

Return faces

End Function

'Returns the adjacent middle edge cube (x, y) to the middle edge cube given (a, b) - adjacent across 2 faces, not the same face

Public Function AdjacentEdge(ByVal a As Integer, ByVal b As Integer) As Integer()

'Adjacent sides = (0,1)(5,7) (0,3)(1,1) (0,5)(3,1) (0,7)(2,1)

'(a,b)(x,y) (1,1)(0,3) (1,3)(5,3) (1,5)(2,3) (1,7)(4,3)

' (2,1)(0,7) (2,3)(1,5) (2,5)(3,3) (2,7)(4,1)

' (3,1)(0,5) (3,3)(2,5) (3,5)(5,5) (3,7)(4,5)

' (4,1)(2,7) (4,3)(1,7) (4,5)(3,7) (4,7)(5,1)

' (5,1)(4,7) (5,3)(1,3) (5,5)(3,5) (5,7)(0,1)

Dim x, y As Integer

Select Case a

Case 0

Select Case b

Case 1

x = 5 : y = 7

Case 3

x = 1 : y = 1

Case 5

x = 3 : y = 1

Case 7

x = 2 : y = 1

End Select

Case 1

Select Case b

Case 1

x = 0 : y = 3

Case 3

x = 5 : y = 3

Case 5

x = 2 : y = 3

Case 7

x = 4 : y = 3

End Select

Case 2

Select Case b

Case 1

x = 0 : y = 7

Case 3

x = 1 : y = 5

Case 5

x = 3 : y = 3

Case 7

x = 4 : y = 1

End Select

Case 3

Select Case b

Case 1

x = 0 : y = 5

Case 3

x = 2 : y = 5

Case 5

x = 5 : y = 5

Case 7

x = 4 : y = 5

End Select

Case 4

Select Case b

Case 1

x = 2 : y = 7

Case 3

x = 1 : y = 7

Case 5

x = 3 : y = 7

Case 7

x = 5 : y = 1

End Select

Case 5

Select Case b

Case 1

x = 4 : y = 7

Case 3

x = 1 : y = 3

Case 5

x = 3 : y = 5

Case 7

x = 0 : y = 1

End Select

End Select

Return {x, y}

End Function

'Returns the adjacent corner cubes (w, x)(y, z) to the corner cube given (a,b) - adjacent across 2 faces, not the same face

Public Function AdjacentConers(ByVal a As Integer, ByVal b As Integer) As Integer()

'Adjacent Coners = (0,0)(1,0)(5,6) (0,2)(5,8)(3,2) (0,6)(2,0)(1,2) (0,8)(3,0)(2,2)

'(a,b)(w,x)(y,z) (1,0)(5,6)(0,0) (1,2)(0,6)(2,0) (1,6)(4,6)(5,0) (1,8)(2,6)(4,0)

' (2,0)(1,2)(0,6) (2,2)(0,8)(3,0) (2,6)(4,0)(1,8) (2,8)(3,6)(4,2)

' (3,0)(2,2)(0,8) (3,2)(0,2)(5,8) (3,6)(4,2)(2,8) (3,8)(5,2)(4,8)

' (4,0)(1,8)(2,6) (4,2)(2,8)(3,6) (4,6)(5,0)(1,6) (4,8)(3,8)(5,2)

' (5,0)(1,6)(4,6) (5,2)(4,8)(3,8) (5,6)(0,0)(1,0) (5,8)(3,2)(0,2)

Dim w, x, y, z As Integer

Select Case a

Case 0

Select Case b

Case 0

w = 1 : x = 0 : y = 5 : z = 6

Case 2

w = 5 : x = 8 : y = 3 : z = 2

Case 6

w = 2 : x = 0 : y = 1 : z = 2

Case 8

w = 3 : x = 0 : y = 2 : z = 2

End Select

Case 1

Select Case b

Case 0

w = 5 : x = 6 : y = 0 : z = 0

Case 2

w = 0 : x = 6 : y = 2 : z = 0

Case 6

w = 4 : x = 6 : y = 5 : z = 0

Case 8

w = 2 : x = 6 : y = 4 : z = 0

End Select

Case 2

Select Case b

Case 0

w = 1 : x = 2 : y = 0 : z = 6

Case 2

w = 0 : x = 8 : y = 3 : z = 0

Case 6

w = 4 : x = 0 : y = 1 : z = 8

Case 8

w = 3 : x = 6 : y = 4 : z = 2

End Select

Case 3

Select Case b

Case 0

w = 2 : x = 2 : y = 0 : z = 8

Case 2

w = 0 : x = 2 : y = 5 : z = 8

Case 6

w = 4 : x = 2 : y = 2 : z = 8

Case 8

w = 5 : x = 2 : y = 4 : z = 8

End Select

Case 4

Select Case b

Case 0

w = 1 : x = 8 : y = 2 : z = 6

Case 2

w = 2 : x = 8 : y = 3 : z = 6

Case 6

w = 5 : x = 0 : y = 1 : z = 6

Case 8

w = 3 : x = 8 : y = 5 : z = 2

End Select

Case 5

Select Case b

Case 0

w = 1 : x = 6 : y = 4 : z = 6

Case 2

w = 4 : x = 8 : y = 3 : z = 8

Case 6

w = 0 : x = 0 : y = 1 : z = 0

Case 8

w = 3 : x = 2 : y = 0 : z = 2

End Select

End Select

Return {w, x, y, z}

End Function

'''<summary>Reassignes values to StickersWRTTop based on top face and front face</summary>

Public Sub GiveRespect(ByVal stickerArray(,) As Char, ByVal top As Char, ByVal front As Char)

' Coverts sticker data array to SWRTTop

' each face can be in 1 of 4 rotations - each method maps the face onto the WRTTop array

Try

If front = Opposite(top) Then

Throw New ArgumentException("The specified top and front face combination is impossible." & vbNewLine & "Top: " & top & vbNewLine & "Front: " & front)

End If

Catch ex As ArgumentException

MsgBox(ex.Message)

End Try

Dim face As Char

top = UCase(top)

'For each side face of cube

For sideFace = 0 To 4

face = AssociatedFaces(top, front)(sideFace)

Dim store(9) As Char

For i = 0 To 8

store(i) = stickerArray(ColourChar2FaceNumber(face), i)

Next

Select Case top

Case "W"

Select Case face

Case "O" : Method2(sideFace, store)

Case top : TopFace(sideFace, store, top, front)

Case Else : Method1(sideFace, store)

End Select

Case "Y"

Select Case face

Case "O" : Method1(sideFace, store)

Case top : TopFace(sideFace, store, top, front)

Case Else : Method2(sideFace, store)

End Select

Case "G"

Select Case face

Case top : TopFace(sideFace, store, top, front)

Case Else : Method3(sideFace, store)

End Select

Case "B"

Select Case face

Case top : TopFace(sideFace, store, top, front)

Case Else : Method4(sideFace, store)

End Select

Case "R"

Select Case face

Case "Y" : Method1(sideFace, store)

Case "W" : Method2(sideFace, store)

Case "B" : Method3(sideFace, store)

Case "G" : Method4(sideFace, store)

Case top : TopFace(sideFace, store, top, front)

End Select

Case "O"

Select Case face

Case "Y" : Method2(sideFace, store)

Case "W" : Method1(sideFace, store)

Case "B" : Method4(sideFace, store)

Case "G" : Method3(sideFace, store)

Case top : TopFace(sideFace, store, top, front)

End Select

End Select

Next

End Sub

'TODO RENAME

Private Sub Method1(ByVal faceNumber As Integer, ByVal faceStickers As Char())

For i = 0 To 8

StickersWRTTop(faceNumber, i) = faceStickers(i)

Next

End Sub

Private Sub Method2(ByVal faceNumber As Integer, ByVal faceStickers As Char())

For i = 0 To 8

StickersWRTTop(faceNumber, i) = faceStickers(8 - i)

Next

End Sub

Private Sub Method3(ByVal faceNumber As Integer, ByVal faceStickers As Char())

Dim count As Integer = 6

For i = 0 To 8

StickersWRTTop(faceNumber, i) = faceStickers(count Mod 10)

count += 7

Next

End Sub

Private Sub Method4(ByVal faceNumber As Integer, ByVal faceStickers As Char())

Dim count As Integer = 2

For i = 0 To 8

StickersWRTTop(faceNumber, i) = faceStickers(count Mod 10)

count += 3

Next

End Sub

Private Sub TopFace(ByVal faceNumber As Integer, ByVal faceStickers As Char(), ByVal top As Char, ByVal front As Char)

If front = "B" Then

Method3(faceNumber, faceStickers)

ElseIf front = "G" Then

Method4(faceNumber, faceStickers)

ElseIf front = "W" Then

If top = "O" Then

Method1(faceNumber, faceStickers)

Else

Method2(faceNumber, faceStickers)

End If

ElseIf front = "Y" Then

If top = "O" Then

Method2(faceNumber, faceStickers)

Else

Method1(faceNumber, faceStickers)

End If

ElseIf front = "R" Then

Select Case top

Case "W" : Method1(faceNumber, faceStickers)

Case "Y" : Method2(faceNumber, faceStickers)

Case "G" : Method3(faceNumber, faceStickers)

Case "B" : Method4(faceNumber, faceStickers)

End Select

ElseIf front = "O" Then

Select Case top

Case "W" : Method2(faceNumber, faceStickers)

Case "Y" : Method1(faceNumber, faceStickers)

Case "G" : Method4(faceNumber, faceStickers)

Case "B" : Method3(faceNumber, faceStickers)

End Select

End If

End Sub

Private Function SetUpCornerTriplets()

' 1st dim. = 'triplet index'

Static cornerTriples(,) As Coordinate = {{New Coordinate, New Coordinate, New Coordinate},

{New Coordinate, New Coordinate, New Coordinate},

{New Coordinate, New Coordinate, New Coordinate},

{New Coordinate, New Coordinate, New Coordinate}}

If cornerTriples(0, 0).x = 0 Then

For i = 0 To 3

cornerTriples(i, 0).x = 0

cornerTriples(i, 1).y = 0

cornerTriples(i, 2).y = 2

Next

cornerTriples(0, 0).y = 0 : cornerTriples(0, 1).x = 1 : cornerTriples(0, 2).x = 2

cornerTriples(1, 0).y = 2 : cornerTriples(1, 1).x = 2 : cornerTriples(1, 2).x = 3

cornerTriples(2, 0).y = 6 : cornerTriples(2, 1).x = 4 : cornerTriples(2, 2).x = 1

cornerTriples(3, 0).y = 8 : cornerTriples(3, 1).x = 3 : cornerTriples(3, 2).x = 4

End If

Return cornerTriples

End Function

'Returns the adjacent corner coordinates for the stickers WRRTTop array

Public Function AdjacentCornersWR(ByVal corner As Coordinate)

'TODO later:

' set up pairs and use a dictionary??? maybe??? or is this solution elegant enough??

Dim cornerArray(,) As Coordinate

Dim pairIndex As Integer

Dim argumentCornerIndex As Integer

'Dim adjacentOne, adjacentTwo As Coordinate

Dim triplet As New CornerTriplet

triplet.Corners(0, "x") = corner.x

triplet.Corners(0, "y") = corner.y

'should only be set up if the array is blank - but dont think works atm

cornerArray = SetUpCornerTriplets()

' Search array for Corner passed as argument

' from 0 to 7

For i = 0 To UBound(cornerArray, 2)

For j = 0 To UBound(cornerArray, 1)

If cornerArray(j, i).x = triplet.Corners(0, "x") And cornerArray(j, i).y = triplet.Corners(0, "y") Then

pairIndex = j

argumentCornerIndex = i

End If

Next

Next

Dim FirstPassDone As Boolean = False

'use 'pairIndex' to return other 2 coordinates

For i = 0 To UBound(cornerArray, 2)

If i <> argumentCornerIndex Then

If Not FirstPassDone Then

triplet.Corners(1, "x") = cornerArray(pairIndex, i).x : triplet.Corners(1, "y") = cornerArray(pairIndex, i).y

FirstPassDone = True

Else

triplet.Corners(2, "x") = cornerArray(pairIndex, i).x : triplet.Corners(2, "y") = cornerArray(pairIndex, i).y

End If

End If

Next

Return triplet

End Function

Private Function SetUpEdgePairs()

Static edgePairs(,) As Coordinate = {{New Coordinate, New Coordinate},

{New Coordinate, New Coordinate},

{New Coordinate, New Coordinate},

{New Coordinate, New Coordinate},

{New Coordinate, New Coordinate},

{New Coordinate, New Coordinate},

{New Coordinate, New Coordinate},

{New Coordinate, New Coordinate}}

' When SetUpCoordinatePairs is called again it will use the same value of cornerTriples(,)

' Thus (0,0).x != 0 ^-^

'nope!

'TODO CHANGE THIS BECAUSE 0,0 WILL BE 0!!!!

If edgePairs(0, 0).x = 0 Then

For i = 0 To 3

edgePairs(i, 0).x = 0

edgePairs(i, 1).y = 1

Next

For i = 4 To 7

edgePairs(i, 0).y = 3

edgePairs(i, 1).y = 5

Next

edgePairs(0, 0).y = 1 : edgePairs(0, 1).x = 2

edgePairs(1, 0).y = 3 : edgePairs(1, 1).x = 1

edgePairs(2, 0).y = 5 : edgePairs(2, 1).x = 3

edgePairs(3, 0).y = 7 : edgePairs(3, 1).x = 4

edgePairs(4, 0).x = 1 : edgePairs(4, 1).x = 2

edgePairs(5, 0).x = 2 : edgePairs(5, 1).x = 3

edgePairs(6, 0).x = 3 : edgePairs(6, 1).x = 4

edgePairs(7, 0).x = 4 : edgePairs(7, 1).x = 1

End If

Return edgePairs

End Function

Public Function AdjacentEdgesWR(ByVal edge As Coordinate)

Dim edgeArray As Coordinate(,)

Dim pairIndex As Integer

Dim argumentEdgeIndex As Integer

Dim pair As New EdgePair

pair.Edges(0, "x") = edge.x

pair.Edges(0, "y") = edge.y

edgeArray = SetUpEdgePairs()

' Search array for edge passed as an argument

' from 0 to 7

For i = 0 To UBound(edgeArray, 2)

For j = 0 To UBound(edgeArray, 1)

If edgeArray(j, i).x = pair.Edges(0, "x") And edgeArray(j, i).y = pair.Edges(0, "y") Then

pairIndex = j

argumentEdgeIndex = i

End If

Next

Next

For i = 0 To UBound(edgeArray, 2)

If i <> argumentEdgeIndex Then

pair.Edges(1, "x") = edgeArray(pairIndex, i).x : pair.Edges(1, "y") = edgeArray(pairIndex, i).y

End If

Next

Return pair

End Function

End Module

End Namespace

# Blocks

Public Module Blocks

Public MustInherit Class Block

Private \_name As String

Public Property Name() As String

Get

Return \_name

End Get

Set(ByVal value As String)

\_name = value

End Set

End Property

Private \_position As New Vector3x1()

Public Property Position As Vector3x1

Get

Return \_position

End Get

Set(value As Vector3x1)

\_position = New Vector3x1(value)

End Set

End Property

'probably isnt needed as wont it always be Colours(0)?

Private \_primaryFace As Char

Public Property PrimaryFace() As Char

Get

Return \_primaryFace

End Get

Set(value As Char)

\_primaryFace = value

End Set

End Property

'The colour of the face that the primary face is on

Private \_rotation As Char

Public Property Rotation() As Char

Get

Return \_rotation

End Get

Set(value As Char)

\_rotation = value

End Set

End Property

Private \_colours() As Char = {"X", "X", "X"}

Public Property Colours(ByVal index As Integer) As Char

Get

Return \_colours(index)

End Get

Set(value As Char)

\_colours(index) = value

End Set

End Property

Public Property Colours() As Char()

Get

Return \_colours

End Get

Set(value As Char())

\_colours = value

End Set

End Property

Public MustOverride Function checkPosition() As Boolean

Public Function checkRotation() As Boolean

For Each i In \_colours

If i = \_primaryFace Then

Return True

End If

Next

Return False

End Function

End Class

Public Class Corner

Inherits Block

Private \_secondaryFace As Char

Public Property SecondaryFace() As Char

Get

Return \_secondaryFace

End Get

Set(value As Char)

\_secondaryFace = value

End Set

End Property

Private \_secondaryFaceRotation As Char

Public Property SecondaryRotation() As Char

Get

Return \_secondaryFaceRotation

End Get

Set(value As Char)

\_secondaryFaceRotation = value

End Set

End Property

Public Overrides Function checkPosition() As Boolean

Dim \_position(2) As Single

\_position = Me.Position.Column

For i = 0 To 2

If \_position(i) <> -1 And \_position(i) <> 1 Then

Return False

End If

Next

Return True

End Function

End Class

Public Class Edge

Inherits Block

Public Overrides Function checkPosition() As Boolean

Dim \_position As Single()

Dim counter As Integer = 0

\_position = Me.Position.Column

'needs to be in the middle on at least one face to be an edge

If \_position(0) <> 0 And \_position(1) <> 0 And \_position(2) <> 0 Then

Return False

Else

For i = 0 To 2

If \_position(i) = 0 Then

counter += 1

End If

If counter > 1 Then

Return False

End If

Next

End If

Return True

End Function

End Class

Public Class Middle

Inherits Block

Public Overrides Function checkPosition() As Boolean

Throw New NotImplementedException()

End Function

End Class

End Module

# Moves

Module Moves

'Matrix rotations are counter clockwise as viewed from the positive end of the axis

Public Enum Axis

X

Y

Z

End Enum

Public Enum Direction

NoChange = 0

Clockwise = -1

Anticlockwise = 1

HalfTurn = 2

End Enum

''' <summary>

''' Rotates the specified face

''' </summary>

Public Sub RotateFace(ByVal face As MyPublic.Faces, ByVal direction As Direction, ByRef corners() As Corner, ByRef edges() As Edge, ByVal top As Char, ByRef front As Char, ByRef Instructions As String)

Dim axis As Axis

Dim axisCoordinate As Integer

Dim faces(6) As Char

faces = MyPublic.AssociatedFaces(top, front)

'actual direction of rotation - with regards to the face not the axis

Dim directionOfInstruction As Direction = direction

If direction <> Moves.Direction.NoChange Then

Select Case face.ToString

Case faces(0) ' top

axis = axis.Y

axisCoordinate = 1

Instructions += "U"

Case faces(1) ' left

axis = axis.X

If direction <> direction.HalfTurn Then

direction = -direction

End If

axisCoordinate = -1

Instructions += "L"

Case faces(2) ' back

axis = axis.Z

If direction <> direction.HalfTurn Then

direction = -direction

End If

axisCoordinate = -1

Instructions += "B"

Case faces(3) ' right

axis = axis.X

axisCoordinate = 1

Instructions += "R"

Case faces(4) ' front

axis = axis.Z

axisCoordinate = 1

Instructions += "F"

Case faces(5) ' bottom

axis = axis.Y

If direction <> direction.HalfTurn Then

direction = -direction

End If

axisCoordinate = -1

Instructions += "D"

End Select

If directionOfInstruction = Moves.Direction.Clockwise Then

Instructions += " "

ElseIf directionOfInstruction = Moves.Direction.Anticlockwise Then

Instructions += "' "

ElseIf directionOfInstruction = Moves.Direction.HalfTurn Then

Instructions += "2 "

End If

Else

Return

End If

Dim angle As Double = (Math.PI / 2) \* direction

Dim rotationMatrix As Matrix3x3

Select Case axis

Case Moves.Axis.X

rotationMatrix = New Matrix3x3(New Vector3x1(1, 0, 0),

New Vector3x1(0, Math.Cos(angle), -Math.Sin(angle)),

New Vector3x1(0, Math.Sin(angle), Math.Cos(angle)))

Case Moves.Axis.Y

rotationMatrix = New Matrix3x3(New Vector3x1(Math.Cos(angle), 0, Math.Sin(angle)),

New Vector3x1(0, 1, 0),

New Vector3x1(-Math.Sin(angle), 0, Math.Cos(angle)))

Case Moves.Axis.Z

rotationMatrix = New Matrix3x3(New Vector3x1(Math.Cos(angle), -Math.Sin(angle), 0),

New Vector3x1(Math.Sin(angle), Math.Cos(angle), 0),

New Vector3x1(0, 0, 1))

Case Else

MsgBox("Error")

rotationMatrix = New Matrix3x3()

Return

End Select

For Each cubie In corners

Dim coordinate As Integer

Select Case axis

Case Moves.Axis.X

coordinate = cubie.Position.x

Case Moves.Axis.Y

coordinate = cubie.Position.y

Case Moves.Axis.Z

coordinate = cubie.Position.z

End Select

If coordinate = axisCoordinate Then

cubie.Position = rotationMatrix \* cubie.Position

Dim faceColours() As Char

'could maybe move this bit so its not called every time - 1nce per face

If face.ToString() <> front And face.ToString() <> MyPublic.Opposite(front) Then

faceColours = MyPublic.AssociatedFaces(face.ToString(), front)

ElseIf face.ToString() = front Then

faceColours = MyPublic.AssociatedFaces(face.ToString(), MyPublic.Opposite(top))

ElseIf face.ToString() = MyPublic.Opposite(front) Then

faceColours = MyPublic.AssociatedFaces(face.ToString(), top)

End If

Dim currentFaceForRotation As Integer

For i = 1 To 4

If cubie.Rotation = faceColours(i) Then

currentFaceForRotation = i

End If

Next

Dim currentFaceForSecondaryRotation As Integer

For i = 1 To 4

If cubie.SecondaryRotation = faceColours(i) Then

currentFaceForSecondaryRotation = i

End If

Next

If cubie.Rotation = face.ToString() Then

Select Case directionOfInstruction

Case Moves.Direction.Clockwise

cubie.SecondaryRotation = faceColours((currentFaceForSecondaryRotation Mod 4) + 1)

Case Moves.Direction.Anticlockwise

cubie.SecondaryRotation = faceColours(((currentFaceForSecondaryRotation + 2) Mod 4) + 1)

Case Moves.Direction.HalfTurn

cubie.SecondaryRotation = faceColours(((currentFaceForSecondaryRotation + 1) Mod 4) + 1)

End Select

ElseIf cubie.SecondaryRotation = face.ToString() Then

Select Case directionOfInstruction

Case Moves.Direction.Clockwise

cubie.Rotation = faceColours((currentFaceForRotation Mod 4) + 1)

Case Moves.Direction.Anticlockwise

cubie.Rotation = faceColours(((currentFaceForRotation + 2) Mod 4) + 1)

Case Moves.Direction.HalfTurn

cubie.Rotation = faceColours(((currentFaceForRotation + 1) Mod 4) + 1)

End Select

Else

Select Case directionOfInstruction

Case Moves.Direction.Clockwise

cubie.Rotation = faceColours((currentFaceForRotation Mod 4) + 1)

cubie.SecondaryRotation = faceColours((currentFaceForSecondaryRotation Mod 4) + 1)

Case Moves.Direction.Anticlockwise

cubie.Rotation = faceColours(((currentFaceForRotation + 2) Mod 4) + 1)

cubie.SecondaryRotation = faceColours(((currentFaceForSecondaryRotation + 2) Mod 4) + 1)

Case Moves.Direction.HalfTurn

cubie.Rotation = faceColours(((currentFaceForRotation + 1) Mod 4) + 1)

cubie.SecondaryRotation = faceColours(((currentFaceForSecondaryRotation + 1) Mod 4) + 1)

End Select

End If

End If

Next

For Each cubie In edges

Dim coordinate As Integer

Select Case axis

Case Moves.Axis.X

coordinate = cubie.Position.x

Case Moves.Axis.Y

coordinate = cubie.Position.y

Case Moves.Axis.Z

coordinate = cubie.Position.z

End Select

If coordinate = axisCoordinate Then

cubie.Position = rotationMatrix \* cubie.Position

If cubie.Rotation <> face.ToString() Then

Dim faceColours() As Char

'could maybe move this bit so its not called every time - 1nce per face

If face.ToString() <> front And face.ToString() <> MyPublic.Opposite(front) Then

faceColours = MyPublic.AssociatedFaces(face.ToString(), front)

ElseIf face.ToString() = front Then

faceColours = MyPublic.AssociatedFaces(face.ToString(), MyPublic.Opposite(top))

ElseIf face.ToString() = MyPublic.Opposite(front) Then

faceColours = MyPublic.AssociatedFaces(face.ToString(), top)

End If

Dim currentFace As Integer

For i = 1 To 4

If cubie.Rotation = faceColours(i) Then

currentFace = i

End If

Next

Select Case directionOfInstruction

Case Moves.Direction.Clockwise

cubie.Rotation = faceColours((currentFace Mod 4) + 1)

Case Moves.Direction.Anticlockwise

cubie.Rotation = faceColours(((currentFace + 2) Mod 4) + 1)

Case Moves.Direction.HalfTurn

cubie.Rotation = faceColours(((currentFace + 1) Mod 4) + 1)

End Select

End If

End If

Next

' MyPublic.WriteCornersToFile(""corners, "RotatedCorners.txt")

'MyPublic.WriteEdgesToFile(edges, "RotatedEdges.txt")

End Sub

''' <summary>

''' Rotates the cube in a specified direction

''' </summary>

''' <param name="direction"> Towards, Backwards, Left, Right, UpsideDown. </param>

Private Sub RotateCube(ByVal direction As Direction, ByVal axis As Axis, ByRef corners() As Corner, ByRef edges() As Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim angle As Double = (Math.PI / 2) \* direction

Dim rotationMatrix As Matrix3x3

Select Case axis

Case Moves.Axis.X

rotationMatrix = New Matrix3x3(New Vector3x1(1, 0, 0),

New Vector3x1(0, Math.Cos(angle), -Math.Sin(angle)),

New Vector3x1(0, Math.Sin(angle), Math.Cos(angle)))

If direction <> direction.NoChange Then

Instructions += "X"

End If

Case Moves.Axis.Y

rotationMatrix = New Matrix3x3(New Vector3x1(Math.Cos(angle), 0, Math.Sin(angle)),

New Vector3x1(0, 1, 0),

New Vector3x1(-Math.Sin(angle), 0, Math.Cos(angle)))

If direction <> Direction.NoChange Then

Instructions += "Y"

End If

Case Moves.Axis.Z

rotationMatrix = New Matrix3x3(New Vector3x1(Math.Cos(angle), -Math.Sin(angle), 0),

New Vector3x1(Math.Sin(angle), Math.Cos(angle), 0),

New Vector3x1(0, 0, 1))

If direction <> Direction.NoChange Then

Instructions += "Z"

End If

Case Else

MsgBox("Error")

rotationMatrix = New Matrix3x3()

Return

End Select

If direction <> direction.NoChange Then

If direction = Moves.Direction.Clockwise Then

Instructions += " "

ElseIf direction = Moves.Direction.Anticlockwise Then

Instructions += "' "

ElseIf direction = Moves.Direction.HalfTurn Then

Instructions += "2 "

End If

End If

' Dim tmp As New Vector3x1

For Each cubie In corners

'Console.WriteLine(cubie.Position.x & ", " & cubie.Position.y & ", " & cubie.Position.z)

'tmp = rotationMatrix \* cubie.Position

'Console.WriteLine(tmp.x & ", " & tmp.y & ", " & tmp.z)

cubie.Position = rotationMatrix \* cubie.Position

Next

For Each cubie In edges

cubie.Position = rotationMatrix \* cubie.Position

Next

'MyPublic.WriteCornersToFile(corners, "RotatedCorners.txt")

'MyPublic.WriteEdgesToFile(edges, "RotatedEdges.txt")

End Sub

''' <summary>

''' Rotates the cube so that a specific face is on top

''' </summary>

Public Sub RotateFaceToTop(ByVal newTop As MyPublic.Faces, ByRef corners() As Corner, ByRef edges() As Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim faces(6) As Char

faces = MyPublic.AssociatedFaces(top, front)

Dim direction As Direction

Dim axis As Axis

Select Case newTop.ToString

Case faces(0) ' current top

axis = Moves.Axis.X

direction = Moves.Direction.NoChange

Case faces(1) ' current left

axis = Moves.Axis.Z

direction = Moves.Direction.Clockwise

Case faces(2) ' current back

axis = Moves.Axis.X

front = faces(0)

direction = Moves.Direction.Anticlockwise

Case faces(3) ' current right

axis = Moves.Axis.Z

direction = Moves.Direction.Anticlockwise

Case faces(4) ' current front

axis = Moves.Axis.X

front = faces(5)

direction = Moves.Direction.Clockwise

Case faces(5) ' current bottom

axis = Moves.Axis.Z

direction = Moves.Direction.HalfTurn

End Select

RotateCube(direction, axis, corners, edges, top, front, Instructions)

top = newTop.ToString()

'Instructions += "(T:" + top + ") "

End Sub

''' <summary>

''' Rotates the cube so that a specific face is on front

''' </summary>

Public Sub RotateFaceToFront(ByVal newFront As MyPublic.Faces, ByRef corners() As Corner, ByRef edges() As Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim faces(6) As Char

faces = MyPublic.AssociatedFaces(top, front)

Dim direction As Direction

Dim axis As Axis

Select Case newFront.ToString

Case faces(0) ' current top

axis = Moves.Axis.X

top = faces(2)

direction = Moves.Direction.Anticlockwise

Case faces(1) ' current left

axis = Moves.Axis.Y

direction = Moves.Direction.Anticlockwise

Case faces(2) ' current back

axis = Moves.Axis.Y

direction = Moves.Direction.HalfTurn

Case faces(3) ' current right

axis = Moves.Axis.Y

direction = Moves.Direction.Clockwise

Case faces(4) ' current front

axis = Moves.Axis.X

direction = Moves.Direction.NoChange

Case faces(5) ' current bottom

axis = Moves.Axis.X

top = faces(4)

direction = Moves.Direction.Clockwise

End Select

RotateCube(direction, axis, corners, edges, top, front, Instructions)

front = newFront.ToString()

'Instructions += "(F:" + front + ") "

End Sub

Public Sub RotateFaceToLeft(ByVal newLeft As MyPublic.Faces, ByRef corners() As Corner, ByRef edges() As Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim faces(6) As Char

faces = MyPublic.AssociatedFaces(top, front)

Dim direction As Direction

Dim axis As Axis

Select Case newLeft.ToString

Case faces(0) ' current top

axis = Moves.Axis.Z

direction = Moves.Direction.Anticlockwise

top = faces(3)

Case faces(1) ' current left

axis = Moves.Axis.Z

direction = Moves.Direction.NoChange

Case faces(2) ' current back

axis = Moves.Axis.Y

direction = Moves.Direction.Anticlockwise

front = faces(1)

Case faces(3) ' current right

axis = Moves.Axis.Y

direction = Moves.Direction.HalfTurn

front = faces(2)

Case faces(4) ' current front

axis = Moves.Axis.Y

direction = Moves.Direction.Clockwise

front = faces(3)

Case faces(5) ' current bottom

axis = Moves.Axis.Z

direction = Moves.Direction.Clockwise

top = faces(1)

End Select

RotateCube(direction, axis, corners, edges, top, front, Instructions)

'Instructions += "(F:" + front + ")"

'Instructions += "(T:" + top + ") "

End Sub

End Module

# Algorithms

Imports PublicValues = RubiksCubeSolver\_v2\_0.MyPublic.PublicVars

Module Algorithms

'ToDo Middle Edge LEft and Right algorithms

Public Sub MiddleLeftAlgorithm(ByVal faces() As PublicValues.Faces, ByRef corners() As Blocks.Corner,

ByRef edges() As Blocks.Edge, ByVal top As Char, ByVal front As Char, ByRef Instructions As String)

RotateFace(faces(5), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(1), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(1), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(4), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(4), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

End Sub

Public Sub MiddleRightAlgorithm(ByVal faces() As PublicValues.Faces, ByRef corners() As Blocks.Corner,

ByRef edges() As Blocks.Edge, ByVal top As Char, ByVal front As Char, ByRef Instructions As String)

RotateFace(faces(5), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(4), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(4), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

End Sub

Public Sub BottomCrossAlgorithm(ByVal faces() As PublicValues.Faces, ByRef corners() As Blocks.Corner,

ByRef edges() As Blocks.Edge, ByVal top As Char, ByVal front As Char, ByRef Instructions As String)

RotateFace(faces(4), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(4), Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

End Sub

Public Sub BottomEdgeAlgorithmRight(ByVal faces() As PublicValues.Faces, ByRef corners() As Blocks.Corner,

ByRef edges() As Blocks.Edge, ByVal top As Char, ByVal front As Char, ByRef Instructions As String)

RotateFace(faces(3), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.HalfTurn, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Anticlockwise, corners, edges, top, front, Instructions)

End Sub

Public Sub BottomEdgeAlgorithmLeft(ByVal faces() As PublicValues.Faces, ByRef corners() As Blocks.Corner,

ByRef edges() As Blocks.Edge, ByVal top As Char, ByVal front As Char, ByRef Instructions As String)

RotateFace(faces(1), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(1), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(1), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.HalfTurn, corners, edges, top, front, Instructions)

RotateFace(faces(1), Direction.Clockwise, corners, edges, top, front, Instructions)

End Sub

Public Sub BottomAnticlockwiseCornerAlgorithm(ByVal faces() As PublicValues.Faces, ByRef corners() As Blocks.Corner,

ByRef edges() As Blocks.Edge, ByVal top As Char, ByVal front As Char, ByRef Instructions As String)

RotateFace(faces(3), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(1), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(1), Direction.Clockwise, corners, edges, top, front, Instructions)

'RotateFace(faces(0), Direction.Clockwise, \_corners, \_edges, \_top, \_front)

End Sub

Public Sub BottomClockwiseCornerAlgorithm(ByVal faces() As PublicValues.Faces, ByRef corners() As Blocks.Corner,

ByRef edges() As Blocks.Edge, ByVal top As Char, ByVal front As Char, ByRef Instructions As String)

RotateFace(faces(1), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(1), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Anticlockwise, corners, edges, top, front, Instructions)

'RotateFace(faces(0), Direction.Anticlockwise, \_corners, \_edges, \_top, \_front)

End Sub

End Module

# Matrices

Public Module Matrices

Public ReadOnly iVector As Vector3x1 = New Vector3x1(1, 0, 0)

Public ReadOnly jVector As Vector3x1 = New Vector3x1(0, 1, 0)

Public ReadOnly kVector As Vector3x1 = New Vector3x1(0, 0, 1)

Public ReadOnly i4Vector As Vector4x1 = New Vector4x1(1, 0, 0, 0)

Public ReadOnly j4Vector As Vector4x1 = New Vector4x1(0, 1, 0, 0)

Public ReadOnly k4Vector As Vector4x1 = New Vector4x1(0, 0, 1, 0)

Public ReadOnly l4Vector As Vector4x1 = New Vector4x1(0, 0, 0, 1)

'ToDo see if you can get the dot functions inside the vector classes

Public MustInherit Class Matrix

Private \_columns As Integer

Public Property NoOfColumns As Integer

Get

Return \_columns

End Get

Protected Set(ByVal value As Integer)

\_columns = value

End Set

End Property

Private \_rows As Integer

Public Property NoOfRows As Integer

Get

Return \_rows

End Get

Protected Set(ByVal value As Integer)

\_rows = value

End Set

End Property

End Class

Public MustInherit Class Vector

Private \_rows

Public Property NoOfRows

Get

Return \_rows

End Get

Protected Set(value)

\_rows = value

End Set

End Property

End Class

Public Class Matrix3x3

Inherits Matrix

Public Sub New()

Me.NoOfColumns = 3

Me.NoOfRows = 3

Me.Row1 = iVector

Me.Row2 = jVector

Me.Row3 = kVector

End Sub

Public Sub New(ByVal mat As Matrix3x3)

Me.Row1 = mat.Row1

Me.Row2 = mat.Row2

Me.Row3 = mat.Row3

End Sub

Public Sub New(ByVal row1 As Vector3x1, ByVal row2 As Vector3x1, ByVal row3 As Vector3x1)

Me.NoOfColumns = 3

Me.NoOfRows = 3

Me.Row1 = row1

Me.Row2 = row2

Me.Row3 = row3

End Sub

Public Shared Operator \*(ByVal mat1 As Matrix3x3, ByVal mat2 As Matrix3x3) As Matrix3x3

Return New Matrix3x3(New Vector3x1(Dot(mat1.Row1, mat2.Column1),

Dot(mat1.Row1, mat2.Column2),

Dot(mat1.Row1, mat2.Column3)),

New Vector3x1(Dot(mat1.Row2, mat2.Column1),

Dot(mat1.Row2, mat2.Column2),

Dot(mat1.Row2, mat2.Column3)),

New Vector3x1(Dot(mat1.Row3, mat2.Column1),

Dot(mat1.Row3, mat2.Column2),

Dot(mat1.Row3, mat2.Column3)))

End Operator

Public Shared Operator \*(ByVal mat As Matrix3x3, ByVal vec As Vector3x1) As Vector3x1

Return New Vector3x1(Dot(mat.Row1, vec), Dot(mat.Row2, vec), Dot(mat.Row3, vec))

End Operator

Private \_row1 As Vector3x1

Private \_row2 As Vector3x1

Private \_row3 As Vector3x1

Public Property Row1() As Vector3x1

Get

Return \_row1

End Get

Set(ByVal value As Vector3x1)

\_row1 = value

End Set

End Property

Public Property Row2() As Vector3x1

Get

Return \_row2

End Get

Set(ByVal value As Vector3x1)

\_row2 = value

End Set

End Property

Public Property Row3() As Vector3x1

Get

Return \_row3

End Get

Set(ByVal value As Vector3x1)

\_row3 = value

End Set

End Property

Public Property Column1() As Vector3x1

Get

Return New Vector3x1(\_row1.x, \_row2.x, \_row3.x)

End Get

Set(value As Vector3x1)

\_row1.x = value.x

\_row2.x = value.y

\_row3.x = value.z

End Set

End Property

Public Property Column2() As Vector3x1

Get

Return New Vector3x1(\_row1.y, \_row2.y, \_row3.y)

End Get

Set(value As Vector3x1)

\_row1.y = value.x

\_row2.y = value.y

\_row3.y = value.z

End Set

End Property

Public Property Column3() As Vector3x1

Get

Return New Vector3x1(\_row1.z, \_row2.z, \_row3.z)

End Get

Set(value As Vector3x1)

\_row1.z = value.x

\_row2.z = value.y

\_row3.z = value.z

End Set

End Property

End Class

Public Class Matrix4x4

Inherits Matrix

Public Sub New()

Me.NoOfColumns = 4

Me.NoOfRows = 4

Me.Row1 = i4Vector

Me.Row2 = j4Vector

Me.Row3 = k4Vector

Me.Row4 = l4Vector

End Sub

Public Sub New(ByVal mat As Matrix4x4)

Me.NoOfColumns = 4

Me.NoOfRows = 4

Me.Row1 = mat.Row1

Me.Row2 = mat.Row2

Me.Row3 = mat.Row3

Me.Row4 = mat.Row4

End Sub

Public Sub New(ByVal row1 As Vector4x1, ByVal row2 As Vector4x1, ByVal row3 As Vector4x1, ByVal row4 As Vector4x1)

Me.NoOfColumns = 4

Me.NoOfRows = 4

Me.Row1 = row1

Me.Row2 = row2

Me.Row3 = row3

Me.Row4 = row4

End Sub

Public Sub New(ByVal mat As OpenTK.Matrix4)

Me.NoOfColumns = 4

Me.NoOfRows = 4

Me.Row1 = New Vector4x1(mat.Column0)

Me.Row2 = New Vector4x1(mat.Column1)

Me.Row3 = New Vector4x1(mat.Column2)

Me.Row4 = New Vector4x1(mat.Column3)

End Sub

'Public Shared Operator \*(ByVal mat1 As Matrix3x3, ByVal mat2 As Matrix3x3) As Matrix3x3

' Return New Matrix3x3(New Vector3x1(Dot(mat1.Row1, mat2.Column1),

' Dot(mat1.Row1, mat2.Column2),

' Dot(mat1.Row1, mat2.Column3)),

' New Vector3x1(Dot(mat1.Row2, mat2.Column1),

' Dot(mat1.Row2, mat2.Column2),

' Dot(mat1.Row2, mat2.Column3)),

' New Vector3x1(Dot(mat1.Row3, mat2.Column1),

' Dot(mat1.Row3, mat2.Column2),

' Dot(mat1.Row3, mat2.Column3)))

'End Operator

Public Shared Operator \*(ByVal mat As Matrix4x4, ByVal vec As Vector4x1) As Vector4x1

Return New Vector4x1(Dot(mat.Row1, vec), Dot(mat.Row2, vec), Dot(mat.Row3, vec), Dot(mat.Row4, vec))

End Operator

Private \_row1 As Vector4x1

Private \_row2 As Vector4x1

Private \_row3 As Vector4x1

Private \_row4 As Vector4x1

Public Property Row1() As Vector4x1

Get

Return \_row1

End Get

Set(ByVal value As Vector4x1)

\_row1 = value

End Set

End Property

Public Property Row2() As Vector4x1

Get

Return \_row2

End Get

Set(ByVal value As Vector4x1)

\_row2 = value

End Set

End Property

Public Property Row3() As Vector4x1

Get

Return \_row3

End Get

Set(ByVal value As Vector4x1)

\_row3 = value

End Set

End Property

Public Property Row4() As Vector4x1

Get

Return \_row4

End Get

Set(ByVal value As Vector4x1)

\_row4 = value

End Set

End Property

'TODO

'Public Property Column1() As Vector3x1

' Get

' Return New Vector3x1(\_row1.x, \_row2.x, \_row3.x)

' End Get

' Set(value As Vector3x1)

' \_row1.x = value.x

' \_row2.x = value.y

' \_row3.x = value.z

' End Set

'End Property

'Public Property Column2() As Vector3x1

' Get

' Return New Vector3x1(\_row1.y, \_row2.y, \_row3.y)

' End Get

' Set(value As Vector3x1)

' \_row1.y = value.x

' \_row2.y = value.y

' \_row3.y = value.z

' End Set

'End Property

'Public Property Column3() As Vector3x1

' Get

' Return New Vector3x1(\_row1.z, \_row2.z, \_row3.z)

' End Get

' Set(value As Vector3x1)

' \_row1.z = value.x

' \_row2.z = value.y

' \_row3.z = value.z

' End Set

'End Property

End Class

Public Class Vector3x1

Inherits Vector

Public Sub New()

Me.NoOfRows = 3

Me.x = 0

Me.y = 0

Me.z = 0

End Sub

Public Sub New(ByVal x As Single, ByVal y As Single, ByVal z As Single)

Me.NoOfRows = 3

Me.x = x

Me.y = y

Me.z = z

End Sub

Public Sub New(ByVal vec As Vector3x1)

Me.NoOfRows = 3

Me.x = vec.x

Me.y = vec.y

Me.z = vec.z

End Sub

Public Sub New(ByVal vec As OpenTK.Vector3)

Me.NoOfRows = 3

Me.x = vec.X

Me.y = vec.Y

Me.z = vec.Z

End Sub

Public Shared Operator =(ByVal vec1 As Vector3x1, ByVal vec2 As Vector3x1)

If vec1.x = vec2.x And vec1.y = vec2.y And vec1.z = vec2.z Then

Return True

Else

Return False

End If

End Operator

Public Shared Operator <>(ByVal vec1 As Vector3x1, ByVal vec2 As Vector3x1)

If vec1.x <> vec2.x Or vec1.y <> vec2.y Or vec1.z <> vec2.z Then

Return True

Else

Return False

End If

End Operator

Private \_x As Single

Private \_y As Single

Private \_z As Single

Public Property x As Single

Get

Return \_x

End Get

Set(ByVal value As Single)

\_x = Math.Round(value, 10)

End Set

End Property

Public Property y As Single

Get

Return \_y

End Get

Set(ByVal value As Single)

\_y = Math.Round(value, 10)

End Set

End Property

Public Property z As Single

Get

Return \_z

End Get

Set(ByVal value As Single)

\_z = Math.Round(value, 10)

End Set

End Property

Public ReadOnly Property Column As Single()

Get

Return {x, y, z}

End Get

End Property

End Class

Public Class Vector4x1

Inherits Vector

Public Sub New()

Me.NoOfRows = 4

Me.x = 0

Me.y = 0

Me.z = 0

Me.w = 0

End Sub

Public Sub New(ByVal x As Single, ByVal y As Single, ByVal z As Single, ByVal w As Single)

Me.NoOfRows = 4

Me.x = x

Me.y = y

Me.z = z

Me.w = w

End Sub

Public Sub New(ByVal vec As Vector4x1)

Me.NoOfRows = 4

Me.x = vec.x

Me.y = vec.y

Me.z = vec.z

Me.w = vec.w

End Sub

Public Sub New(ByVal vec As OpenTK.Vector4)

Me.NoOfRows = 4

Me.x = vec.X

Me.y = vec.Y

Me.z = vec.Z

Me.w = vec.W

End Sub

Public Shared Operator =(ByVal vec1 As Vector4x1, ByVal vec2 As Vector4x1)

If vec1.x = vec2.x And vec1.y = vec2.y And vec1.z = vec2.z And vec1.w = vec2.w Then

Return True

Else

Return False

End If

End Operator

Public Shared Operator <>(ByVal vec1 As Vector4x1, ByVal vec2 As Vector4x1)

If vec1.x <> vec2.x Or vec1.y <> vec2.y Or vec1.z <> vec2.z Or vec1.w <> vec2.w Then

Return True

Else

Return False

End If

End Operator

Private \_x As Single

Private \_y As Single

Private \_z As Single

Private \_w As Single

Public Property x As Single

Get

Return \_x

End Get

Set(ByVal value As Single)

\_x = Math.Round(value, 10)

End Set

End Property

Public Property y As Single

Get

Return \_y

End Get

Set(ByVal value As Single)

\_y = Math.Round(value, 10)

End Set

End Property

Public Property z As Single

Get

Return \_z

End Get

Set(ByVal value As Single)

\_z = Math.Round(value, 10)

End Set

End Property

Public Property w As Single

Get

Return \_w

End Get

Set(ByVal value As Single)

\_w = Math.Round(value, 10)

End Set

End Property

Public ReadOnly Property Column As Single()

Get

Return {x, y, z, w}

End Get

End Property

End Class

Public Function Dot(ByVal vec1 As Vector3x1, ByVal vec2 As Vector3x1) As Single

Return (vec1.x \* vec2.x + vec1.y \* vec2.y + vec1.z \* vec2.z)

End Function

Public Function Dot(ByVal vec1 As Vector4x1, ByVal vec2 As Vector4x1) As Single

Return (vec1.x \* vec2.x + vec1.y \* vec2.y + vec1.z \* vec2.z + vec1.w \* vec2.w)

End Function

End Module

# Coordinate Pairs

Module CoordinatePairs

'rename sticker??

Class Coordinate

Public Property x As Integer

Public Property y As Integer

'Property Colour????

Public Sub New()

x = 0

y = 0

End Sub

End Class

Class CornerTriplet

Private \_corners() As Coordinate = {New Coordinate, New Coordinate, New Coordinate}

Public Property Corners(ByVal index As Integer, ByVal dimention As Char) As Integer

Get

If UCase(dimention) = "X" Then

Return \_corners(index).x

ElseIf UCase(dimention) = "Y" Then

Return \_corners(index).y

End If

'TODO add try catch to catch !X & !Y

End Get

Set(ByVal value As Integer)

If UCase(dimention) = "X" Then

\_corners(index).x = value

ElseIf UCase(dimention) = "Y" Then

\_corners(index).y = value

End If

End Set

End Property

Public Property Corners(ByVal index As Integer) As Coordinate

Get

Return \_corners(index)

End Get

Set(value As Coordinate)

\_corners(index) = value

End Set

End Property

End Class

Class EdgePair

Private \_edges() As Coordinate = {New Coordinate, New Coordinate}

Public Property Edges(ByVal index As Integer, ByVal dimention As Char) As Integer

Get

If UCase(dimention) = "X" Then

Return \_edges(index).x

ElseIf UCase(dimention) = "Y" Then

Return \_edges(index).y

End If

'TODO add try catch to catch !X & !Y

End Get

Set(ByVal value As Integer)

If UCase(dimention) = "X" Then

' TODO FIX THIS!!!!

'IS THIS FIXEDDD???

\_edges(index).x = value

ElseIf UCase(dimention) = "Y" Then

\_edges(index).y = value

End If

End Set

End Property

Public Property Edges(ByVal index As Integer) As Coordinate

Get

Return \_edges(index)

End Get

Set(value As Coordinate)

\_edges(index) = value

End Set

End Property

End Class

End Module

# Processing

Imports PublicValues = RubiksCubeSolver\_v2\_0.MyPublic.PublicVars

Imports PublicFuncs = RubiksCubeSolver\_v2\_0.MyPublic.PublicFunctions

Imports Faces = RubiksCubeSolver\_v2\_0.MyPublic.PublicVars.Faces

'TODO relook at middleedges

'Blueface2 and bluefacetoprotated does 9 middle edges

'bluefacewrongrotation does 6 middle edges

'whiteline does 7 middle edges

Module Checks

Public Function CheckComplete(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char) As Boolean

For face = 0 To 5

Dim faceBlocks(8) As Block

Dim faceColour As PublicValues.Faces

faceColour = face

faceBlocks = BlocksByColour(faceColour.ToString, corners, edges)

For i = 0 To 7

If Not OnFace(faceBlocks(i), faceColour, top, front) Or

Not CheckCubieRotation(faceBlocks(i), faceColour) Then

Console.WriteLine(faceColour.ToString & " face NOT complete!")

Return False

End If

Next

Next

Return True

End Function

Public Function CheckForAnyCompleteFace(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String) As Boolean

For face = 0 To 5

Dim faceBlocks(8) As Block

Dim faceColour As PublicValues.Faces

faceColour = face

faceBlocks = BlocksByColour(faceColour.ToString, corners, edges)

For i = 0 To 7

If Not OnFace(faceBlocks(i), faceColour, top, front) Or

Not CheckCubieRotation(faceBlocks(i), faceColour) Then

'Try next face

Exit For

Else

If i = 7 Then

RotateFaceToTop(faceColour, corners, edges, top, front, Instructions)

top = faceColour.ToString

Console.WriteLine(faceColour.ToString & " face complete!")

Return True

End If

End If

Next

Next

Console.WriteLine("No faces complete")

Return False

End Function

Public Function CheckForAnyCross(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String) As Boolean

For face = 0 To 5

Dim faceBlocks(8) As Block

Dim faceColour As PublicValues.Faces

faceColour = face

faceBlocks = BlocksByColour(faceColour.ToString, corners, edges)

For i = 0 To 7

If faceBlocks(i).GetType() = GetType(Edge) Then

If Not OnFace(faceBlocks(i), faceColour, top, front) Or

Not CheckCubieRotation(faceBlocks(i), faceColour) Then

'Try next face

'Console.WriteLine(faceColour.ToString & " cross not complete")

Exit For

Else

If i = 7 Then

RotateFaceToTop(faceColour, corners, edges, top, front, Instructions)

top = faceColour.ToString

Console.WriteLine(faceColour.ToString & " cross complete!")

Return True

End If

End If

End If

Next

Next

Console.WriteLine("No crosses complete")

Return False

End Function

Public Function CheckEdgesOfTopCross(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char) As Boolean

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim topEdges() As Edge = TopEdgesClockwise(corners, edges, top, front)

Dim start As Integer

For i = 1 To 4

If CheckCubieColours(topEdges(0), faceColours(i)) Then

start = i

End If

Next

For i = 0 To 3

'Console.WriteLine(topEdges(i).Name & " -> " & faceColours(((start + i - 1) Mod 4) + 1).ToString() & ": " & PublicFuncs.CheckCubieColours(topEdges(i), faceColours(((start + i - 1) Mod 4) + 1)).ToString())

If Not CheckCubieColours(topEdges(i), faceColours(((start + i - 1) Mod 4) + 1)) Then

Console.WriteLine("Edges of cross in wrong order")

Return False

End If

Next

Console.WriteLine("Edges of cross in right order")

Return True

End Function

Public Function CheckCornersOfTopFace(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char) As Boolean

'remarks: ONLY WORKS PROVIDED top HAS THE CORRECT CUBIES ALREADY

'TODo ^^ this could be a problem as this is called whithout checking corner location from solvecube

Dim topBlocks() As Block = TopBlocksClockwise(corners, edges, top, front)

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim start As Integer

For i = 1 To 4

If CheckCubieColours(topBlocks(1), faceColours(i)) Then

start = i

Exit For

End If

Next

For i = 0 To 7

If topBlocks(i).Position.y <> 1 Then

Return False

End If

If topBlocks(i).GetType() = GetType(Edge) Then

If Not CheckCubieColours(topBlocks(i), faceColours((start - 2 + (i + 1) / 2) Mod 4 + 1)) Then

Console.WriteLine("Edges of " & top & " face in wrong place")

Return False

End If

ElseIf topBlocks(i).GetType() = GetType(Corner) Then

'checks colours of corners - rotation already checked/not needed checked

Dim clockwiseEdge, anticlockwiseEdge As Edge

Dim clockwiseEdgeColour, anticlockwiseEdgeColour As Char

clockwiseEdge = topBlocks(i + 1)

anticlockwiseEdge = topBlocks((i + 7) Mod 8)

If clockwiseEdge.Colours(0) = top Then

clockwiseEdgeColour = clockwiseEdge.Colours(1)

Else

clockwiseEdgeColour = clockwiseEdge.Colours(0)

End If

If anticlockwiseEdge.Colours(0) = top Then

anticlockwiseEdgeColour = anticlockwiseEdge.Colours(1)

Else

anticlockwiseEdgeColour = anticlockwiseEdge.Colours(0)

End If

'Console.WriteLine(anticlockwiseEdge.Name & "-" & topBlocks(i).Name & "-" & clockwiseEdge.Name)

If Not CheckCubieColours(topBlocks(i), clockwiseEdgeColour) Or

Not CheckCubieColours(topBlocks(i), anticlockwiseEdgeColour) Then

Console.WriteLine("Edges of " & top & " face in wrong place")

Return False

End If

End If

Next

Console.WriteLine("Edges of " & top & " face in right place")

Return True

End Function

Public Function CheckTopFaceRotation(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char) As Boolean

'remarks - only works if top edges are correct

Dim topEdges() As Edge = TopEdgesClockwise(corners, edges, top, front, 1)

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

For i = 1 To 4

'Console.WriteLine(topEdges(i).Name & " " & faceColours(i).ToString())

If Not CheckCubieColours(topEdges(i), faceColours(i)) Then

Console.WriteLine("Top face in wrong orientation")

Return False

End If

Next

'Else

'Console.WriteLine("Top face in wrong orientation")

'Return False

'End If

Console.WriteLine("Top face in right orientation")

Return True

End Function

Public Function CheckMiddleRow(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char) As Boolean

Dim middleEdges() As Edge = MiddleEdgesClockwise(corners, edges, top, front)

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

If Not CheckCubieColours(middleEdges(0), faceColours(4)) Or

Not CheckCubieColours(middleEdges(0), faceColours(1)) Then

Console.WriteLine("Middle row not correct")

Return False

End If

For i = 1 To 3

If Not CheckCubieColours(middleEdges(i), faceColours(i)) Or

Not CheckCubieColours(middleEdges(i), faceColours(i + 1)) Then

Console.WriteLine("Middle row not correct")

Return False

End If

Next

Console.WriteLine("Middle row correct")

Return True

End Function

Public Function CheckBottomCross(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String) As Boolean

Dim bottom As PublicValues.Faces

bottom = PublicFuncs.ColourChar2FaceNumber(MyPublic.Opposite(top))

Moves.RotateFaceToTop(bottom, corners, edges, top, front, Instructions)

Console.WriteLine("CUBE FLIPPED UPSIDE DOWN: bottom -> top")

top = bottom.ToString()

Dim faceBlocks(8) As Block

faceBlocks = BlocksByColour(bottom.ToString, corners, edges)

For i = 0 To 7

If faceBlocks(i).GetType() = GetType(Edge) Then

If Not OnFace(faceBlocks(i), bottom, top, front) Or

Not CheckCubieRotation(faceBlocks(i), bottom) Then

Console.WriteLine(bottom.ToString & " cross not complete")

Return False

Else

If i = 7 Then

Console.WriteLine(bottom.ToString & " cross complete!")

Return True

End If

End If

End If

Next

Return False

End Function

'is this needed or could CheckSidesOfTopFace() do the same job?

Public Function CheckTopFace(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge,

top As Char, ByVal front As Char) As Boolean

Dim faceBlocks(8) As Block

Dim faceColour As PublicValues.Faces

faceColour = PublicFuncs.ColourChar2FaceNumber(top)

faceBlocks = BlocksByColour(faceColour.ToString, corners, edges)

For i = 0 To 7

If Not OnFace(faceBlocks(i), faceColour, top, front) Or

Not CheckCubieRotation(faceBlocks(i), faceColour) Then

Console.WriteLine(faceColour.ToString & " face not complete")

Return False

Else

If i = 7 Then

'RotateFaceToTop(faceColour, \_corners, \_edges, \_top, \_front)

'\_top = faceColour.ToString

Console.WriteLine(faceColour.ToString & " face complete!")

Return True

End If

End If

Next

Return False

End Function

Public Function TopCornerCorrect(ByVal cubie As Corner, ByVal topFace As Faces, ByVal corners() As Blocks.Corner, ByVal edges() As Blocks.Edge, ByVal top As Char, ByVal front As Char)

If Not OnFace(cubie, topFace, top, front) Then

Return False

ElseIf Not CheckCubieRotation(cubie, topFace) Then

Return False

Else

'Only edges will be correct

Dim topBlocks() As Block = TopBlocksClockwise(corners, edges, top, front)

Dim index As Integer

'For i = 0 To 7

' If topBlocks(i).Name = cubie.Name Then

' index = i

' End If

'Next

Select Case (cubie.Position.x + cubie.Position.z)

Case -2 'Back left corner

index = 2

Case 0

If cubie.Position.x = -1 Then ' Front left corner

index = 0

ElseIf cubie.Position.x = 1 Then ' Back right corner

index = 4

End If

Case 2 ' Front right corner

index = 6

End Select

Dim clockwiseEdge, anticlockwiseEdge As Edge

Dim clockwiseEdgeColour, anticlockwiseEdgeColour As Char

clockwiseEdge = topBlocks(index + 1)

anticlockwiseEdge = topBlocks((index + 7) Mod 8)

If clockwiseEdge.Colours(0) = top Then

clockwiseEdgeColour = clockwiseEdge.Colours(1)

Else

clockwiseEdgeColour = clockwiseEdge.Colours(0)

End If

If anticlockwiseEdge.Colours(0) = top Then

anticlockwiseEdgeColour = anticlockwiseEdge.Colours(1)

Else

anticlockwiseEdgeColour = anticlockwiseEdge.Colours(0)

End If

'Console.WriteLine(anticlockwiseEdge.Name & "-" & topBlocks(i).Name & "-" & clockwiseEdge.Name)

If Not CheckCubieColours(cubie, clockwiseEdgeColour) Or

Not CheckCubieColours(cubie, anticlockwiseEdgeColour) Then

Return False

Else

Return True

End If

End If

End Function

End Module

Module Stages

Public Sub DoTopFace(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

DoTopCross(corners, edges, top, front, Instructions)

DoTopCorners(corners, edges, top, front, Instructions)

End Sub

Private Sub DoTopCross(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim topBlocksWhenSolved() As Block = BlocksByColour(top, corners, edges)

Dim topEdgesWhenSolved(0 To 3) As Edge

Dim index As Integer

For Each block In topBlocksWhenSolved

If block.GetType = GetType(Edge) Then

topEdgesWhenSolved(index) = block

index += 1

End If

Next

Dim wrongCubies(0 To 3), rightCubies(0 To 3) As Edge

Dim topFace As Faces = PublicFuncs.ColourChar2FaceNumber(top)

'move to seperate sub? - probbaly, its used 3/4 times

Dim wrongIndex As Integer = 0

Dim rightIndex As Integer = 0

For Each cubie In topEdgesWhenSolved

If Not OnFace(cubie, topFace, top, front) Or Not CheckCubieRotation(cubie, topFace) Then

wrongCubies(wrongIndex) = cubie

wrongIndex += 1

Else

rightCubies(rightIndex) = cubie

rightIndex += 1

End If

Next

ReDim Preserve wrongCubies(0 To wrongIndex - 1)

Dim numberOnTopFace = rightIndex

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

If numberOnTopFace = 0 Then

MakeTopDash(rightCubies, rightIndex, wrongCubies, numberOnTopFace, faces, corners, edges, top, front, Instructions)

End If

If numberOnTopFace = 1 Then

MakeTopLine(rightCubies, rightIndex, wrongCubies, numberOnTopFace, topFace, faces, corners, edges, top, front, Instructions)

End If

If numberOnTopFace = 2 Then

MakeTopT(rightCubies, rightIndex, wrongCubies, numberOnTopFace, topEdgesWhenSolved, topFace, faces, faceColours, corners, edges, top, front, Instructions)

End If

If numberOnTopFace = 3 Then

MakeTopCross(rightCubies, rightIndex, wrongCubies, numberOnTopFace, topEdgesWhenSolved, topFace, faces, faceColours, corners, edges, top, front, Instructions)

'Console.WriteLine(PublicValues.Instructions)

End If

If numberOnTopFace = 4 Then

Dim topEdges() As Edge = TopEdgesClockwise(corners, edges, top, front)

Dim colourIndex As Integer

If topEdges(0).PrimaryFace = top Then

colourIndex = 1

Else

colourIndex = 0

End If

'wrong if statement

Dim colourOfFrontCubie As Char = topEdges(0).Colours(colourIndex)

For i = 1 To 4

If faceColours(i) = colourOfFrontCubie Then

index = i

Exit For

End If

Next

' if colour that's supposed to be clockwise is not the same as the clockwise cubie

If faceColours((index Mod 4) + 1) <> topEdges(1).Colours(colourIndex) Then

RotateFace(faces(1), Direction.HalfTurn, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.HalfTurn, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.HalfTurn, corners, edges, top, front, Instructions)

RotateFace(faces(1), Direction.HalfTurn, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.HalfTurn, corners, edges, top, front, Instructions)

ElseIf topEdges(0).Colours(colourIndex) = PublicFuncs.Opposite(topEdges(1).Colours(colourIndex)) Or

topEdges(1).Colours(colourIndex) = PublicFuncs.Opposite(topEdges(2).Colours(colourIndex)) Or

topEdges(2).Colours(colourIndex) = PublicFuncs.Opposite(topEdges(3).Colours(colourIndex)) Or

topEdges(3).Colours(colourIndex) = PublicFuncs.Opposite(topEdges(0).Colours(colourIndex)) Then

'else use same method as bottom cross to permute edges

DoBottomEdges(corners, edges, top, front, Instructions)

End If

End If

'Console.WriteLine(PublicValues.Instructions)

End Sub

Private Sub MakeTopDash(ByRef rightCubies() As Edge, ByRef rightIndex As Integer, ByRef wrongCubies() As Edge, ByRef numberOnTopFace As Integer, ByVal faces() As Faces, ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim cubie As Edge

For Each wrongCubie In wrongCubies

If wrongCubie.Position.y = 0 Then

cubie = wrongCubie

Dim faceToRotate As Faces = GetFaceToRotateForMakingTopCross(cubie, corners, edges, top, front)

Dim direction As Direction

Select Case faceToRotate

Case faces(1) ' left

direction = cubie.Position.z

Case faces(2) ' back

direction = -cubie.Position.x

Case faces(3) ' right

direction = -cubie.Position.z

Case faces(4) ' front

direction = cubie.Position.x

End Select

RotateFace(faceToRotate, direction, corners, edges, top, front, Instructions)

Exit For

End If

Next

If cubie Is Nothing Then

For Each wrongCubie In wrongCubies

If wrongCubie.PrimaryFace = top And wrongCubie.Rotation = PublicFuncs.Opposite(top) Or

wrongCubie.PrimaryFace <> top And wrongCubie.Rotation <> PublicFuncs.Opposite(top) Then

cubie = wrongCubie

Dim faceToRotate As Faces = GetFaceToRotateForMakingTopCross(cubie, corners, edges, top, front)

RotateFace(faceToRotate, Direction.HalfTurn, corners, edges, top, front, Instructions)

Exit For

End If

Next

End If

If cubie Is Nothing Then

cubie = wrongCubies(0)

Dim faceToRotate, secondFaceToRotate As Faces

Select Case cubie.Position.x

Case -1 ' left edge cubie

faceToRotate = faces(1)

secondFaceToRotate = faces(2)

Case 0

If cubie.Position.z = -1 Then ' back edge cubie

faceToRotate = faces(2)

secondFaceToRotate = faces(3)

Else ' front edge cubie

faceToRotate = faces(4)

secondFaceToRotate = faces(1)

End If

Case 1 ' right edge cubie

faceToRotate = faces(3)

secondFaceToRotate = faces(4)

End Select

RotateFace(faceToRotate, Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(secondFaceToRotate, Direction.Anticlockwise, corners, edges, top, front, Instructions)

End If

wrongCubies = RemoveBlockFromEdgeArray(cubie, wrongCubies)

rightCubies(rightIndex) = cubie

rightIndex += 1

numberOnTopFace += 1

End Sub

Private Sub MakeTopLine(ByRef rightCubies() As Edge, ByRef rightIndex As Integer, ByRef wrongCubies() As Edge, ByRef numberOnTopFace As Integer, ByVal topFace As Faces, ByVal faces() As Faces, ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

' pick opposite colour wrong cubie

' put in correct place (opposite) on top rowon top row

' remove cubie from wrong cubies list

' add 1 to NO

Dim colourOfTopCubie As Char

Dim topCubie As Edge = rightCubies(0)

Dim currentCubie As Edge

Dim colourIndex As Integer

If topCubie.PrimaryFace = top Then

colourIndex = 1

Else

colourIndex = 0

End If

colourOfTopCubie = topCubie.Colours(colourIndex)

For Each wrongCubie In wrongCubies

If wrongCubie.Colours(colourIndex) = PublicFuncs.Opposite(colourOfTopCubie) Then

currentCubie = wrongCubie

Exit For

End If

Next

If currentCubie Is Nothing Then

For Each rightCubie In rightCubies

If rightCubie.Colours(colourIndex) = PublicFuncs.Opposite(colourOfTopCubie) Then

currentCubie = rightCubie

Exit For

End If

Next

End If

Dim faceToRotate, secondFaceToRotate As Faces

Dim oneMove As Boolean

oneMove = True

Dim direction As Direction = Moves.Direction.NoChange

Dim cubieCorrectlyOnTopFace As Boolean = False

If currentCubie.Position.y = 0 Then

faceToRotate = GetFaceToRotateForMakingTopCross(currentCubie, corners, edges, top, front)

Select Case faceToRotate

Case faces(1) ' left

direction = currentCubie.Position.z

Case faces(2) ' back

direction = -currentCubie.Position.x

Case faces(3) ' right

direction = -currentCubie.Position.z

Case faces(4) ' front

direction = currentCubie.Position.x

End Select

ElseIf currentCubie.Position.y = -1 Then

If currentCubie.PrimaryFace = top And currentCubie.Rotation = PublicFuncs.Opposite(top) Or

currentCubie.PrimaryFace <> top And currentCubie.Rotation <> PublicFuncs.Opposite(top) Then

faceToRotate = GetFaceToRotateForMakingTopCross(currentCubie, corners, edges, top, front)

direction = Moves.Direction.HalfTurn

Else

oneMove = False

Select Case currentCubie.Position.x

Case -1 ' left edge cubie

faceToRotate = faces(1)

secondFaceToRotate = faces(2)

Case 0

If currentCubie.Position.z = -1 Then ' back edge cubie

faceToRotate = faces(2)

secondFaceToRotate = faces(3)

Else ' front edge cubie

faceToRotate = faces(4)

secondFaceToRotate = faces(1)

End If

Case 1 ' right edge cubie

faceToRotate = faces(3)

secondFaceToRotate = faces(4)

End Select

direction = Moves.Direction.Clockwise

End If

ElseIf currentCubie.Position.y = 1 Then

ReDim Preserve wrongCubies(0 To wrongCubies.Length)

wrongCubies(wrongCubies.Length - 1) = currentCubie

RemoveBlockFromEdgeArray(currentCubie, rightCubies)

faceToRotate = GetFaceToRotateForMakingTopCross(currentCubie, corners, edges, top, front)

If CheckCubieRotation(currentCubie, topFace) Then

'rotate top

Dim topDirection As Direction

Select Case faceToRotate

Case faces(1) ' left

Select Case topCubie.Position.x

Case 0

topDirection = topCubie.Position.z

Case 1 ' right edge cubie

topDirection = Moves.Direction.NoChange

End Select

Case faces(2) ' back

Select Case topCubie.Position.z

Case 0

topDirection = -topCubie.Position.x

Case 1 ' front edge cubie

topDirection = Moves.Direction.NoChange

End Select

Case faces(3) ' right

Select Case topCubie.Position.x

Case -1 ' left edge cubie

topDirection = Moves.Direction.NoChange

Case 0

topDirection = -topCubie.Position.z

End Select

Case faces(4) ' front

Select Case topCubie.Position.z

Case -1 ' back edge cubie

topDirection = Moves.Direction.NoChange

Case 0

topDirection = topCubie.Position.x

End Select

End Select

RotateFace(faceToRotate, Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(topFace, topDirection, corners, edges, top, front, Instructions)

RotateFace(faceToRotate, Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

cubieCorrectlyOnTopFace = True

Else

RotateFace(faceToRotate, Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

For Each wrongCubie In wrongCubies

If wrongCubie.Colours(colourIndex) = PublicFuncs.Opposite(colourOfTopCubie) Then

currentCubie = wrongCubie

Exit For

End If

Next

If currentCubie Is Nothing Then

For Each rightCubie In rightCubies

If rightCubie.Colours(colourIndex) = PublicFuncs.Opposite(colourOfTopCubie) Then

currentCubie = rightCubie

Exit For

End If

Next

End If

faceToRotate = GetFaceToRotateForMakingTopCross(currentCubie, corners, edges, top, front)

Select Case faceToRotate

Case faces(1) ' left

direction = currentCubie.Position.z

Case faces(2) ' back

direction = -currentCubie.Position.x

Case faces(3) ' right

direction = -currentCubie.Position.z

Case faces(4) ' front

direction = currentCubie.Position.x

End Select

End If

End If

If Not cubieCorrectlyOnTopFace Then

Dim topDirection As Direction

Select Case faceToRotate

Case faces(1) ' left

Select Case topCubie.Position.x

Case -1 ' left edge cubie

topDirection = Moves.Direction.HalfTurn

Case 0

topDirection = topCubie.Position.z

Case 1 ' right edge cubie

topDirection = Moves.Direction.NoChange

End Select

Case faces(2) ' back

Select Case topCubie.Position.z

Case -1 ' back edge cubie

topDirection = Moves.Direction.HalfTurn

Case 0

topDirection = topCubie.Position.x

Case 1 ' front edge cubie

topDirection = Moves.Direction.NoChange

End Select

Case faces(3) ' right

Select Case topCubie.Position.x

Case -1 ' left edge cubie

topDirection = Moves.Direction.NoChange

Case 0

topDirection = -topCubie.Position.z

Case 1 ' right edge cubie

topDirection = Moves.Direction.HalfTurn

End Select

Case faces(4) ' front

Select Case topCubie.Position.z

Case -1 ' back edge cubie

topDirection = Moves.Direction.NoChange

Case 0

topDirection = -topCubie.Position.x

Case 1 ' front edge cubie

topDirection = Moves.Direction.HalfTurn

End Select

End Select

RotateFace(topFace, topDirection, corners, edges, top, front, Instructions)

RotateFace(faceToRotate, direction, corners, edges, top, front, Instructions)

If Not oneMove Then

RotateFace(topFace, Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(secondFaceToRotate, Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

End If

End If

wrongCubies = RemoveBlockFromEdgeArray(currentCubie, wrongCubies)

rightCubies(rightIndex) = currentCubie

rightIndex += 1

numberOnTopFace += 1

End Sub

Private Sub MakeTopT(ByRef rightCubies() As Edge, ByRef rightIndex As Integer, ByRef wrongCubies() As Edge, ByRef numberOnTopFace As Integer, ByVal topEdgesWhenSolved() As Edge, ByVal topFace As Faces, ByVal faces() As Faces, ByVal faceColours() As Char, ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim colourIndex As Integer

If rightCubies(0).PrimaryFace = top Then

colourIndex = 1

Else

colourIndex = 0

End If

If (rightCubies(0).Position.x = rightCubies(1).Position.x Or rightCubies(0).Position.z = rightCubies(1).Position.z) And

rightCubies(0).Colours(colourIndex) = PublicFuncs.Opposite(rightCubies(1).Colours(colourIndex)) Then

PutNextWrongEdgeOnTopFace(rightCubies, wrongCubies, colourIndex, topEdgesWhenSolved, topFace, faces, faceColours, corners, edges, top, front, Instructions)

Else

'Line but wrong colours or L

MakeTopLine(rightCubies, rightIndex, wrongCubies, numberOnTopFace, topFace, faces, corners, edges, top, front, Instructions)

End If

Dim wrongIndex As Integer = 0

rightIndex = 0

ReDim wrongCubies(0 To 3)

For Each cubie In topEdgesWhenSolved

If Not OnFace(cubie, topFace, top, front) Or Not CheckCubieRotation(cubie, topFace) Then

wrongCubies(wrongIndex) = cubie

wrongIndex += 1

Else

rightCubies(rightIndex) = cubie

rightIndex += 1

End If

Next

ReDim Preserve wrongCubies(0 To wrongIndex - 1)

numberOnTopFace = rightIndex

If numberOnTopFace = 2 Then

MakeTopT(rightCubies, rightIndex, wrongCubies, numberOnTopFace, topEdgesWhenSolved, topFace, faces, faceColours, corners, edges, top, front, Instructions)

End If

End Sub

Private Sub MakeTopCross(ByRef rightCubies() As Edge, ByRef rightIndex As Integer, ByRef wrongCubies() As Edge, ByRef numberOnTopFace As Integer, ByVal topEdgesWhenSolved() As Edge, ByVal topFace As Faces, ByVal faces() As Faces, ByVal faceColours() As Char, ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim colourIndex As Integer

If rightCubies(0).PrimaryFace = top Then

colourIndex = 1

Else

colourIndex = 0

End If

PutNextWrongEdgeOnTopFace(rightCubies, wrongCubies, colourIndex, topEdgesWhenSolved, topFace, faces, faceColours, corners, edges, top, front, Instructions)

Dim wrongIndex As Integer = 0

rightIndex = 0

ReDim wrongCubies(0 To 3)

For Each cubie In topEdgesWhenSolved

If Not OnFace(cubie, topFace, top, front) Or Not CheckCubieRotation(cubie, topFace) Then

wrongCubies(wrongIndex) = cubie

wrongIndex += 1

Else

rightCubies(rightIndex) = cubie

rightIndex += 1

End If

Next

ReDim Preserve wrongCubies(0 To wrongIndex - 1)

numberOnTopFace = rightIndex

If numberOnTopFace = 3 Then

MakeTopCross(rightCubies, rightIndex, wrongCubies, numberOnTopFace, topEdgesWhenSolved, topFace, faces, faceColours, corners, edges, top, front, Instructions)

End If

End Sub

Private Sub PutNextWrongEdgeOnTopFace(ByRef rightCubies() As Edge, ByRef wrongCubies() As Edge, ByVal colourIndex As Integer, ByVal topEdgesWhenSolved() As Edge, ByVal topFace As Faces, ByVal faces() As Faces, ByVal faceColours() As Char, ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim currentCubie As Edge = wrongCubies(0)

Dim faceToRotate, secondFaceToRotate As Faces

Dim direction As Direction = Moves.Direction.NoChange

Dim oneMove As Boolean

oneMove = True

If currentCubie.Position.y = 0 Then

faceToRotate = GetFaceToRotateForMakingTopCross(currentCubie, corners, edges, top, front)

Select Case faceToRotate

Case faces(1) ' left

direction = currentCubie.Position.z

Case faces(2) ' back

direction = -currentCubie.Position.x

Case faces(3) ' right

direction = -currentCubie.Position.z

Case faces(4) ' front

direction = currentCubie.Position.x

End Select

ElseIf currentCubie.Position.y = -1 Then

If currentCubie.PrimaryFace = top And currentCubie.Rotation = PublicFuncs.Opposite(top) Or

currentCubie.PrimaryFace <> top And currentCubie.Rotation <> PublicFuncs.Opposite(top) Then

faceToRotate = GetFaceToRotateForMakingTopCross(currentCubie, corners, edges, top, front)

direction = Moves.Direction.HalfTurn

Else

oneMove = False

Select Case currentCubie.Position.x

Case -1 ' left edge cubie

faceToRotate = faces(1)

secondFaceToRotate = faces(2)

Case 0

If currentCubie.Position.z = -1 Then ' back edge cubie

faceToRotate = faces(2)

secondFaceToRotate = faces(3)

Else ' front edge cubie

faceToRotate = faces(4)

secondFaceToRotate = faces(1)

End If

Case 1 ' right edge cubie

faceToRotate = faces(3)

secondFaceToRotate = faces(4)

End Select

direction = Moves.Direction.Clockwise

End If

ElseIf currentCubie.Position.y = 1 Then

ReDim Preserve wrongCubies(0 To wrongCubies.Length)

wrongCubies(wrongCubies.Length - 1) = currentCubie

RemoveBlockFromEdgeArray(currentCubie, rightCubies)

faceToRotate = GetFaceToRotateForMakingTopCross(currentCubie, corners, edges, top, front)

'will always be wrongly rotated as only wrongly rotated cubies on top face will be a

RotateFace(faceToRotate, Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

For Each wrongCubie In wrongCubies

If wrongCubie.Colours(colourIndex) = currentCubie.Colours(colourIndex) Then

currentCubie = wrongCubie

Exit For

End If

Next

'If currentCubie Is Nothing Then

' For Each rightCubie In rightCubies

' If rightCubie.Colours(colourIndex) = currentCubie.Colours(colourIndex) Then

' currentCubie = rightCubie

' Exit For

' End If

' Next

'End If

faceToRotate = GetFaceToRotateForMakingTopCross(currentCubie, corners, edges, top, front)

Select Case faceToRotate

Case faces(1) ' left

direction = currentCubie.Position.z

Case faces(2) ' back

direction = -currentCubie.Position.x

Case faces(3) ' right

direction = -currentCubie.Position.z

Case faces(4) ' front

direction = currentCubie.Position.x

End Select

End If

'||||...... up to here||||

'get colour of currentcubiie

'find the topcubie with the correct colour + 1

'orientate top face

Dim currentCubieColour As Char = currentCubie.Colours(colourIndex)

Dim clockwiseCubieColour As Char

Dim clockwiseCubie As Edge

For i = 1 To 4

If faceColours(i) = currentCubieColour Then

clockwiseCubieColour = faceColours((i Mod 4) + 1)

Exit For

End If

Next

For Each rightCubie In rightCubies

If rightCubie.Colours(colourIndex) = clockwiseCubieColour Then

clockwiseCubie = rightCubie

Exit For

End If

Next

Dim topDirection As Direction

Select Case faceToRotate

Case faces(1) ' left

Select Case clockwiseCubie.Position.z

Case -1 ' back edge cubie

topDirection = Moves.Direction.NoChange

Case 0

topDirection = clockwiseCubie.Position.x

Case 1 ' front edge cubie

topDirection = Moves.Direction.HalfTurn

End Select

Case faces(2) ' back

Select Case clockwiseCubie.Position.x

Case -1 ' left edge cubie

topDirection = Moves.Direction.HalfTurn

Case 0

topDirection = clockwiseCubie.Position.z

Case 1 ' right edge cubie

topDirection = Moves.Direction.NoChange

End Select

Case faces(3) ' right

Select Case clockwiseCubie.Position.z

Case -1 ' back edge cubie

topDirection = Moves.Direction.HalfTurn

Case 0

topDirection = -clockwiseCubie.Position.x

Case 1 ' front edge cubie

topDirection = Moves.Direction.NoChange

End Select

Case faces(4) ' front

Select Case clockwiseCubie.Position.x

Case -1 ' left edge cubie

topDirection = Moves.Direction.NoChange

Case 0

topDirection = -clockwiseCubie.Position.z

Case 1 ' right edge cubie

topDirection = Moves.Direction.HalfTurn

End Select

End Select

RotateFace(topFace, topDirection, corners, edges, top, front, Instructions)

RotateFace(faceToRotate, direction, corners, edges, top, front, Instructions)

If Not oneMove Then

RotateFace(topFace, Moves.Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(secondFaceToRotate, Moves.Direction.Anticlockwise, corners, edges, top, front, Instructions)

End If

End Sub

Public Sub DoTopCorners(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim topBlocksWhenSolved() As Block = BlocksByColour(top, corners, edges)

Dim topCornersWhenSolved(0 To 3) As Corner

Dim index As Integer

For Each block In topBlocksWhenSolved

If block.GetType = GetType(Corner) Then

topCornersWhenSolved(index) = block

index += 1

End If

Next

Dim wrongCubies(0 To 3), rightCubies(0 To 3) As Corner

Dim topFace As Faces = PublicFuncs.ColourChar2FaceNumber(top)

'move to seperate sub? - probbaly,

Dim wrongIndex As Integer = 0

Dim rightIndex As Integer = 0

For Each topCubie In topCornersWhenSolved

If Not TopCornerCorrect(topCubie, topFace, corners, edges, top, front) Then

wrongCubies(wrongIndex) = topCubie

wrongIndex += 1

Else

rightCubies(rightIndex) = topCubie

rightIndex += 1

End If

Next

ReDim Preserve wrongCubies(0 To wrongIndex - 1)

Dim numberOnTopFace = rightIndex

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

While numberOnTopFace < 4

PutNextCornerOnTopFace(rightCubies, wrongCubies, corners, edges, top, front, Instructions)

numberOnTopFace += 1

End While

End Sub

Private Sub PutNextCornerOnTopFace(ByRef rightCubies() As Corner, ByRef wrongCubies() As Corner, ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim cubie As Corner

cubie = wrongCubies(0)

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

If cubie.Position.y = -1 Then

Dim colourIndex As Integer

For i = 0 To 2

If cubie.Colours(i) = top Then

colourIndex = i

Exit For

End If

Next

Dim faceCubieRotatedTowards As Faces

Dim anticlockwiseFace, clockwiseFace As Faces

Select Case (cubie.Position.x + cubie.Position.z)

Case -2 'Back left corner

anticlockwiseFace = faces(1)

clockwiseFace = faces(2)

Case 0

If cubie.Position.x = -1 Then ' Front left corner

anticlockwiseFace = faces(4)

clockwiseFace = faces(1)

ElseIf cubie.Position.x = 1 Then ' Back right corner

anticlockwiseFace = faces(2)

clockwiseFace = faces(3)

End If

Case 2 ' Front right corner

anticlockwiseFace = faces(3)

clockwiseFace = faces(4)

End Select

Select Case colourIndex

Case 0

faceCubieRotatedTowards = PublicFuncs.ColourChar2FaceNumber(cubie.Rotation)

Case 1

faceCubieRotatedTowards = PublicFuncs.ColourChar2FaceNumber(cubie.SecondaryRotation)

Case 2

Dim facesAroundCorner() As Faces = {clockwiseFace, anticlockwiseFace, faces(5)}

For Each face In facesAroundCorner

If face.ToString() <> cubie.Rotation And face.ToString() <> cubie.SecondaryRotation Then

faceCubieRotatedTowards = face

Exit For

End If

Next

End Select

If faceCubieRotatedTowards = clockwiseFace Then

'if left put put underneath where it should be (bottom left):

RotateFaceToFront(anticlockwiseFace, corners, edges, top, front, Instructions)

faceColours = PublicFuncs.AssociatedFaces(top, front)

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

RotateTopFaceForCorner(cubie, faces, corners, edges, top, front, Instructions)

RotateFace(faces(1), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(1), Direction.Anticlockwise, corners, edges, top, front, Instructions)

ElseIf faceCubieRotatedTowards = anticlockwiseFace Then

'if right put underneath where it should be (bottom right):

RotateFaceToFront(clockwiseFace, corners, edges, top, front, Instructions)

faceColours = PublicFuncs.AssociatedFaces(top, front)

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

RotateTopFaceForCorner(cubie, faces, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Clockwise, corners, edges, top, front, Instructions)

Else

'if bottom put underneath corner (bottom right):

RotateFaceToFront(clockwiseFace, corners, edges, top, front, Instructions)

faceColours = PublicFuncs.AssociatedFaces(top, front)

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

RotateTopFaceForCorner(cubie, faces, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(4), Direction.Clockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Direction.HalfTurn, corners, edges, top, front, Instructions)

RotateFace(faces(4), Direction.Anticlockwise, corners, edges, top, front, Instructions)

End If

'remove from wrong corners

'add to rightcorners

wrongCubies = RemoveBlockFromCornerArray(cubie, wrongCubies)

For i = 0 To 3

If rightCubies(i) Is Nothing Then

rightCubies(i) = cubie

Exit For

End If

Next

ElseIf cubie.Position.y = 1 Then

Dim newFront As Faces

Select Case (cubie.Position.x + cubie.Position.z)

Case -2 'Back left corner

newFront = faces(2)

Case 0

If cubie.Position.x = -1 Then ' Front left corner

newFront = faces(1)

ElseIf cubie.Position.x = 1 Then ' Back right corner

newFront = faces(3)

End If

Case 2 ' Front right corner

newFront = faces(4)

End Select

RotateFaceToFront(newFront, corners, edges, top, front, Instructions)

faceColours = PublicFuncs.AssociatedFaces(top, front)

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

RotateFace(faces(3), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(5), Direction.Anticlockwise, corners, edges, top, front, Instructions)

RotateFace(faces(3), Direction.Clockwise, corners, edges, top, front, Instructions)

PutNextCornerOnTopFace(rightCubies, wrongCubies, corners, edges, top, front, Instructions)

End If

End Sub

Private Sub RotateTopFaceForCorner(ByVal corner As Corner, ByVal faces() As Faces, ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

'rotates top face so that the correct corner of the top face is above the corner passed as an argument

Dim topEdges() As Edge = TopEdgesClockwise(corners, edges, top, front)

Dim cornerColours(1) As Char

Dim colourIndex As Integer

If topEdges(0).Colours(0) = top Then

colourIndex = 1

Else

colourIndex = 0

End If

Dim cornerLocation As Integer

For i = 0 To 3

cornerColours(0) = topEdges(i).Colours(colourIndex)

cornerColours(1) = topEdges((i + 1) Mod 4).Colours(colourIndex)

Dim colourIndexOne, colourIndexTwo As Integer

If corner.Colours(0) = top Then

colourIndexOne = 1

colourIndexTwo = 2

ElseIf corner.Colours(1) = top Then

colourIndexOne = 0

colourIndexTwo = 2

ElseIf corner.Colours(2) = top Then

colourIndexOne = 0

colourIndexTwo = 1

End If

'if the corner cubie colour matches the colours of the edge pair

If corner.Colours(colourIndexOne) = cornerColours(0) And corner.Colours(colourIndexTwo) = cornerColours(1) Or

corner.Colours(colourIndexOne) = cornerColours(1) And corner.Colours(colourIndexTwo) = cornerColours(0) Then

cornerLocation = i

Exit For

End If

Next

Dim cubiePosition As Integer

Select Case (corner.Position.x + corner.Position.z)

Case -2 'Back left corner

cubiePosition = 0

Case 0

If corner.Position.x = -1 Then ' Front left corner

cubiePosition = 3

ElseIf corner.Position.x = 1 Then ' Back right corner

cubiePosition = 1

End If

Case 2 ' Front right corner

cubiePosition = 2

End Select

Dim rotationDifference As Integer = cornerLocation – cubiePosition

If Math.Abs(rotationDifference) <> 3 And Math.Abs(rotationDifference) <> 2 Then

RotateFace(faces(0), rotationDifference, corners, edges, top, front, Instructions)

ElseIf Math.Abs(rotationDifference) = 2 Then

RotateFace(faces(0), Direction.HalfTurn, corners, edges, top, front, Instructions)

ElseIf rotationDifference = -3 Then

RotateFace(faces(0), Direction.Anticlockwise, corners, edges, top, front, Instructions)

ElseIf rotationDifference = 3 Then

RotateFace(faces(0), Direction.Clockwise, corners, edges, top, front, Instructions)

End If

End Sub

Public Sub OrientateTopFace(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim topFace As PublicValues.Faces

topFace = PublicFuncs.ColourChar2FaceNumber(top)

Dim topBlocks() As Block = BlocksByColour(top, corners, edges)

Dim topLeftEdge As New Edge

For Each cubie In topBlocks

If cubie.GetType() = GetType(Edge) And cubie.Position.x = -1 Then

topLeftEdge = cubie

Exit For

End If

Next

Dim direction As Direction

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

'Dim instructionString As String = ""

For i = 1 To 4

If CheckCubieColours(topLeftEdge, faceColours(i)) Then

Select Case i

Case 1 : direction = Direction.NoChange ': instructionString = ""

Case 2 : direction = Direction.Clockwise ': instructionString = "U "

Case 3 : direction = Direction.HalfTurn ': instructionString = "U2 "

Case 4 : direction = Direction.Anticlockwise ': instructionString = "U' "

End Select

End If

Next

RotateFace(topFace, direction, corners, edges, top, front, Instructions)

End Sub

Public Sub DoMiddleRow(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim bottom As Char = PublicFuncs.Opposite(top)

Dim bottomFace As PublicValues.Faces =

PublicFuncs.ColourChar2FaceNumber(bottom)

Dim MiddleEdgesWhenSolved(0 To 3) As Edge

Dim index As Integer = 0

For Each cubie In edges

If Not CheckCubieColours(cubie, top) And

Not CheckCubieColours(cubie, bottom) Then

MiddleEdgesWhenSolved(index) = cubie

index += 1

End If

Next

For Each cubie In MiddleEdgesWhenSolved

If cubie.Position.y = -1 Then

Dim face As PublicValues.Faces

Dim otherFace As Char

If cubie.Rotation = bottom Then

face = PublicFuncs.ColourChar2FaceNumber(cubie.Colours(1))

otherFace = cubie.Colours(0)

Else

face = PublicFuncs.ColourChar2FaceNumber(cubie.Colours(0))

otherFace = cubie.Colours(1)

End If

RotateFaceToFront(face, corners, edges, top, front, Instructions)

Dim direction As Direction

Select Case cubie.Position.x

Case -1 ' left edge cubie

direction = Moves.Direction.Clockwise

Case 0

If cubie.Position.z = -1 Then ' back edge cubie

direction = Moves.Direction.HalfTurn

Else ' front edge cubie

direction = Moves.Direction.NoChange

End If

Case 1 ' right edge cubie

direction = Moves.Direction.Anticlockwise

End Select

RotateFace(bottomFace, direction, corners, edges, top, front, Instructions)

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

'Write function to do middle algorithm

If otherFace = faceColours(1) Then

'Insert cubie into left

Algorithms.MiddleLeftAlgorithm(faces, corners, edges, top, front, Instructions)

ElseIf otherFace = faceColours(3) Then

'Insert cubie into right

Algorithms.MiddleRightAlgorithm(faces, corners, edges, top, front, Instructions)

Else

Console.WriteLine("Error, middle edge cubie is not in correct place to insert into middle row")

End If

Else

'TEST

Dim faceToLeft, faceToRight As Char

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

'why isnt this used?

'Dim direction As Moves.Direction

Select Case (cubie.Position.x + cubie.Position.z)

Case -2 'Back left corner

faceToLeft = faceColours(2)

faceToRight = faceColours(1)

Case 0

If cubie.Position.x = -1 Then ' Front left corner

faceToLeft = faceColours(1)

faceToRight = faceColours(4)

ElseIf cubie.Position.x = 1 Then ' Back right corner

faceToLeft = faceColours(3)

faceToRight = faceColours(2)

End If

Case 2 ' Front right corner

faceToLeft = faceColours(4)

faceToRight = faceColours(3)

End Select

If cubie.PrimaryFace = faceToLeft And cubie.Rotation <> faceToLeft Or

cubie.PrimaryFace = faceToRight And cubie.Rotation <> faceToRight Or

cubie.PrimaryFace <> faceToLeft And cubie.PrimaryFace <> faceToRight Or

Not (CheckCubieColours(cubie, faceToLeft) And CheckCubieColours(cubie, faceToRight)) Then

Dim face As PublicValues.Faces = PublicFuncs.ColourChar2FaceNumber(faceToLeft)

RotateFaceToFront(face, corners, edges, top, front, Instructions)

faceColours = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

'Insert cubie into right

Algorithms.MiddleRightAlgorithm(faces, corners, edges, top, front, Instructions)

' Recurse??? WILL THIS WORK????? TODO CHECK

DoMiddleRow(corners, edges, top, front, Instructions)

Exit For

End If

End If

Console.WriteLine("Next Middle Edge")

Next

End Sub

Public Sub DoBottomCross(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim bottomFace As PublicValues.Faces =

PublicFuncs.ColourChar2FaceNumber(top)

Dim bottomCrossCorrectRotated(0 To 3) As Edge

Dim index As Integer = 0

For Each cubie In edges

If CheckCubieColours(cubie, top) Then

If CheckCubieRotation(cubie, bottomFace) Then

bottomCrossCorrectRotated(index) = cubie

index += 1

End If

End If

Next

ReDim Preserve bottomCrossCorrectRotated(0 To index - 1)

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

Select Case bottomCrossCorrectRotated.Length()

Case 0, 1 'middle only

Algorithms.BottomCrossAlgorithm(faces, corners, edges, top, front, Instructions)

DoBottomCross(corners, edges, top, front, Instructions)

Case 2

'corner or middle line

Dim bottomEdgesOrdered(0 To 3) As Edge

For i = 0 To bottomCrossCorrectRotated.Length - 1

Select Case bottomCrossCorrectRotated(i).Position.x

Case -1 ' left edge cubie

index = 0

Case 0

If bottomCrossCorrectRotated(i).Position.z = -1 Then ' back edge cubie

index = 1

Else ' front edge cubie

index = 3

End If

Case 1 ' right edge cubie

index = 2

End Select

bottomEdgesOrdered(index) = bottomCrossCorrectRotated(i)

Next

For i = 0 To bottomEdgesOrdered.Length - 1

If bottomEdgesOrdered(i) IsNot Nothing Then

If bottomEdgesOrdered((i + 2) Mod 4) IsNot Nothing Then ' Bottom face in line configuration

If i = 1 Or i = 3 Then

RotateFace(bottomFace, Direction.Clockwise, corners, edges, top, front, Instructions)

End If

Algorithms.BottomCrossAlgorithm(faces, corners, edges, top, front, Instructions)

Exit For

Else ' Bottom face in L config.

Select Case i

Case 0

If bottomEdgesOrdered(1) Is Nothing Then

RotateFace(bottomFace, Direction.Clockwise, corners, edges, top, front, Instructions)

End If

Case 1 ' top right

RotateFace(bottomFace, Direction.Anticlockwise, corners, edges, top, front, Instructions)

Case 2 ' bottom right

RotateFace(bottomFace, Direction.HalfTurn, corners, edges, top, front, Instructions)

Case 3 ' bottom left

RotateFace(bottomFace, Direction.Clockwise, corners, edges, top, front, Instructions)

End Select

Algorithms.BottomCrossAlgorithm(faces, corners, edges, top, front, Instructions)

Algorithms.BottomCrossAlgorithm(faces, corners, edges, top, front, Instructions)

Exit For

End If

End If

Next

'Case Else

' Console.WriteLine("Hopefully the bottom cross is solved")

End Select

End Sub

Public Sub DoBottomEdges(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim bottomEdges() As Edge = TopEdgesClockwise(corners, edges, top, front)

Dim bottomEdgeColours(0 To 3) As Char

For i = 0 To 3

For colour = 0 To 1

If bottomEdges(i).Colours(colour) <> top Then

bottomEdgeColours(i) = bottomEdges(i).Colours(colour)

End If

Next

Next

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

If bottomEdgeColours(0) = PublicFuncs.Opposite(bottomEdgeColours(2)) Then

'opposites are correct

Console.WriteLine("LINE")

Algorithms.BottomEdgeAlgorithmRight(faces, corners, edges, top, front, Instructions)

RotateFace(faces(0), Direction.Anticlockwise, corners, edges, top, front, Instructions)

Algorithms.BottomEdgeAlgorithmRight(faces, corners, edges, top, front, Instructions)

Else

Dim edgesCorrect As Boolean = False

For i = 0 To 3

If bottomEdgeColours(i) <> PublicFuncs.Opposite(bottomEdgeColours((i + 1) Mod 4)) Then

For face = 1 To 4

If bottomEdgeColours(i) = faceColours(face) Then

If bottomEdgeColours((i + 1) Mod 4) = faceColours(((face Mod 4) + 1)) Then

Select Case i

Case 0

RotateFace(faces(0), Direction.Clockwise, corners, edges, top, front, Instructions)

Case 2

RotateFace(faces(0), Direction.Anticlockwise, corners, edges, top, front, Instructions)

Case 3

RotateFace(faces(0), Direction.HalfTurn, corners, edges, top, front, Instructions)

End Select

Algorithms.BottomEdgeAlgorithmRight(faces, corners, edges, top, front, Instructions)

edgesCorrect = True

Exit For

End If

End If

Next face

If edgesCorrect Then

Exit For

End If

End If

Next i

End If

End Sub

Public Sub PositionBottomCorners(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

Dim bottomBlocks() As Block = TopBlocksClockwise(corners, edges, top, front)

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

For i = 0 To 6 Step 2

Dim clockwiseEdge As Edge = bottomBlocks(i + 1)

Dim antiClockwiseEdge As Edge = bottomBlocks((i + 7) Mod 8)

Dim clockwiseEdgeColour, anticlockwiseEdgeColour As Char

If clockwiseEdge.Colours(0) = top Then

clockwiseEdgeColour = clockwiseEdge.Colours(1)

Else

clockwiseEdgeColour = clockwiseEdge.Colours(0)

End If

If antiClockwiseEdge.Colours(0) = top Then

anticlockwiseEdgeColour = antiClockwiseEdge.Colours(1)

Else

anticlockwiseEdgeColour = antiClockwiseEdge.Colours(0)

End If

'Console.WriteLine(anticlockwiseEdge.Name & "-" & topBlocks(i).Name & "-" & clockwiseEdge.Name)

If CheckCubieColours(bottomBlocks(i), clockwiseEdgeColour) And

CheckCubieColours(bottomBlocks(i), anticlockwiseEdgeColour) Then

' topBlocks(i) is in the right place

' work out whether we need to go clockwise or anticlockwise

Dim nextCorner As Integer = (i + 2) Mod 8

Dim clockwiseEdgeForNextCorner As Edge = bottomBlocks(nextCorner + 1)

Dim twiceClockwiseEdgeForNextCorner As Edge = bottomBlocks((nextCorner + 3) Mod 8)

Dim clockwiseEdgeColourForNextCorner, twiceClockwiseEdgeColourForNextCorner As Char

If clockwiseEdgeForNextCorner.Colours(0) = top Then

clockwiseEdgeColourForNextCorner = clockwiseEdgeForNextCorner.Colours(1)

Else

clockwiseEdgeColourForNextCorner = clockwiseEdgeForNextCorner.Colours(0)

End If

If twiceClockwiseEdgeForNextCorner.Colours(0) = top Then

twiceClockwiseEdgeColourForNextCorner = twiceClockwiseEdgeForNextCorner.Colours(1)

Else

twiceClockwiseEdgeColourForNextCorner = twiceClockwiseEdgeForNextCorner.Colours(0)

End If

Dim bottomFace As PublicValues.Faces = PublicFuncs.ColourChar2FaceNumber(top)

If CheckCubieColours(bottomBlocks(nextCorner), clockwiseEdgeColourForNextCorner) And

CheckCubieColours(bottomBlocks(nextCorner), twiceClockwiseEdgeColourForNextCorner) Then

'corners need shifted clockwise

Select Case i

Case 0 : RotateFace(bottomFace, Direction.Anticlockwise, corners, edges, top, front, Instructions)

Case 2 : RotateFace(bottomFace, Direction.HalfTurn, corners, edges, top, front, Instructions)

Case 4 : RotateFace(bottomFace, Direction.Clockwise, corners, edges, top, front, Instructions)

End Select

Algorithms.BottomClockwiseCornerAlgorithm(faces, corners, edges, top, front, Instructions)

Else

'corners need shifted anticlockwise

Select Case i

Case 2 : RotateFace(bottomFace, Direction.Anticlockwise, corners, edges, top, front, Instructions)

Case 4 : RotateFace(bottomFace, Direction.HalfTurn, corners, edges, top, front, Instructions)

Case 6 : RotateFace(bottomFace, Direction.Clockwise, corners, edges, top, front, Instructions)

End Select

Algorithms.BottomAnticlockwiseCornerAlgorithm(faces, corners, edges, top, front, Instructions)

End If

Exit For

ElseIf i = 6 Then 'none in right place

Algorithms.BottomClockwiseCornerAlgorithm(faces, corners, edges, top, front, Instructions)

PositionBottomCorners(corners, edges, top, front, Instructions)

Exit For

End If

Next

End Sub

Public Sub OrientateBottomCorners(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge, ByRef top As Char, ByRef front As Char, ByRef Instructions As String)

OrientateTopFace(corners, edges, top, front, Instructions)

' TODO: This is bad maybe rewrite at some point

' maybe write function to check whether it needs to be rotated clockwise or anticlockwise instead of horrible if statemtns

Dim bottomCorners() As Corner = TopCornersClockwise(corners, edges, top, front)

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

'Dim originalFaces() As Faces = faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

Dim numberOfWrongCorners As Integer = 0

For Each cubie In bottomCorners

If Not CheckCubieRotation(cubie, faces(0)) Then

numberOfWrongCorners += 1

End If

Next

For i = 0 To 3

Dim originalBottom As PublicValues.Faces

Dim cornerOne As Corner = bottomCorners(i)

faceColours = PublicFuncs.AssociatedFaces(top, front)

For j = 0 To 5

faces(j) = PublicFuncs.ColourChar2FaceNumber(faceColours(j))

Next

If Not CheckCubieRotation(cornerOne, faces(0)) Then

Dim cornerTwo As Corner = bottomCorners((i + 1) Mod 4)

Dim faceOne As PublicValues.Faces = faces(0)

originalBottom = faceOne

Dim faceTwo As PublicValues.Faces = faces(i + 1)

Dim pairOfCorners As Boolean = True

If cornerOne.Rotation = cornerTwo.Rotation And

cornerOne.PrimaryFace = cornerTwo.PrimaryFace Then

'If cornerOne.PrimaryFace = cornerTwo.PrimaryFace Then

'rotate other face colour to top, primaryface to left

If faceOne = PublicFuncs.ColourChar2FaceNumber(cornerOne.PrimaryFace) Then

RotateFaceToTop(faceTwo, corners, edges, top, front, Instructions)

RotateFaceToLeft(faceOne, corners, edges, top, front, Instructions)

Else

RotateFaceToTop(faceOne, corners, edges, top, front, Instructions)

RotateFaceToLeft(faceTwo, corners, edges, top, front, Instructions)

End If

ElseIf cornerOne.SecondaryRotation = cornerTwo.SecondaryRotation And

cornerOne.SecondaryFace = cornerTwo.SecondaryFace Then

'rotate other face colour to top, secondaryFace to left

If faceOne = PublicFuncs.ColourChar2FaceNumber(cornerOne.SecondaryFace) Then

RotateFaceToTop(faceTwo, corners, edges, top, front, Instructions)

RotateFaceToLeft(faceOne, corners, edges, top, front, Instructions)

Else

RotateFaceToTop(faceOne, corners, edges, top, front, Instructions)

RotateFaceToLeft(faceTwo, corners, edges, top, front, Instructions)

End If

ElseIf cornerOne.Rotation = cornerTwo.Rotation Or

cornerOne.SecondaryRotation = cornerTwo.SecondaryRotation Then

'whichever of the faces is = primary or secondary face

'then rotate that one to the top and the other left?

'maybe?

If faceOne = PublicFuncs.ColourChar2FaceNumber(cornerOne.PrimaryFace) Or

faceOne = PublicFuncs.ColourChar2FaceNumber(cornerOne.SecondaryFace) Or

faceOne = PublicFuncs.ColourChar2FaceNumber(cornerTwo.PrimaryFace) Or

faceOne = PublicFuncs.ColourChar2FaceNumber(cornerTwo.SecondaryFace) Then

RotateFaceToTop(faceOne, corners, edges, top, front, Instructions)

RotateFaceToLeft(faceTwo, corners, edges, top, front, Instructions)

Else

RotateFaceToTop(faceTwo, corners, edges, top, front, Instructions)

RotateFaceToLeft(faceOne, corners, edges, top, front, Instructions)

End If

ElseIf numberOfWrongCorners = 3 Then

'1 is correctly orientated

Console.WriteLine("Fish")

Dim clockwise As Boolean = False

If cornerOne.PrimaryFace = faceOne.ToString() Then

If cornerOne.Rotation = faceTwo.ToString() Then

clockwise = True

End If

ElseIf cornerOne.SecondaryFace = faceOne.ToString() Then

If cornerOne.SecondaryRotation = faceTwo.ToString() Then

clockwise = True

End If

ElseIf cornerOne.PrimaryFace = faceTwo.ToString() Then

If cornerOne.Rotation <> faceOne.ToString() Then

clockwise = True

End If

ElseIf cornerOne.SecondaryFace = faceTwo.ToString() Then

If cornerOne.SecondaryRotation <> faceOne.ToString() Then

clockwise = True

End If

Else

'TODO THIS DOESNT WORK - maybe not needed now?

If cornerOne.PrimaryFace <> faceOne.ToString() And cornerOne.SecondaryFace <> faceOne.ToString() Then

clockwise = True

End If

End If

If clockwise Then 'if cubie (i.e. all 3) needs rotated clockwise: rotate edge to right of fish

If CheckCubieRotation(bottomCorners((i + 3) Mod 4), faceOne) Then

faceOne = faces(i + 1)

faceTwo = faces(0)

ElseIf CheckCubieRotation(bottomCorners((i + 2) Mod 4), faceOne) Then

faceOne = faces((i + 3) Mod 4 + 1)

faceTwo = faces(0)

ElseIf CheckCubieRotation(bottomCorners((i + 1) Mod 4), faceOne) Then

faceOne = faces((i + 2) Mod 4 + 1)

faceTwo = faces(0)

End If

Else 'if cubie (i.e. all 3) needs rotated anticlockwise: rotate edge below fish

If CheckCubieRotation(bottomCorners((i + 2) Mod 4), faceOne) Then

faceOne = faces(i + 1)

faceTwo = faces(0)

ElseIf CheckCubieRotation(bottomCorners((i + 3) Mod 4), faceOne) Then

faceOne = faces((i + 1) Mod 4 + 1)

faceTwo = faces(0)

ElseIf CheckCubieRotation(bottomCorners((i + 1) Mod 4), faceOne) Then

'check this

faceOne = faces((i + 3) Mod 4 + 1)

faceTwo = faces(0)

End If

End If

' |Y|Y| | <-

' |Y|Y|Y| <-Right of fish

' | |Y| | <-

' /\ /\

' || ||

' below fish

RotateFaceToTop(faceOne, corners, edges, top, front, Instructions)

RotateFaceToLeft(faceTwo, corners, edges, top, front, Instructions)

faceColours = PublicFuncs.AssociatedFaces(top, front)

For j = 0 To 5

faces(j) = PublicFuncs.ColourChar2FaceNumber(faceColours(j))

Next

Algorithms.BottomEdgeAlgorithmRight(faces, corners, edges, top, front, Instructions)

Algorithms.BottomEdgeAlgorithmLeft(faces, corners, edges, top, front, Instructions)

RotateFaceToTop(originalBottom, corners, edges, top, front, Instructions)

OrientateBottomCorners(corners, edges, top, front, Instructions)

Exit For

ElseIf Not CheckCubieRotation(bottomCorners((i + 2) Mod 4), faceOne) Then

If CheckCubieRotation(bottomCorners((i + 1) Mod 4), faceOne) And

CheckCubieRotation(bottomCorners((i + 3) Mod 4), faceOne) Then

'current cubie is wronglyrotated

'find which face is == top

'rotate to it on top and otherface on left :)

Dim newTop As PublicValues.Faces

If cornerOne.PrimaryFace = faceOne.ToString() Then

newTop = PublicFuncs.ColourChar2FaceNumber(cornerOne.Rotation)

ElseIf cornerOne.SecondaryFace = faceOne.ToString Then

newTop = PublicFuncs.ColourChar2FaceNumber(cornerOne.SecondaryRotation)

Else

Dim facesAroundCorner() As PublicValues.Faces = {faces(0), faces(i + 1), faces(i + 2)}

For Each face In facesAroundCorner

If face.ToString() <> cornerOne.Rotation And

face.ToString() <> cornerOne.SecondaryRotation Then

newTop = face

End If

Next

If newTop = faceOne Then

faceOne = faceTwo

ElseIf newTop = Nothing Then

MsgBox("Error - cannot find face of corner cubie that matches the bottom face")

newTop = faceOne

End If

End If

RotateFaceToTop(newTop, corners, edges, top, front, Instructions)

RotateFaceToLeft(faceOne, corners, edges, top, front, Instructions)

'NO: what Is it's 2 opposite corners only --> what do about this

' rotating any random pair appears to give 1 correct orientation

' but you want to flip it when the correct face is on top

' i.e. when solving red, you want a red cubie face on top and

' the red face on the left :: doesnt matter whetehr it's top or bottom

End If

Else

'they dont need dealt with cos they dont make a pair maybe?

'BUT ALGO STILL BEING DONE ON NON-PAIRS

pairOfCorners = False

End If

faceColours = PublicFuncs.AssociatedFaces(top, front)

For j = 0 To 5

faces(j) = PublicFuncs.ColourChar2FaceNumber(faceColours(j))

Next

If pairOfCorners Then

Algorithms.BottomEdgeAlgorithmRight(faces, corners, edges, top, front, Instructions)

Algorithms.BottomEdgeAlgorithmLeft(faces, corners, edges, top, front, Instructions)

If top <> originalBottom.ToString() Then

RotateFaceToTop(originalBottom, corners, edges, top, front, Instructions)

End If

'is this a good idea??

If Not CheckComplete(corners, edges, top, front) Then

OrientateBottomCorners(corners, edges, top, front, Instructions)

Return

Else

Return

End If

End If

End If

Next

End Sub

Public Sub OutputInstructions(ByVal corners() As Blocks.Corner, ByVal edges() As Blocks.Edge, ByRef Instructions As String)

Console.WriteLine("CUBE SOLVED!!! ^-^")

PublicFuncs.WriteCornersToFile(corners, "FINAL-CORNERS.txt")

PublicFuncs.WriteEdgesToFile(edges, "FINAL-EDGES.txt")

FileOpen(2, "INSTRUCTIONS.txt", OpenMode.Output)

Print(2, Instructions)

FileClose(2)

End Sub

End Module

Module Functions

Public Function RemoveBlockFromEdgeArray(ByVal cubie As Block, ByRef array() As Edge) As Edge()

Dim newArray(0 To array.Length - 2) As Edge

Dim index As Integer

For i = 0 To array.Length - 1

Try

If array(i).Name <> cubie.Name Then

newArray(index) = array(i)

index += 1

End If

Catch ex As NullReferenceException

End Try

Next

Return newArray

End Function

Public Function RemoveBlockFromCornerArray(ByVal cubie As Block, ByRef array() As Corner) As Corner()

Dim newArray(0 To array.Length - 2) As Corner

Dim index As Integer

For i = 0 To array.Length - 1

Try

If array(i).Name <> cubie.Name Then

newArray(index) = array(i)

index += 1

End If

Catch ex As NullReferenceException

End Try

Next

Return newArray

End Function

''' <returns>

''' index 0 = front left corner

''' </returns>

Public Function TopBlocksClockwise(ByVal corners As Corner(), ByVal edges As Edge(), ByVal top As Char, ByVal front As Char) As Block()

Dim topBlocks() As Block

Dim topBlocksCopy(0 To 7) As Block

topBlocks = BlocksByColour(top, corners, edges)

Array.Copy(topBlocks, topBlocksCopy, 8)

Dim index As Integer = 0

For i = 0 To 7

Dim cubie As Block = topBlocksCopy(i)

If cubie.GetType() = GetType(Edge) Then

Select Case cubie.Position.x

Case -1 ' left edge cubie

index = 1

Case 0

If cubie.Position.z = -1 Then ' back edge cubie

index = 3

Else ' front edge cubie

index = 7

End If

Case 1 ' right edge cubie

index = 5

End Select

ElseIf cubie.GetType() = GetType(Corner) Then

Select Case (cubie.Position.x + cubie.Position.z)

Case -2 'Back left corner

index = 2

Case 0

If cubie.Position.x = -1 Then ' Front left corner

index = 0

ElseIf cubie.Position.x = 1 Then ' Back right corner

index = 4

End If

Case 2 ' Front right corner

index = 6

End Select

End If

topBlocks(index) = cubie

Next

Return topBlocks

End Function

Public Function TopEdgesClockwise(ByVal corners As Corner(), ByVal edges As Edge(), ByVal top As Char, ByVal front As Char, Optional startIndex As Integer = 0) As Edge()

Dim topBlocks(0 To 7) As Block

topBlocks = BlocksByColour(top, corners, edges)

Dim topEdges(0 To startIndex + 3) As Edge

Dim index As Integer

For i = 0 To 7

If topBlocks(i).GetType() = GetType(Edge) Then

Select Case topBlocks(i).Position.x

Case -1 ' left edge cubie

index = 0

Case 0

If topBlocks(i).Position.z = -1 Then ' back edge cubie

index = 1

Else ' front edge cubie

index = 3

End If

Case 1 ' right edge cubie

index = 2

End Select

topEdges(startIndex + index) = topBlocks(i)

End If

Next

Return topEdges

End Function

Public Function TopCornersClockwise(ByVal corners As Corner(), ByVal edges As Edge(), ByVal top As Char, ByVal front As Char) As Corner()

Dim topBlocks(0 To 7) As Block

topBlocks = BlocksByColour(top, corners, edges)

Dim topCorners(0 To 3) As Corner

Dim index As Integer

For Each cubie In topBlocks

If cubie.GetType() = GetType(Corner) Then

Select Case (cubie.Position.x + cubie.Position.z)

Case -2 'Back left corner

index = 1

Case 0

If cubie.Position.x = -1 Then ' Front left corner

index = 0

ElseIf cubie.Position.x = 1 Then ' Back right corner

index = 2

End If

Case 2 ' Front right corner

index = 3

End Select

topCorners(index) = cubie

End If

Next

Return topCorners

End Function

Public Function MiddleEdgesClockwise(ByVal corners As Corner(), ByVal edges As Edge(), ByVal top As Char, ByVal front As Char, Optional startIndex As Integer = 0) As Edge()

Dim middleEdges(0 To startIndex + 3) As Edge

Dim index As Integer

For Each cubie In edges

If cubie.Position.y = 0 Then

Select Case (cubie.Position.x + cubie.Position.z)

Case -2 'Back left edge

index = 1

Case 0

If cubie.Position.x = -1 Then ' Front left edge

index = 0

ElseIf cubie.Position.x = 1 Then ' Back right edge

index = 2

End If

Case 2 ' Front right edge

index = 3

End Select

middleEdges(startIndex + index) = cubie

End If

Next

Return middleEdges

End Function

Public Function GetFaceToRotateForMakingTopCross(ByVal cubie As Edge, ByVal corners As Corner(), ByVal edges As Edge(), ByVal top As Char, ByVal front As Char) As Faces

Dim faceToRotate As Faces

Dim faceColours() As Char = PublicFuncs.AssociatedFaces(top, front)

Dim faces(0 To 5) As PublicValues.Faces

For i = 0 To 5

faces(i) = PublicFuncs.ColourChar2FaceNumber(faceColours(i))

Next

If cubie.Position.y = 0 Then

If cubie.PrimaryFace <> top Then

faceToRotate = PublicFuncs.ColourChar2FaceNumber(cubie.Rotation)

Else

Dim faceOne, faceTwo As Faces

Select Case (cubie.Position.x + cubie.Position.z)

Case -2 'Back left edge

faceOne = faces(1)

faceTwo = faces(2)

Case 0

If cubie.Position.x = -1 Then ' Front left edge

faceOne = faces(4)

faceTwo = faces(1)

ElseIf cubie.Position.x = 1 Then ' Back right edge

faceOne = faces(2)

faceTwo = faces(3)

End If

Case 2 ' Front right edge

faceOne = faces(3)

faceTwo = faces(4)

End Select

If faceOne.ToString = cubie.Rotation Then

faceToRotate = faceTwo

Else

faceToRotate = faceOne

End If

End If

ElseIf cubie.Position.y = -1 Or cubie.Position.y = 1 Then

Select Case cubie.Position.x

Case -1 ' left edge cubie

faceToRotate = faces(1)

Case 0

If cubie.Position.z = -1 Then ' back edge cubie

faceToRotate = faces(2)

Else ' front edge cubie

faceToRotate = faces(4)

End If

Case 1 ' right edge cubie

faceToRotate = faces(3)

End Select

End If

Return faceToRotate

End Function

''' <summary>

''' Returns an array of cubies with the correct colour stickers

''' </summary>

Public Function BlocksByColour(ByVal colour As Char, ByVal corners As Corner(), ByVal edges As Edge()) As Block()

Dim cubies(0 To 7) As Block

Dim count As Integer = 0

'Console.WriteLine(vbNewLine & colour)

For Each corner In corners

If CheckCubieColours(corner, colour) Then

cubies(count) = corner

'Console.WriteLine(corner.Name & " " & corner.Position(0) & corner.Position(1) & corner.Position(2))

count += 1

End If

Next

For Each edge In edges

If CheckCubieColours(edge, colour) Then

cubies(count) = edge

'Console.WriteLine(edge.Name & " " & edge.Position(0) & edge.Position(1) & edge.Position(2))

count += 1

End If

Next

Return cubies

End Function

''' <summary>

''' Checks if a cubie has a particular colour sticker

''' </summary>

Public Function CheckCubieColours(ByVal cubie As Block, ByVal colour As Char)

If cubie.GetType() = GetType(Edge) Then

For col = 0 To 1

If cubie.Colours(col) = colour Then

Return True

End If

Next

ElseIf cubie.GetType() = GetType(Corner) Then

For col = 0 To 2

If cubie.Colours(col) = colour Then

Return True

End If

Next

End If

Return False

End Function

''' <summary>

''' Analyses whether a particular cubie is on a particular face of the cube

''' </summary>

''' <param name="cubie">The cubie to be checked</param>

Public Function OnFace(ByVal cubie As Block, ByVal face As Faces, ByVal top As Char, ByVal front As Char) As Boolean

Dim faces As Char()

faces = PublicFuncs.AssociatedFaces(top, front)

' 0 = Top Face

' 5 = bottom face

' 1 = left face

' 2 = back face

' 3 = right face

' 4 = front face

If face.ToString = faces(0) And cubie.Position.y = 1 Or

face.ToString = faces(5) And cubie.Position.y = -1 Or

face.ToString = faces(4) And cubie.Position.z = 1 Or

face.ToString = faces(2) And cubie.Position.z = -1 Or

face.ToString = faces(1) And cubie.Position.x = -1 Or

face.ToString = faces(3) And cubie.Position.x = 1 Then

Return True

Else

Return False

End If

End Function

''' <summary>

''' Checks if a cubie is rotated correctly onto the given face.

''' Given that it is already on the face.

''' I.e. that the blue side of a blue cubie is on the blue face of the cube

''' </summary>

Public Function CheckCubieRotation(ByVal cubie As Block, ByVal faceColour As Faces)

If cubie.GetType() = GetType(Edge) Then

'Console.WriteLine(cubie.Name & " " & cubie.Position(0) & cubie.Position(1) & cubie.Position(2) & " is edge")

Select Case faceColour

Case Faces.W, Faces.Y

If cubie.Rotation <> faceColour.ToString Then

Return False

End If

Case Faces.R, Faces.O

If faceColour.ToString() = cubie.PrimaryFace Then

If cubie.Rotation <> faceColour.ToString Then

Return False

End If

Else

If cubie.Rotation = faceColour.ToString Then

Return False

End If

End If

Case Faces.B, Faces.G

If cubie.Rotation = faceColour.ToString Then

Return False

End If

End Select

ElseIf cubie.GetType() = GetType(Corner) Then

Dim tempCorner As Corner = cubie

Select Case faceColour

Case Faces.W, Faces.Y

If tempCorner.Rotation <> faceColour.ToString Then

Return False

End If

Case Faces.R, Faces.O

If tempCorner.Rotation = faceColour.ToString Or tempCorner.SecondaryRotation <> faceColour.ToString Then

Return False

End If

Case Faces.B, Faces.G

If tempCorner.Rotation = faceColour.ToString Or tempCorner.SecondaryRotation = faceColour.ToString Then

Return False

End If

End Select

End If

Return True

End Function

Public Function OptimiseInstructions(ByVal instructionString As String) As String()

Dim instructions(0) As String

Dim subString As String = ""

Dim index As Integer = 0

For i = 0 To Len(instructionString) - 1

If instructionString(i) = " " Then

If subString <> "" Then

ReDim Preserve instructions(instructions.Length)

instructions(index) = subString

index += 1

subString = ""

End If

Else

subString += instructionString(i)

End If

Next

Dim currentInstruction, nextInstruction As String

For i = 0 To instructions.Length - 2

currentInstruction = instructions(i)

nextInstruction = instructions(i + 1)

If currentInstruction <> "" And nextInstruction <> "" Then

If Left(currentInstruction, 1) = Left(nextInstruction, 1) Then

If Right(currentInstruction, 1) = "'" Then

If Right(nextInstruction, 1) = "2" Then

'anticlockwise then 180 - > clockwise

instructions(i) = ""

instructions(i + 1) = Left(currentInstruction, 1)

ElseIf Right(nextInstruction, 1) = "'" Then

'180

instructions(i) = ""

instructions(i + 1) = Left(currentInstruction, 1) + "2"

Else

'turn anticlockwise then clockwise

instructions(i) = ""

instructions(i + 1) = ""

End If

ElseIf Right(currentInstruction, 1) = "2" Then

If Right(nextInstruction, 1) = "2" Then

'full 360

instructions(i) = ""

instructions(i + 1) = ""

ElseIf Right(nextInstruction, 1) = "'" Then

'180 then back -> clockwise

instructions(i) = ""

instructions(i + 1) = Left(currentInstruction, 1)

Else

'180 then forward -> anticclockwise

instructions(i) = ""

instructions(i + 1) = Left(currentInstruction, 1) + "'"

End If

Else

If Right(nextInstruction, 1) = "'" Then

instructions(i) = ""

instructions(i + 1) = ""

ElseIf Right(nextInstruction, 1) = "2" Then

'clockwise then 180 - > anticlockwise

instructions(i) = ""

instructions(i + 1) = Left(currentInstruction, 1) + "'"

ElseIf Right(nextInstruction, 1) = "'" Then

'turn clockwise then anticlockwise

instructions(i) = ""

instructions(i + 1) = ""

Else

'180

instructions(i) = ""

instructions(i + 1) = Left(currentInstruction, 1) + "2"

End If

End If

End If

End If

Next

instructions = RemoveBlanksFromArray(instructions)

Return instructions

End Function

Private Function RemoveBlanksFromArray(ByVal array() As String) As String()

Dim newArray(0 To array.Length - 1) As String

Dim index As Integer = 0

For i = 0 To array.Length - 1

If array(i) <> "" And array(i) <> Nothing Then

newArray(index) = array(i)

index += 1

End If

Next

ReDim Preserve newArray(0 To index - 1)

Return newArray

End Function

End Module

Public Class Processing

'FANCY NAMES:

'4. Create yellow cross == Edge Orientation (EO)

'5. Place cross in correct position == Edge Permutation (EP)

'6. Position yellow corners at correct positions == Edge Permutation (EP)

'7. Orient all the yellow corners == Corner Orientation (CO)

Private \_corners As Corner()

Private \_edges As Edge()

Public Sub New(ByRef corners As Corner(), ByRef edges As Edge())

' This call is required by the designer.

InitializeComponent()

\_corners = corners

\_edges = edges

End Sub

Private Sub Processing\_Shown(sender As Object, e As EventArgs) Handles MyBase.Shown

lblText.Visible = True

lblText.Refresh()

SolveCube(\_corners, \_edges)

End Sub

Private Function copyCornerArray(ByVal corners() As Corner) As Corner()

Dim cornerCopy(0 To corners.Length - 1) As Corner

For i = 0 To corners.Length - 1

cornerCopy(i) = New Corner

cornerCopy(i).Name = corners(i).Name

cornerCopy(i).PrimaryFace = corners(i).PrimaryFace

cornerCopy(i).Rotation = corners(i).Rotation

cornerCopy(i).SecondaryFace = corners(i).SecondaryFace

cornerCopy(i).SecondaryRotation = corners(i).SecondaryRotation

cornerCopy(i).Position = corners(i).Position

cornerCopy(i).Colours = corners(i).Colours

Next

Return cornerCopy

End Function

Private Function copyEdgeArray(ByVal edges() As Edge) As Edge()

Dim edgeCopy(0 To edges.Length - 1) As Edge

For i = 0 To edges.Length - 1

edgeCopy(i) = New Edge

edgeCopy(i).Name = edges(i).Name

edgeCopy(i).PrimaryFace = edges(i).PrimaryFace

edgeCopy(i).Rotation = edges(i).Rotation

edgeCopy(i).Position = edges(i).Position

edgeCopy(i).Colours = edges(i).Colours

Next

Return edgeCopy

End Function

Private Sub SolveCube(ByRef corners() As Blocks.Corner, ByRef edges() As Blocks.Edge)

Dim Instructions As String = ""

'todo check is top and front of scrambled are W and R

Dim scrambledCorners() As Corner = copyCornerArray(corners)

Dim scrambledEdges() As Edge = copyEdgeArray(edges)

Dim top As Char = "W"

Dim front As Char = "R"

lblStatus.Text = "Solving Top Face"

lblStatus.Refresh()

If CheckForAnyCompleteFace(corners, edges, top, front, Instructions) Then

If CheckEdgesOfTopCross(corners, edges, top, front) Then

If Not CheckCornersOfTopFace(corners, edges, top, front) Then

DoTopCorners(corners, edges, top, front, Instructions)

End If

Else

'PermuteTopEdges(corners, edges, top, front, Instructions)

DoBottomEdges(corners, edges, top, front, Instructions)

DoTopFace(corners, edges, top, front, Instructions)

End If

Else

If CheckForAnyCross(corners, edges, top, front, Instructions) Then

If CheckEdgesOfTopCross(corners, edges, top, front) Then

If Not CheckCornersOfTopFace(corners, edges, top, front) Then

DoTopCorners(corners, edges, top, front, Instructions)

End If

Else

'PermuteTopEdges(corners, edges, top, front, Instructions)

DoBottomEdges(corners, edges, top, front, Instructions)

DoTopFace(corners, edges, top, front, Instructions)

End If

Else

DoTopFace(corners, edges, top, front, Instructions)

End If

End If

'top face done::

If Not CheckTopFaceRotation(corners, edges, top, front) Then

OrientateTopFace(corners, edges, top, front, Instructions)

End If

lblStatus.Text = "Solving Middle Row"

lblStatus.Refresh()

If Not CheckMiddleRow(corners, edges, top, front) Then

DoMiddleRow(corners, edges, top, front, Instructions)

End If

lblStatus.Text = "Solving Bottom Cross"

lblStatus.Refresh()

If Not CheckBottomCross(corners, edges, top, front, Instructions) Then

DoBottomCross(corners, edges, top, front, Instructions)

End If

If Not CheckEdgesOfTopCross(corners, edges, top, front) Then

DoBottomEdges(corners, edges, top, front, Instructions)

End If

lblStatus.Text = "Solving Bottom Face"

lblStatus.Refresh()

If Not CheckCornersOfTopFace(corners, edges, top, front) Then

PositionBottomCorners(corners, edges, top, front, Instructions)

End If

If Not CheckTopFace(corners, edges, top, front) Then

OrientateBottomCorners(corners, edges, top, front, Instructions)

End If

Console.WriteLine("Should now be complete")

If CheckComplete(corners, edges, top, front) Then

lblStatus.Text = "Solved"

lblStatus.Refresh()

Dim InstructionsList() As String

InstructionsList = OptimiseInstructions(Instructions)

OutputInstructions(corners, edges, Instructions)

Dim outputForm As New \_3DOutput(scrambledCorners, scrambledEdges, InstructionsList)

outputForm.Show()

Me.Close()

Else

Console.WriteLine("CUBE NOT SOVLED :(")

lblStatus.Text = "Not solved"

lblStatus.Refresh()

End If

End Sub

End Class

# Output Classes

Imports OpenTK

Imports OpenTK.Graphics.OpenGL

Imports System.Drawing

Imports System.IO

Module OutputClasses

Structure Vertex3D

Public Shared ReadOnly SizeInBytes As Integer = (Vector3.SizeInBytes + Vector4.SizeInBytes + Vector3.SizeInBytes)

'Public Shared ReadOnly SizeInBytes As Integer = (Vector3.SizeInBytes + Vector4.SizeInBytes)

Public position As Vector3

Public colour As Vector4

Public normal As Vector3

Public Property Color() As Color

Get

Return Drawing.Color.FromArgb(CInt(colour.W \* 255), CInt(colour.X \* 255), CInt(colour.Y \* 255), CInt(colour.Z \* 255))

End Get

Set(ByVal value As Color)

Me.colour = New Vector4(value.R / 255, value.G / 255, value.B / 255, value.A / 255)

End Set

End Property

Public Sub New(ByVal position As Vector3, ByVal colour As Vector4, ByVal normal As Vector3)

Me.position = position

Me.colour = colour

Me.normal = normal

End Sub

Public Sub New(ByVal position As Vector3, ByVal colour As Color, ByVal normal As Vector3)

Me.position = position

'Converts colour(ARGB) to vector4(R, G, B, A)

Me.colour = New Vector4(colour.R / 255, colour.G / 255, colour.B / 255, colour.A / 255)

Me.normal = normal

End Sub

Public Sub New(ByVal position As Vector3, ByVal colour As Vector4)

Me.position = position

Me.colour = colour

Me.normal = New Vector3(0, 0, 0)

End Sub

Public Sub New(ByVal position As Vector3, ByVal colour As Color)

Me.position = position

'Converts colour(ARGB) to vector4(R, G, B, A)

Me.colour = New Vector4(colour.R / 255, colour.G / 255, colour.B / 255, colour.A / 255)

Me.normal = New Vector3(0, 0, 0)

End Sub

End Structure

Public Class MyShaderProgram

Dim program As Integer

Sub New(ByVal vertexPath As String, ByVal fragmentPath As String)

'Dim infolog As String

Dim versionNo As Integer = CInt(CStr(GL.GetString(StringName.Version)(0) + GL.GetString(StringName.Version)(2)))

Dim versionString As String = "330"

'DOENST WORK ON 4.4.0

'If versionNo >= 44 Then

' versionString = "330"

'versionString = CStr(versionNo \* 10)

'hmmm....

'#400 didnt work on 4.0.0

'Else

If versionNo >= 33 Then

versionString = "330"

ElseIf versionNo >= 21 Then

versionString = "120"

Else

MsgBox("3D output is not supported by your graphics library")

program = -1

versionString = "-1"

Return

End If

Dim vertexShader As Integer

vertexShader = GL.CreateShader(ShaderType.VertexShader)

GL.ShaderSource(vertexShader, "#version " + versionString + File.ReadAllText(vertexPath))

GL.CompileShader(vertexShader)

Console.WriteLine("Vertex Shader Success:" & GL.GetShaderInfoLog(vertexShader).ToString())

Dim fragmentShader As Integer

fragmentShader = GL.CreateShader(ShaderType.FragmentShader)

GL.ShaderSource(fragmentShader, "#version " + versionString + File.ReadAllText(fragmentPath))

GL.CompileShader(fragmentShader)

Console.WriteLine("Fragment Shader Success:" & GL.GetShaderInfoLog(fragmentShader).ToString())

program = GL.CreateProgram()

GL.AttachShader(program, vertexShader)

GL.AttachShader(program, fragmentShader)

GL.LinkProgram(program)

Console.WriteLine("Shader Program Success:" & GL.GetProgramInfoLog(program).ToString())

GL.DeleteShader(vertexShader)

GL.DeleteShader(fragmentShader)

End Sub

Public Sub Use()

GL.UseProgram(program)

End Sub

Public ReadOnly Property Handle() As Integer

Get

Return program

End Get

End Property

End Class

Public Structure PosAndType

Public position As Vector3

Public type As Char

Public Sub New(ByVal pos As Vector3, ByVal type As Char)

position = pos

Me.type = type

End Sub

End Structure

Public Class TkBlock

'3d position vector

Private \_position As Vector3 = New Vector3(0, 0, 0)

Public Property Position() As Vector3

Get

Return \_position

End Get

Set(ByVal value As Vector3)

\_position = New Vector3(value)

End Set

End Property

Private \_type As Char

''' <summary>

''' C = corner, E = edge, M = middle

''' </summary>

''' <returns>

''' BLock type

''' </returns>

Public Property Type() As Char

Get

Return \_type

End Get

Set(ByVal value As Char)

\_type = value

End Set

End Property

Public Sub New(ByVal pos As Vector3, ByVal type As Char)

\_position = pos

\_type = type

End Sub

End Class

Public Class MyMouse

Private \_firstMouseX As Integer

Public Property FirstMouseX() As Integer

Get

Return \_firstMouseX

End Get

Set(ByVal value As Integer)

\_firstMouseX = value

End Set

End Property

Private \_firstMouseY As Integer

Public Property FirstMouseY() As Integer

Get

Return \_firstMouseY

End Get

Set(ByVal value As Integer)

\_firstMouseY = value

End Set

End Property

End Class

End Module

# 3dOutput

Imports OpenTK

Imports OpenTK.Graphics.OpenGL

'TODO:: ALSO add back all 3 instructions labels once changed to list & pointer

'TODO:: Disble enter on undo button

Public Class \_3DOutput

Dim positions(0 To 26) As Vector3

Dim initialCubieRotation(26) As Quaternion

Dim scrambledCornerInput() As Corner

Dim scrambledEdgeInput() As Edge

Dim VBO, EBO, VAO As Integer

Dim cameraRotation As Single = -60.0

Const frontCameraAngle As Single = -63.0

Const cameraRadius As Single = 10

Dim cameraPitch As Single = 0

Dim deltaTime As Single = 0

Dim \_keys(256) As Boolean

Dim \_mouseClick As Boolean

Dim shaderProgram As MyShaderProgram

'Dim pad As Single = 0.03

Dim cubeRotationMatrix As Matrix4 = Matrix4.Identity

Dim undoing As Boolean = False

'ToDo list & pointer

Dim prevInstructions As New Stack(Of String)

Dim nextInstructions As New Stack(Of String)

ReadOnly projection As Matrix4 = Matrix4.CreateOrthographic(4 / 3 \* 7, 7, 1, 1000)

Public Sub New(ByVal corners() As Corner, ByVal edges() As Edge, ByVal Instructions As String())

' This call is required by the designer.

InitializeComponent()

scrambledCornerInput = corners

scrambledEdgeInput = edges

For i = Instructions.Length - 1 To 0 Step -1

nextInstructions.Push(Instructions(i))

Next

UpdateInstructionLabels()

End Sub

'previous button

Private Sub btnUndo\_MouseEnter(sender As Object, e As EventArgs) Handles btnUndo.MouseEnter, btnUndo.MouseHover

lblPrev.ForeColor = Color.Red

End Sub

Private Sub btnUndo\_MouseLeave(sender As Object, e As EventArgs) Handles btnUndo.MouseLeave

lblPrev.ForeColor = Color.DarkGray

End Sub

Private Sub btnUndo\_Click(sender As Object, e As EventArgs) Handles btnUndo.Click

If lblPrev.Text <> "- -" Then

Try

nextInstructions.Push(prevInstructions.Pop)

Catch

nextInstructions.Push("- -")

End Try

UpdateInstructionLabels()

UndoLastStage()

End If

If lblNext.Text <> "- -" Then

btnNext.Enabled = True

End If

End Sub

'next button

Private Sub btnNext\_MouseEnter(sender As Object, e As EventArgs) Handles btnNext.MouseEnter, btnNext.MouseHover

lblNext.ForeColor = Color.Red

End Sub

Private Sub btnNext\_MouseLeave(sender As Object, e As EventArgs) Handles btnNext.MouseLeave

lblNext.ForeColor = Color.Black

End Sub

Private Sub btnNext\_Click(sender As Object, e As EventArgs) Handles btnNext.Click

If shaderProgram.Handle <> -1 Then

btnNext.Enabled = False

End If

If lblNext.Text <> "- -" Then

Try

prevInstructions.Push(nextInstructions.Pop())

Catch

prevInstructions.Push("- -")

End Try

UpdateInstructionLabels()

If prevInstructions.Peek() <> "- -" Then

DoStage(prevInstructions.Peek())

End If

End If

If lblPrev.Text <> "- -" Then

btnUndo.Enabled = True

End If

End Sub

Private Sub DoStage(ByVal instruction As String)

Static top As Char = "W"

Static front As Char = "R"

Dim char1 As Char = Mid(instruction, 1, 1)

Dim char2 As Char = Mid(instruction, 2, 1)

Dim faces() As Char = MyPublic.AssociatedFaces(top, front)

If char1 <> "X" And char1 <> "Y" And char1 <> "Z" Then

Dim faceToTurn As Char

Select Case char1

Case "U" : faceToTurn = faces(0)

Case "L" : faceToTurn = faces(1)

Case "B" : faceToTurn = faces(2)

Case "R" : faceToTurn = faces(3)

Case "F" : faceToTurn = faces(4)

Case "D" : faceToTurn = faces(5)

End Select

\_keys(Asc(faceToTurn)) = True

If char2 = "'" Then

\_keys(16) = True

\_keys(17) = False

ElseIf char2 = "2" Then

\_keys(16) = False

\_keys(17) = True

Else

\_keys(16) = False

\_keys(17) = False

End If

Else

Select Case instruction

Case "X"

top = faces(4) 'front

front = faces(5) 'bottom

cubeRotationMatrix \*= Matrix4.CreateFromAxisAngle(New Vector3(1, 0, 0), -MathHelper.PiOver2)

Case "X'"

top = faces(2) 'back

front = faces(0) 'top

cubeRotationMatrix \*= Matrix4.CreateFromAxisAngle(New Vector3(1, 0, 0), MathHelper.PiOver2)

Case "X2"

top = faces(5) 'bottom

front = faces(2) 'back

cubeRotationMatrix \*= Matrix4.CreateFromAxisAngle(New Vector3(1, 0, 0), MathHelper.Pi)

Case "Y"

front = faces(3) 'right

cubeRotationMatrix \*= Matrix4.CreateFromAxisAngle(New Vector3(0, 1, 0), -MathHelper.PiOver2)

Case "Y'"

front = faces(1) 'left

cubeRotationMatrix \*= Matrix4.CreateFromAxisAngle(New Vector3(0, 1, 0), MathHelper.PiOver2)

Case "Y2"

front = faces(2) 'back

cubeRotationMatrix \*= Matrix4.CreateFromAxisAngle(New Vector3(0, 1, 0), MathHelper.Pi)

Case "Z"

top = faces(1) 'left

cubeRotationMatrix \*= Matrix4.CreateFromAxisAngle(New Vector3(0, 0, 1), -MathHelper.PiOver2)

Case "Z'"

top = faces(3) 'right

cubeRotationMatrix \*= Matrix4.CreateFromAxisAngle(New Vector3(0, 0, 1), MathHelper.PiOver2)

Case "Z2"

top = faces(5) 'bottom

cubeRotationMatrix \*= Matrix4.CreateFromAxisAngle(New Vector3(0, 0, 1), MathHelper.Pi)

End Select

btnNext.Enabled = True

End If

End Sub

Private Sub UndoLastStage()

undoing = True

Dim instruction, reverseInstruction As String

instruction = nextInstructions.Peek()

Select Case Mid(instruction, 2, 1)

Case ""

reverseInstruction = Mid(instruction, 1, 1) + "'"

Case "'"

reverseInstruction = Mid(instruction, 1, 1)

Case Else

reverseInstruction = instruction

End Select

DoStage(reverseInstruction)

End Sub

Private Sub GlControl1\_Load(sender As Object, e As EventArgs) Handles GlControl1.Load

Console.WriteLine("Load")

GL.Viewport(0, 0, GlControl1.Width, GlControl1.Height)

GL.Enable(EnableCap.DepthTest)

GL.DepthFunc(DepthFunction.Lequal)

Console.WriteLine("gl version: " + GL.GetString(StringName.Version))

sldrAmbient.Value = 23

shaderProgram = New MyShaderProgram("ambientVertex.vrt", "ambientFragment.frg")

If shaderProgram.Handle <> -1 Then

LoadFor3D(scrambledCornerInput, scrambledEdgeInput)

Else

lblError.Visible = True

End If

End Sub

Private Sub GlControl1\_MouseDown(sender As Object, e As MouseEventArgs) Handles GlControl1.MouseDown

Dim mousePos As New MyMouse

If Not \_mouseClick Then

mousePos.FirstMouseX = OpenTK.Input.Mouse.GetCursorState.X

'Console.WriteLine(mouse(0))

mousePos.FirstMouseY = OpenTK.Input.Mouse.GetCursorState.Y

'Console.WriteLine(mouse(1))

End If

\_mouseClick = True

MoveCamera(mousePos, True)

End Sub

Private Sub GlControl1\_MouseUp(sender As Object, e As MouseEventArgs) Handles GlControl1.MouseUp

\_mouseClick = False

End Sub

Private Sub OnKeyUp(sender As Object, e As KeyEventArgs) Handles GlControl1.KeyUp, Me.KeyUp

\_keys(e.KeyCode) = False

End Sub

Private Sub OnKeyDown(sender As Object, e As KeyEventArgs) Handles GlControl1.KeyDown, Me.KeyDown

\_keys(e.KeyCode) = True

End Sub

Private Sub \_3DOutput\_FormClosed(sender As Object, e As FormClosedEventArgs) Handles MyBase.FormClosed

If shaderProgram.Handle <> -1 Then

GL.DeleteVertexArray(VAO)

GL.DeleteBuffer(VBO)

GL.DeleteBuffer(EBO)

End If

End Sub

Private Sub LoadFor3D(ByVal corners() As Corner, ByVal edges() As Edge)

Console.WriteLine("3D Load")

Dim vertices3d As Vertex3D()

Dim indices As UInteger()

' BLOCK WITH NORMALS

vertices3d = New Vertex3D(0 To 23) {

New Vertex3D(New Vector3(0, 0, 0), Color.DarkOrange, New Vector3(0, 0, -1)),

New Vertex3D(New Vector3(1, 0, 0), Color.DarkOrange, New Vector3(0, 0, -1)),

New Vertex3D(New Vector3(1, 1, 0), Color.DarkOrange, New Vector3(0, 0, -1)),

New Vertex3D(New Vector3(0, 1, 0), Color.DarkOrange, New Vector3(0, 0, -1)),

New Vertex3D(New Vector3(0, 0, 1), Color.Red, New Vector3(0, 0, 1)),

New Vertex3D(New Vector3(1, 0, 1), Color.Red, New Vector3(0, 0, 1)),

New Vertex3D(New Vector3(1, 1, 1), Color.Red, New Vector3(0, 0, 1)),

New Vertex3D(New Vector3(0, 1, 1), Color.Red, New Vector3(0, 0, 1)),

New Vertex3D(New Vector3(0, 0, 0), Color.Green, New Vector3(-1, 0, 0)),

New Vertex3D(New Vector3(1, 0, 0), Color.Blue, New Vector3(1, 0, 0)),

New Vertex3D(New Vector3(1, 1, 0), Color.Blue, New Vector3(1, 0, 0)),

New Vertex3D(New Vector3(0, 1, 0), Color.Green, New Vector3(-1, 0, 0)),

New Vertex3D(New Vector3(0, 0, 1), Color.Green, New Vector3(-1, 0, 0)),

New Vertex3D(New Vector3(1, 0, 1), Color.Blue, New Vector3(1, 0, 0)),

New Vertex3D(New Vector3(1, 1, 1), Color.Blue, New Vector3(1, 0, 0)),

New Vertex3D(New Vector3(0, 1, 1), Color.Green, New Vector3(-1, 0, 0)),

New Vertex3D(New Vector3(0, 0, 0), Color.Yellow, New Vector3(0, -1, 0)),

New Vertex3D(New Vector3(1, 0, 0), Color.Yellow, New Vector3(0, -1, 0)),

New Vertex3D(New Vector3(1, 1, 0), Color.White, New Vector3(0, 1, 0)),

New Vertex3D(New Vector3(0, 1, 0), Color.White, New Vector3(0, 1, 0)),

New Vertex3D(New Vector3(0, 0, 1), Color.Yellow, New Vector3(0, -1, 0)),

New Vertex3D(New Vector3(1, 0, 1), Color.Yellow, New Vector3(0, -1, 0)),

New Vertex3D(New Vector3(1, 1, 1), Color.White, New Vector3(0, 1, 0)),

New Vertex3D(New Vector3(0, 1, 1), Color.White, New Vector3(0, 1, 0))}

indices = New UInteger(0 To 35) {

4, 5, 6, 4, 6, 7,

8, 11, 15, 8, 12, 15,

16, 20, 21, 16, 17, 21,

9, 10, 14, 9, 13, 14,

19, 18, 22, 19, 22, 23,

0, 1, 2, 0, 2, 3}

LoadPositions(corners, edges)

LoadInitialRotation(corners, edges)

VAO = GL.GenVertexArray()

VBO = GL.GenBuffer()

EBO = GL.GenBuffer()

GL.BindVertexArray(VAO) ' Bind the VAO first Before binding and setting buffers/pointers

GL.BindBuffer(BufferTarget.ArrayBuffer, VBO)

GL.BufferData(BufferTarget.ArrayBuffer, CType(Vertex3D.SizeInBytes \* vertices3d.Length, IntPtr), vertices3d, BufferUsageHint.StaticDraw)

GL.VertexAttribPointer(0, 3, VertexAttribPointerType.Float, False, Vertex3D.SizeInBytes, 0)

GL.EnableVertexAttribArray(0)

GL.VertexAttribPointer(1, 4, VertexAttribPointerType.Float, False, Vertex3D.SizeInBytes, Vector3.SizeInBytes)

GL.EnableVertexAttribArray(1)

GL.VertexAttribPointer(2, 3, VertexAttribPointerType.Float, False, Vertex3D.SizeInBytes, Vector3.SizeInBytes + Vector4.SizeInBytes)

GL.EnableVertexAttribArray(2)

GL.BindBuffer(BufferTarget.ElementArrayBuffer, EBO)

GL.BufferData(BufferTarget.ElementArrayBuffer, CType(4 \* indices.Length, IntPtr), indices, BufferUsageHint.StaticDraw)

GL.BindBuffer(BufferTarget.ArrayBuffer, 0)

GL.BindVertexArray(0) ' Unbinds VAO

GL.PolygonMode(MaterialFace.FrontAndBack, PolygonMode.Fill)

End Sub

Private Sub LoadPositions(ByVal scrambledCorners() As Corner, ByVal scrambledEdges() As Edge)

'CUBE

Static corners() As Corner

Static edges() As Edge

If scrambledCorners IsNot Nothing And scrambledEdges IsNot Nothing Then

corners = scrambledCorners

edges = scrambledEdges

End If

Dim pad As Single = 5 / 100

Dim padLoc As Integer = GL.GetUniformLocation(shaderProgram.Handle, "pad")

GL.Uniform1(padLoc, pad)

'Console.WriteLine(PadSlider.Value)

For i = 0 To 7

positions(i) = New Vector3()

Select Case corners(i).Position.x

Case -1

positions(i).X = -1.5 - pad

Case 0

positions(i).X = -0.5

Case 1

positions(i).X = 0.5 + pad

End Select

Select Case corners(i).Position.y

Case -1

positions(i).Y = -1.5 - pad

Case 0

positions(i).Y = -0.5

Case 1

positions(i).Y = 0.5 + pad

End Select

Select Case corners(i).Position.z

Case -1

positions(i).Z = -1.5 - pad

Case 0

positions(i).Z = -0.5

Case 1

positions(i).Z = 0.5 + pad

End Select

Next

For i = 0 To 11

positions(i + 8) = New Vector3()

Select Case edges(i).Position.x

Case -1

positions(i + 8).X = -1.5 - pad

Case 0

positions(i + 8).X = -0.5

Case 1

positions(i + 8).X = 0.5 + pad

End Select

Select Case edges(i).Position.y

Case -1

positions(i + 8).Y = -1.5 - pad

Case 0

positions(i + 8).Y = -0.5

Case 1

positions(i + 8).Y = 0.5 + pad

End Select

Select Case edges(i).Position.z

Case -1

positions(i + 8).Z = -1.5 - pad

Case 0

positions(i + 8).Z = -0.5

Case 1

positions(i + 8).Z = 0.5 + pad

End Select

Next

positions(20) = New Vector3(-0.5, 0.5 + pad, -0.5)

positions(21) = New Vector3(-1.5 - pad, -0.5, -0.5)

positions(22) = New Vector3(-0.5, -0.5, -1.5 - pad)

positions(23) = New Vector3(0.5 + pad, -0.5, -0.5)

positions(24) = New Vector3(-0.5, -0.5, 0.5 + pad)

positions(25) = New Vector3(-0.5, -1.5 - pad, -0.5)

positions(26) = New Vector3(-0.5, -0.5, -0.5)

End Sub

Private Sub LoadInitialRotation(ByVal scrambledCorners() As Corner, ByVal scrambledEdges() As Edge)

Dim initalBlocks(19) As Block

For i = 0 To 7

initalBlocks(i) = scrambledCorners(i)

Next

For i = 0 To 11

initalBlocks(i + 8) = scrambledEdges(i)

Next

Dim cubie As Block

For i = 0 To 19

cubie = initalBlocks(i)

Dim vec As Vector3 = New Vector3

Dim whiteNormal As Vector3 = New Vector3(0, 1, 0)

Dim redNormal As Vector3 = New Vector3(0, 0, 1)

Select Case cubie.PrimaryFace

Case "W"

Select Case cubie.Rotation

Case "Y" : vec.X = 2

Case "R" : vec.X = 1

Case "O" : vec.X = -1

Case "G" : vec.Z = 1

Case "B" : vec.Z = -1

End Select

Case "Y"

Select Case cubie.Rotation

Case "W" : vec.X = 2

Case "R" : vec.X = -1

Case "O" : vec.X = 1

Case "G" : vec.Z = -1

Case "B" : vec.Z = 1

End Select

Case "R"

Select Case cubie.Rotation

Case "W" : vec.X = -1

Case "Y" : vec.X = 1

Case "O" : vec.X = 2

Case "G" : vec.Y = -1

Case "B" : vec.Y = 1

End Select

Case "O"

Select Case cubie.Rotation

Case "W" : vec.X = 1

Case "Y" : vec.X = -1

Case "R" : vec.X = 2

Case "G" : vec.Y = 1

Case "B" : vec.Y = -1

End Select

End Select

Dim quart As Quaternion = New Quaternion

quart.X = vec.X \* Math.Sin(Math.Abs(vec.X) \* MathHelper.PiOver2 / 2)

quart.Y = vec.Y \* Math.Sin(Math.Abs(vec.Y) \* MathHelper.PiOver2 / 2)

quart.Z = vec.Z \* Math.Sin(Math.Abs(vec.Z) \* MathHelper.PiOver2 / 2)

Dim vecSum As Integer = Math.Abs(vec.X) + Math.Abs(vec.Y) + Math.Abs(vec.Z)

quart.W = Math.Cos(vecSum \* MathHelper.PiOver2 / 2)

If quart.Length = quart.W Then

quart = Quaternion.Identity

End If

recalculateNormals(whiteNormal, redNormal, quart)

Dim correctWhiteNormal, correctRedNormal As Vector3

GetNormalsFromCubie(cubie, correctWhiteNormal, correctRedNormal)

Dim current2d, correct2d As Vector2

Dim angle As Single

Dim axis As Vector3

angle = 0

If whiteNormal = correctWhiteNormal Then

'rotate around white normal

axis = whiteNormal

If redNormal = correctRedNormal Then

angle = 0

ElseIf Math.Abs(Vector3.Dot(redNormal, correctRedNormal)) = 1 Then

angle = MathHelper.Pi

Else

'collapse the vectors onto 1 plane, perpendicular to white normal

If whiteNormal.X <> 0 Then

current2d = New Vector2(-redNormal.Z, redNormal.Y)

correct2d = New Vector2(-correctRedNormal.Z, correctRedNormal.Y)

ElseIf whiteNormal.Y <> 0 Then

current2d = New Vector2(redNormal.X, -redNormal.Z)

correct2d = New Vector2(correctRedNormal.X, -correctRedNormal.Z)

ElseIf whiteNormal.Z <> 0 Then

current2d = New Vector2(redNormal.X, redNormal.Y)

correct2d = New Vector2(correctRedNormal.X, correctRedNormal.Y)

End If

If current2d.Y <> 0 Then

If current2d.Y = correct2d.X Then

angle = -MathHelper.PiOver2

Else

angle = MathHelper.PiOver2

End If

ElseIf current2d.X <> 0 Then

If current2d.X = correct2d.Y Then

angle = MathHelper.PiOver2

Else

angle = -MathHelper.PiOver2

End If

End If

End If

ElseIf redNormal = correctRedNormal Then

'rotate around red normal

axis = redNormal

If whiteNormal = correctWhiteNormal Then

angle = 0

ElseIf Math.Abs(Vector3.Dot(whiteNormal, correctWhiteNormal)) = 1 Then

angle = MathHelper.Pi

Else

'collapse the vectors onto 1 plane, perpendicular to white normal

If redNormal.X <> 0 Then

current2d = New Vector2(-whiteNormal.Z, whiteNormal.Y)

correct2d = New Vector2(-correctWhiteNormal.Z, correctWhiteNormal.Y)

ElseIf redNormal.Y <> 0 Then

current2d = New Vector2(whiteNormal.X, -whiteNormal.Z)

correct2d = New Vector2(correctWhiteNormal.X, -correctWhiteNormal.Z)

ElseIf redNormal.Z <> 0 Then

current2d = New Vector2(whiteNormal.X, whiteNormal.Y)

correct2d = New Vector2(correctWhiteNormal.X, correctWhiteNormal.Y)

End If

If current2d.Y <> 0 Then

If current2d.Y = correct2d.X Then

angle = -MathHelper.PiOver2

Else

angle = MathHelper.PiOver2

End If

ElseIf current2d.X <> 0 Then

If current2d.X = correct2d.Y Then

angle = MathHelper.PiOver2

Else

angle = -MathHelper.PiOver2

End If

End If

End If

End If

Dim quart2 As Quaternion

quart2 = Quaternion.Identity

If angle <> 0 Then

quart2.X = Math.Abs(axis.X) \* Math.Sin(angle / 2)

quart2.Y = Math.Abs(axis.Y) \* Math.Sin(angle / 2)

quart2.Z = Math.Abs(axis.Z) \* Math.Sin(angle / 2)

quart2.W = Math.Cos(angle / 2)

Else

quart2 = Quaternion.Identity

End If

quart = quart2 \* quart

initialCubieRotation(i) = quart

Next

For i = 20 To 26

initialCubieRotation(i) = Quaternion.Identity

Next

End Sub

Private Sub recalculateNormals(ByRef whiteNormal As Vector3, ByRef redNormal As Vector3, ByVal quart As Quaternion)

Dim rotationMatrix As Matrix4

rotationMatrix = Matrix4.CreateTranslation(New Vector3(-0.5, -0.5, -0.5))

rotationMatrix \*= Matrix4.CreateFromQuaternion(quart)

rotationMatrix \*= Matrix4.CreateTranslation(-(New Vector3(-0.5, -0.5, -0.5)))

Dim myRotMatrix As Matrices.Matrix4x4

myRotMatrix = New Matrix4x4(rotationMatrix)

myRotMatrix.Row1.w = 0

myRotMatrix.Row2.w = 0

myRotMatrix.Row3.w = 0

Dim myRedNormal, myWhiteNormal As Matrices.Vector4x1

myWhiteNormal = New Vector4x1(New Vector4(whiteNormal, 1))

myRedNormal = New Vector4x1(New Vector4(redNormal, 1))

myWhiteNormal = myRotMatrix \* myWhiteNormal

myRedNormal = myRotMatrix \* myRedNormal

whiteNormal = New Vector3(Math.Round(myWhiteNormal.x, 4), Math.Round(myWhiteNormal.y, 4), Math.Round(myWhiteNormal.z, 4))

redNormal = New Vector3(Math.Round(myRedNormal.x, 4), Math.Round(myRedNormal.y, 4), Math.Round(myRedNormal.z, 4))

End Sub

Private Function GetFaceOfNormal(ByRef normal As Vector3) As Char

If normal.X = 1 Then

Return "B"

ElseIf normal.X = -1 Then

Return "G"

ElseIf normal.Y = 1 Then

Return "W"

ElseIf normal.Y = -1 Then

Return "Y"

ElseIf normal.Z = 1 Then

Return "R"

ElseIf normal.Z = "-1" Then

Return "O"

Else

Return ""

End If

End Function

Private Sub GetNormalsFromCubie(ByVal cubie As Block, ByRef whiteNormal As Vector3, ByRef redNormal As Vector3)

Dim whiteSet, redSet As Boolean

whiteSet = False

redSet = False

Dim secondaryRotation, secondaryFace As Char

secondaryRotation = ""

secondaryFace = ""

If cubie.GetType() = GetType(Corner) Then

Dim cornerCubie As Corner = cubie

secondaryRotation = cornerCubie.SecondaryRotation

secondaryFace = cornerCubie.SecondaryFace

ElseIf cubie.GetType() = GetType(Edge) Then

For Each colour In cubie.Colours

If colour <> "" And colour <> cubie.PrimaryFace Then

secondaryFace = colour

Exit For

End If

Next

For Each facecol As Char In MyPublic.PublicVars.FACE\_COLOURS.ToUpper()

If facecol <> cubie.Rotation Then

Dim face As MyPublic.Faces

face = MyPublic.ColourChar2FaceNumber(facecol)

If OnFace(cubie, face, "W", "R") Then

secondaryRotation = facecol

Exit For

End If

End If

Next

End If

Select Case cubie.PrimaryFace

Case "W", "Y"

Select Case cubie.Rotation

Case "W" : whiteNormal = New Vector3(0, 1, 0)

Case "Y" : whiteNormal = New Vector3(0, -1, 0)

Case "R" : whiteNormal = New Vector3(0, 0, 1)

Case "O" : whiteNormal = New Vector3(0, 0, -1)

Case "G" : whiteNormal = New Vector3(-1, 0, 0)

Case "B" : whiteNormal = New Vector3(1, 0, 0)

End Select

If cubie.PrimaryFace = "Y" Then

whiteNormal = -whiteNormal

End If

whiteSet = True

Case "R", "O"

Select Case cubie.Rotation

Case "W" : redNormal = New Vector3(0, 1, 0)

Case "Y" : redNormal = New Vector3(0, -1, 0)

Case "R" : redNormal = New Vector3(0, 0, 1)

Case "O" : redNormal = New Vector3(0, 0, -1)

Case "G" : redNormal = New Vector3(-1, 0, 0)

Case "B" : redNormal = New Vector3(1, 0, 0)

End Select

If cubie.PrimaryFace = "O" Then

redNormal = -redNormal

End If

redSet = True

End Select

Select Case secondaryFace

Case "R", "O"

Select Case secondaryRotation

Case "W" : redNormal = New Vector3(0, 1, 0)

Case "Y" : redNormal = New Vector3(0, -1, 0)

Case "R" : redNormal = New Vector3(0, 0, 1)

Case "O" : redNormal = New Vector3(0, 0, -1)

Case "G" : redNormal = New Vector3(-1, 0, 0)

Case "B" : redNormal = New Vector3(1, 0, 0)

End Select

If secondaryFace = "O" Then

redNormal = -redNormal

End If

redSet = True

Case "G", "B"

Dim greenNormal As Vector3

Select Case secondaryRotation

Case "W" : greenNormal = New Vector3(0, 1, 0)

Case "Y" : greenNormal = New Vector3(0, -1, 0)

Case "R" : greenNormal = New Vector3(0, 0, 1)

Case "O" : greenNormal = New Vector3(0, 0, -1)

Case "G" : greenNormal = New Vector3(-1, 0, 0)

Case "B" : greenNormal = New Vector3(1, 0, 0)

End Select

If secondaryFace = "B" Then

greenNormal = -greenNormal

End If

If whiteSet Then

redNormal = Vector3.Cross(whiteNormal, greenNormal)

ElseIf redSet Then

'Not sure if this should be +ve or -ve

whiteNormal = -Vector3.Cross(redNormal, greenNormal)

Else

Throw New NotImplementedException("GetNormalsFromCubie: primary normal not set!")

End If

End Select

End Sub

Private Sub LoadLighting()

Static ambientStrength As Single = 0.95

Dim lightColorLoc As Integer = GL.GetUniformLocation(shaderProgram.Handle, "lightColor")

Dim brightness As Single = 255

Dim white As OpenTK.Graphics.Color4 = New OpenTK.Graphics.Color4(255, 255, 255, 255)

ambientStrength = sldrAmbient.Value / 20

Dim ambient As Vector4 = New Vector4(white.R / 255, white.G / 255, white.B / 255, 1)

GL.Uniform4(lightColorLoc, ambient.X, ambient.Y, ambient.Z, ambient.W)

Dim ambientStrengthLoc As Integer = GL.GetUniformLocation(shaderProgram.Handle, "ambientStrength")

GL.Uniform1(ambientStrengthLoc, ambientStrength)

End Sub

Private Sub Render()

GlControl1.SwapBuffers()

GlControl1.Invalidate() ' calls paint

End Sub

Private Sub GlControl1\_Paint(sender As Object, e As PaintEventArgs) Handles GlControl1.Paint

If shaderProgram.Handle <> -1 Then

Dim currentFrame As Decimal

Static lastFrame As Decimal

Static firsttime As Boolean = True

Dim camX As Single = Math.Sin(cameraRotation / 10) \* cameraRadius

Dim camZ As Single = Math.Cos(cameraRotation / 10) \* cameraRadius

Dim view As Matrix4

If firsttime Then

deltaTime = 5

firsttime = False

End If

Do

currentFrame = DateTime.Now.Ticks / 10000

deltaTime = currentFrame - lastFrame

Loop While deltaTime = 0

GL.ClearColor(Color.CornflowerBlue)

GL.Clear(ClearBufferMask.ColorBufferBit Or ClearBufferMask.DepthBufferBit)

view = Matrix4.LookAt(New Vector3(camX, cameraPitch, camZ), New Vector3(0, 0, 0), New Vector3(0, 1, 0))

Dim viewPosLoc As Integer = GL.GetUniformLocation(shaderProgram.Handle, "viewPos")

GL.Uniform3(viewPosLoc, camX, cameraPitch, camZ)

Dim NoChangeMouse As New MyMouse

RenderCube(view, projection)

MoveCamera(NoChangeMouse, False)

Render()

lastFrame = DateTime.Now.Ticks / 10000

End If

End Sub

Private Sub MoveCamera(ByVal MousePos As MyMouse, ByVal PositionUpdated As Boolean)

Static StoredMousePos As New MyMouse

If PositionUpdated = True Then

StoredMousePos.FirstMouseX = MousePos.FirstMouseX

StoredMousePos.FirstMouseY = MousePos.FirstMouseY

End If

Dim cameraSpeed As Single = 0.07 \* deltaTime

If Not \_mouseClick Then

If cameraPitch > 2 + (cameraSpeed \* 2.2) Then

cameraPitch -= cameraSpeed \* 2.2

ElseIf cameraPitch < 2 - (cameraSpeed \* 2.2) Then

cameraPitch += cameraSpeed \* 2.2

ElseIf cameraPitch >= 2 - (cameraSpeed \* 2.2) And cameraPitch <= 2 + (cameraSpeed \* 2.2) Then

cameraPitch = 2

End If

End If

'End If

If \_mouseClick Then

Dim xPosition As Integer = OpenTK.Input.Mouse.GetCursorState.X

Dim yPosition As Integer = OpenTK.Input.Mouse.GetCursorState.Y

Dim xOffset As Integer = xPosition - StoredMousePos.FirstMouseX

Dim yOffset As Integer = yPosition - StoredMousePos.FirstMouseY

Dim sensitivity As Single = 0.1

StoredMousePos.FirstMouseX = xPosition

StoredMousePos.FirstMouseY = yPosition

If cameraPitch <= 25 And cameraPitch >= -20 Then

cameraPitch += (yOffset \* sensitivity)

ElseIf cameraPitch > 25 Then

cameraPitch = 25

ElseIf cameraPitch < -20 Then

cameraPitch = -20

End If

If (cameraRotation - (xOffset \* sensitivity \* 0.5)) > (frontCameraAngle - 6) And (cameraRotation - (xOffset \* sensitivity \* 0.5)) < (frontCameraAngle + 6) Then

cameraRotation -= (xOffset \* sensitivity \* 0.5)

End If

End If

End Sub

Private Sub RenderCube(ByRef view As Matrix4, ByRef projection As Matrix4)

shaderProgram.Use()

LoadLighting()

Dim projectionLocation As Integer = GL.GetUniformLocation(shaderProgram.Handle, "projection")

GL.UniformMatrix4(projectionLocation, False, projection)

Dim viewLocation As Integer = GL.GetUniformLocation(shaderProgram.Handle, "view")

GL.UniformMatrix4(viewLocation, False, view)

Dim model As Matrix4

model = Matrix4.Identity

GL.Enable(EnableCap.DepthTest)

GL.DepthFunc(DepthFunction.Less)

RenderBlocks()

End Sub

Private Sub RenderBlocks()

'TODO CHANGE TO INDEXES OF AXIS READONLY ARRAY

'TODO CHANGE TO STORED ROTATION MATRICIES? OR INDEXES TO READONLY AS THEY WILL ONLY BE 90 degrees

Dim rotationSpeed As Single = sldrSpeed.Value / 1000

'Const rotationSpeed As Single = 0.010

'Dynamic Jagged Array?

Static rotationAxis(26)() As Vector3

Static firstTime As Boolean = True

Static cublets(26) As Block

Static showRotation As Boolean = False

Static rotateBlock(26) As Boolean

If firstTime Then

For i = 0 To 7

cublets(i) = scrambledCornerInput(i)

Next

For i = 0 To 11

cublets(i + 8) = scrambledEdgeInput(i)

Next

CreateMiddles(cublets)

For i = 0 To positions.Length - 1

AppendRotationVector(rotationAxis, i, New Vector3(0, 0, 0), cublets)

Next

firstTime = False

End If

Dim pad As Single = 5 / 100

Dim padLoc As Integer = GL.GetUniformLocation(shaderProgram.Handle, "pad")

GL.Uniform1(padLoc, pad)

GL.BindVertexArray(VAO)

Static displayAngle As Double = 0

Static face As Char

If \_keys(17) Then ' 180 degrees

For Each character In MyPublic.FACE\_COLOURS.ToUpper()

If \_keys(Asc(character)) Then

face = character

Exit For

End If

Next

End If

'TODO change to select cases?

If \_keys(16) Then 'Shift

If \_keys(Asc("W")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.W, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(0, 1, 0), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.W, Direction.Anticlockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("W")) = False

showRotation = True

ElseIf \_keys(Asc("Y")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.Y, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(0, -1, 0), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.Y, Direction.Anticlockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("Y")) = False

showRotation = True

ElseIf \_keys(Asc("G")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.G, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(-1, 0, 0), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.G, Direction.Anticlockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("G")) = False

showRotation = True

ElseIf \_keys(Asc("B")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.B, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(1, 0, 0), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.B, Direction.Anticlockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("B")) = False

showRotation = True

ElseIf \_keys(Asc("R")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.R, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(0, 0, 1), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.R, Direction.Anticlockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("R")) = False

showRotation = True

ElseIf \_keys(Asc("O")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.O, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(0, 0, -1), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.O, Direction.Anticlockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("O")) = False

showRotation = True

End If

Else

If \_keys(Asc("W")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.W, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(0, -1, 0), cublets)

rotateBlock(i) = True

'Console.WriteLine(i)

End If

Next

RotateFace(MyPublic.Faces.W, Direction.Clockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("W")) = False

showRotation = True

ElseIf \_keys(Asc("Y")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.Y, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(0, 1, 0), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.Y, Direction.Clockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("Y")) = False

showRotation = True

ElseIf \_keys(Asc("G")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.G, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(1, 0, 0), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.G, Direction.Clockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("G")) = False

showRotation = True

ElseIf \_keys(Asc("B")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.B, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(-1, 0, 0), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.B, Direction.Clockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("B")) = False

showRotation = True

ElseIf \_keys(Asc("R")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.R, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(0, 0, -1), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.R, Direction.Clockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("R")) = False

showRotation = True

ElseIf \_keys(Asc("O")) Then

For i = 0 To 26

If OnFace(cublets(i), MyPublic.Faces.O, "W", "R") Then

AppendRotationVector(rotationAxis, i, New Vector3(0, 0, 1), cublets)

rotateBlock(i) = True

End If

Next

RotateFace(MyPublic.Faces.O, Direction.Clockwise, scrambledCornerInput, scrambledEdgeInput, "W", "R", "")

\_keys(Asc("O")) = False

showRotation = True

End If

End If

If showRotation Then

If undoing Then

displayAngle = MathHelper.PiOver2

End If

displayAngle += deltaTime \* rotationSpeed

If displayAngle >= MathHelper.PiOver2 Then

displayAngle = MathHelper.PiOver2

btnNext.Enabled = True

showRotation = False

End If

Else

displayAngle = MathHelper.PiOver2

If \_keys(17) Then 'if 180 degrees reset key for another 90 deg. rotation

\_keys(Asc(face)) = True

\_keys(17) = False

End If

End If

For i = 0 To 26

Dim rotationMatrix As Matrix4 = Matrix4.Identity

Dim model As Matrix4

model = Matrix4.CreateTranslation(positions(26))

model \*= Matrix4.CreateFromQuaternion(initialCubieRotation(i))

model \*= Matrix4.CreateTranslation(-positions(26))

model \*= Matrix4.CreateTranslation(positions(i))

'model matrix passed to shaders to set colours in correct places

Dim modelUniformLoc As Integer = GL.GetUniformLocation(shaderProgram.Handle, "colourMat")

GL.UniformMatrix4(modelUniformLoc, False, model)

Try

For j = 1 To rotationAxis(i).Length - 1

If j = rotationAxis(i).Length - 1 And rotateBlock(i) Then

rotationMatrix \*= Matrix4.CreateFromAxisAngle(rotationAxis(i)(rotationAxis(i).Length - 1), displayAngle)

Else

rotationMatrix \*= Matrix4.CreateFromAxisAngle(rotationAxis(i)(j), MathHelper.PiOver2)

End If

Next

'ToDo Show this rotation

rotationMatrix \*= cubeRotationMatrix

Catch ex As NullReferenceException

Console.WriteLine("NULL")

End Try

model \*= rotationMatrix

Dim modelLoc As Integer = GL.GetUniformLocation(shaderProgram.Handle, "model")

GL.UniformMatrix4(modelLoc, False, model)

GL.DrawElements(PrimitiveType.Triangles, 36, DrawElementsType.UnsignedInt, 0)

Next

If displayAngle = MathHelper.PiOver2 Then

displayAngle = 0

For i = 0 To 26

rotateBlock(i) = False

Next

End If

If undoing And Not \_keys(17) Then

undoing = False

btnNext.Enabled = True

End If

End Sub

Private Sub CreateMiddles(ByRef cublets() As Block)

For i = 20 To 26

cublets(i) = New Middle

Next

cublets(20).Name = "W"

cublets(21).Name = "G"

cublets(22).Name = "O"

cublets(23).Name = "B"

cublets(24).Name = "R"

cublets(25).Name = "Y"

cublets(26).Name = ""

For i = 20 To 26

cublets(i).Colours(0) = cublets(i).Name

cublets(i).PrimaryFace = cublets(i).Name

cublets(i).Rotation = cublets(i).Name

Next

cublets(20).Position = New Vector3x1(0, 1, 0)

cublets(21).Position = New Vector3x1(-1, 0, 0)

cublets(22).Position = New Vector3x1(0, 0, -1)

cublets(23).Position = New Vector3x1(1, 0, 0)

cublets(24).Position = New Vector3x1(0, 0, 1)

cublets(25).Position = New Vector3x1(0, -1, 0)

cublets(26).Position = New Vector3x1(0, 0, 0)

End Sub

Private Sub UpdateInstructionLabels()

Try

lblPrev.Text = prevInstructions.Peek

lblPrev.Refresh()

Catch

lblPrev.Text = "- -"

btnUndo.Enabled = False

End Try

Try

lblNext.Text = nextInstructions.Peek

lblNext.Refresh()

Catch

lblNext.Text = "- -"

btnNext.Enabled = False

End Try

End Sub

Private Sub AppendRotationVector(ByRef array()() As Vector3, ByVal blockIndex As Integer, ByVal vector As Vector3, ByRef blockArray As Block())

Dim nextIndex As Integer

Try

nextIndex = array(blockIndex).Length

Catch ex As NullReferenceException

nextIndex = 0

End Try

ReDim Preserve array(blockIndex)(nextIndex)

array(blockIndex)(nextIndex) = vector

'RotateBlockPosition()

End Sub

End Class