## MainMenu.vb

Public Class MainMenu

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

Dim inputForm As New Input()

inputForm.Show()

Me.Close()

End Sub

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

LoadCube()

End Sub

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

Dim creditsForm As New Credits()

creditsForm.ShowDialog()

End Sub

Private Sub LoadCube()

Dim fileBrowser As New OpenFileDialog()

fileBrowser.Filter = "cube files (\*.cube)|\*.cube"

fileBrowser.ShowDialog()

Try

Dim filePath As String = fileBrowser.FileName

If filePath = "" Then Return

Dim outputForm As New \_3DOutput(filePath)

outputForm.Show()

Me.Close()

Catch ex As Exception

MsgBox("Cannot Read File: " & ex.Message)

End Try

End Sub

End Class

## Input.vb

Imports System.IO

Imports RubiksCubeSolver\_v2\_0.Helpers

Public Class Input

ReadOnly buttonColours As Color() = {Color.White, Color.Lime, Color.Red, Color.Blue, Color.Yellow, Color.DarkOrange}

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

Private Sub ColourChange\_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

DirectCast(sender, Button).BackColor = GetNextButtonColour(sender)

End Sub

''' <summary> Returns the next colour in the sequence for a specific button </summary>

Private Function GetNextButtonColour(ByVal button As Button) As Color

Return buttonColours((Array.IndexOf(buttonColours, button.BackColor) + 1) Mod 6)

End Function

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

btnReset.Enabled = False

Dim stickerColours As Char(,) = ConvertButtonColoursToCharArray(Controls)

If Not Analyse.CubePossible(stickerColours) Then

Return

End If

Dim cube As New Cube(stickerColours)

Dim processingWindow As New Processing(cube)

processingWindow.Show()

Me.Close()

End Sub

''' <summary> Reads each of the sticker buttons and returns an array of their colours </summary>

Private Function ConvertButtonColoursToCharArray(ByVal controls As Control.ControlCollection) As Char(,)

Dim stickersArray(5, 8) As Char

For face = 0 To 5

For Each sticker As Char In STICKER\_LETTERS.ToCharArray()

Dim buttonName As String = sticker & (face + 1).ToString() 'e.g. "a2"

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

Dim stickerNumber As Integer = InStr(STICKER\_LETTERS, sticker) - 1

'InStr() used to get (position of sticker in "abcdefghi" - 1) == y-coordinate of sticker

stickersArray(face, stickerNumber) = GetButtonColour(button, controls)

Next

Next

Return stickersArray

End Function

'''<summary> Gets the colour character of a specific button </summary>

Private Function GetButtonColour(ByVal box As Button, ByVal controls As Control.ControlCollection) As Char

Dim buttonColour As Char = Mid(box.BackColor.ToString(), 8, 1)

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

If buttonColour = "D" Then

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

ElseIf buttonColour = "L" Then

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

End If

Return buttonColour

End Function

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

ResetBoxColours(Controls)

ResetStickersArray(\_stickers)

End Sub

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

Dim main As New MainMenu

main.Show()

Me.Close()

End Sub

''' <summary> Resets all the colours of the buttons to their default colours </summary>

Private Sub ResetBoxColours(ByVal controls As Control.ControlCollection)

For Each faceChar As Char In STICKER\_LETTERS\_NO\_MIDDLE.ToCharArray()

For faceNumber = 1 To 6

Dim face As Helpers.FaceColour = faceNumber - 1

Dim buttonName As String = faceChar & faceNumber.ToString() 'e.g. "b5"

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

button.BackColor = buttonColours(face)

Next

Next

End Sub

Private Sub ResetStickersArray(ByRef stickersArray(,) As Char)

For i = 0 To 5

Dim face As Helpers.FaceColour = i

For j = 0 To 8

stickersArray(i, j) = face.ToString()

Next

Next

End Sub

End Class

## Analyse.vb

Imports RubiksCubeSolver\_v2\_0.Helpers.PublicFunctions

Imports RubiksCubeSolver\_v2\_0.Helpers.PublicConstants

Public Module Analyse

''' <summary> Returns true if the cube is fully possible </summary>

Public Function CubePossible(ByVal stickers(,) As Char) As Boolean

If Not NineOfEachColourSticker(stickers) Then Return False

If Not EdgesPossible(stickers) Then Return False

If Not CornersPossible(stickers) Then Return False

Return CornerRotationsPossible(stickers)

End Function

''' <summary> Returns true if there are 9 of each colour sticker </summary>

Private Function NineOfEachColourSticker(ByVal stickers(,) As Char) As Boolean

If AnythingElseInArray(NumberOfEachColour(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

''' <summary>

''' 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...

''' </summary>

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

If chararray.Length < 1 Then Return Nothing

Dim number(0 To 5) As Integer

number.Zero()

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

''' <summary> Returns true if all the edges are possible </summary>

Private Function EdgesPossible(ByVal stickers(,) As Char) As Boolean

For faceNumber = 0 To 5

For edgeNumber = 1 To 7 Step 2

Dim stickerColour As Char = stickers(faceNumber, edgeNumber)

Dim oppColour As Char = Opposite(stickerColour).ToString()

Dim adjacentSticker As Sticker = AdjacentEdge(faceNumber, edgeNumber)

Dim adjacentStickerColour As Char = stickers(adjacentSticker.FaceNumber, adjacentSticker.StickerNumber)

If adjacentStickerColour = stickerColour Or adjacentStickerColour = oppColour Then

DisplayEdgeErrorMessage(stickers, {faceNumber, adjacentSticker.FaceNumber})

Return False

End If

Next edgeNumber

Next faceNumber

Return True

End Function

Private Sub DisplayEdgeErrorMessage(ByVal stickers(,) As Char, ByVal wrongFaceColours() As Integer)

Dim wrongFace1, wrongFace2 As String

wrongFace1 = ColourCharToWord(stickers(wrongFaceColours(0), 4)).ToLower()

wrongFace2 = ColourCharToWord(stickers(wrongFaceColours(1), 4)).ToLower()

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

entered it correctly?")

End Sub

''' <summary> Returns true if all the corners are possible </summary>

Private Function CornersPossible(ByVal stickers(,) As Char) As Boolean

For faceNumber = 0 To 5

For cornerNumber = 0 To 8 Step 2

If cornerNumber = MIDDLE\_STICKER Then Continue For

Dim otherTwoStickers() As Sticker = AdjacentCorners(faceNumber, cornerNumber)

'3 stickers per corner

Dim sticker0Colour As Char = stickers(faceNumber, cornerNumber)

Dim sticker1Colour As Char = stickers(otherTwoStickers(0).FaceNumber, otherTwoStickers(0).StickerNumber)

Dim sticker2Colour As Char = stickers(otherTwoStickers(1).FaceNumber, otherTwoStickers(1).StickerNumber)

If Not CornerPossible({sticker0Colour, sticker1Colour, sticker2Colour}) Then

DisplayCornerErrorMessage(stickers,

{faceNumber, otherTwoStickers(0).FaceNumber, otherTwoStickers(1).FaceNumber})

Return False

End If

Next cornerNumber

Next faceNumber

Return True

End Function

Private Function CornerPossible(ByVal cornerColours() As Char) As Boolean

Dim oppColour As Char = Opposite(cornerColours(0)).ToString()

'if any pair of adjacent corner stickers are the same or opposite colours, return false

Return Not (cornerColours(1) = cornerColours(0) Or

cornerColours(2) = cornerColours(0) Or

cornerColours(1) = oppColour Or

cornerColours(2) = oppColour Or

cornerColours(1) = cornerColours(2))

End Function

Private Sub DisplayCornerErrorMessage(ByVal stickers(,) As Char, ByVal wrongFaces() As Integer)

Dim wrongFace1, wrongFace2, wrongFace3 As String

wrongFace1 = ColourCharToWord(stickers(wrongFaces(0), 4)).ToLower()

wrongFace2 = ColourCharToWord(stickers(wrongFaces(1), 4)).ToLower()

wrongFace3 = ColourCharToWord(stickers(wrongFaces(2), 4)).ToLower()

MsgBox("The cube on the " & wrongFace1 & "/" & wrongFace2 & "/" & wrongFace3 & " corner can't be that.

Are you sure you entered it correctly?")

End Sub

''' <summary> Returns true if all the corners have their stickers in the right order </summary>

Private Function CornerRotationsPossible(ByVal stickers(,) As Char) As Boolean

For faceNumber = 0 To 5

For cornerNumber = 0 To 8 Step 2

If cornerNumber = MIDDLE\_STICKER Then Continue For

Dim cornerStickers() As Sticker = AdjacentCorners(faceNumber, cornerNumber)

Dim firstStickerColour As Char = stickers(faceNumber, cornerNumber)

Dim secondStickerColour As Char = stickers(cornerStickers(0).FaceNumber, cornerStickers(0).StickerNumber)

Dim thirdStickerColour As Char = stickers(cornerStickers(1).FaceNumber, cornerStickers(1).StickerNumber)

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

If thirdStickerColour <> getCorrectThirdCornerStickerColour(firstStickerColour, secondStickerColour) Then

DisplayCornerErrorMessage(stickers,

{faceNumber, cornerStickers(0).FaceNumber, cornerStickers(1).FaceNumber})

Return False

End If

Next cornerNumber

Next faceNumber

Return True

End Function

Private Function getCorrectThirdCornerStickerColour(ByVal firstStickerColour As Char, ByVal secondStickerColour As Char) As Char

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

Return correctThirdStickerColour

End Function

'''<summary> Returns the 2nd sticker that makes up an edge piece </summary>

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

'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 New Sticker(x, y)

End Function

'''<summary> Returns the 2nd and 3rd stickers that makes up a corner piece </summary>

Public Function AdjacentCorners(ByVal a As Integer, ByVal b As Integer) As Sticker()

'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 {New Sticker(w, x), New Sticker(y, z)}

End Function

End Module

## CoordinatePairs.vb

Public Module CoordinatePairs

Public Class Sticker

Public Property FaceNumber As Integer

Public Property StickerNumber As Integer

Public Sub New()

FaceNumber = 0

StickerNumber = 0

End Sub

Public Sub New(ByVal x As Integer, ByVal y As Integer)

Me.FaceNumber = x

Me.StickerNumber = y

End Sub

Public Sub New(ByVal button As Button)

Me.FaceNumber = Mid(button.Name, 2, 1) - 1 'e.g. a5 -> 4

Dim stickerLetter As String = Mid(button.Name, 1, 1) 'e.g. a5 -> a

Me.StickerNumber = Asc(stickerLetter) - 97

End Sub

Public Shared Operator =(ByVal sticker1 As Sticker, ByVal sticker2 As Sticker) As Boolean

Return sticker1.FaceNumber = sticker2.FaceNumber And sticker1.StickerNumber = sticker2.StickerNumber

End Operator

Public Shared Operator <>(ByVal sticker1 As Sticker, ByVal sticker2 As Sticker) As Boolean

Return sticker1.FaceNumber <> sticker2.FaceNumber Or sticker1.StickerNumber <> sticker2.StickerNumber

End Operator

End Class

' A collection of 3 stickers that make up a corner cubie

Public Class CornerTriplet

Private ReadOnly cornerTriples(,) As Sticker = {{New Sticker, New Sticker, New Sticker},

{New Sticker, New Sticker, New Sticker},

{New Sticker, New Sticker, New Sticker},

{New Sticker, New Sticker, New Sticker}}

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

Public Property Corners(ByVal index As Integer) As Sticker

Get

Return \_corners(index)

End Get

Set(value As Sticker)

\_corners(index) = value

End Set

End Property

Public Sub New(ByVal stickerOne As Sticker)

If cornerTriples(0, 2).FaceNumber = 0 Then SetUpCornerTriplets()

\_corners(0) = stickerOne

Dim pairIndex, argumentCornerIndex As Integer

' Gets the index for the triple containing the sticker passed as an argument

For j = 0 To UBound(cornerTriples, 2)

For i = 0 To UBound(cornerTriples, 1)

If cornerTriples(i, j) <> stickerOne Then Continue For

pairIndex = i

argumentCornerIndex = j

Next

Next

' sets stickers 2 and 3

Dim index As Integer = 1

For i = 0 To UBound(cornerTriples, 2)

If i = argumentCornerIndex Then Continue For

\_corners(index) = cornerTriples(pairIndex, i)

index = 2

Next

End Sub

Private Sub SetUpCornerTriplets()

For i = 0 To 3

cornerTriples(i, 0).FaceNumber = 0

cornerTriples(i, 1).StickerNumber = 0

cornerTriples(i, 2).StickerNumber = 2

Next

cornerTriples(0, 0).StickerNumber = 0 : cornerTriples(0, 1).FaceNumber = 1 : cornerTriples(0, 2).FaceNumber = 2

cornerTriples(1, 0).StickerNumber = 2 : cornerTriples(1, 1).FaceNumber = 2 : cornerTriples(1, 2).FaceNumber = 3

cornerTriples(2, 0).StickerNumber = 6 : cornerTriples(2, 1).FaceNumber = 4 : cornerTriples(2, 2).FaceNumber = 1

cornerTriples(3, 0).StickerNumber = 8 : cornerTriples(3, 1).FaceNumber = 3 : cornerTriples(3, 2).FaceNumber = 4

End Sub

End Class

' A collection of 2 stickers that make up an edge cubie

Public Class EdgePair

Private ReadOnly edgePairs(,) As Sticker = {{New Sticker, New Sticker},

{New Sticker, New Sticker},

{New Sticker, New Sticker},

{New Sticker, New Sticker},

{New Sticker, New Sticker},

{New Sticker, New Sticker},

{New Sticker, New Sticker},

{New Sticker, New Sticker}}

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

Public Property Edges(ByVal index As Integer) As Sticker

Get

Return \_edges(index)

End Get

Set(value As Sticker)

\_edges(index) = value

End Set

End Property

Public Sub New(ByVal stickerOne As Sticker)

If edgePairs(0, 1).StickerNumber = 0 Then SetUpEdgePairs()

\_edges(0) = stickerOne

Dim pairIndex, argumentEdgeIndex As Integer

' Gets the index for the pair containing the sticker passed as an argument

For j = 0 To UBound(edgePairs, 2)

For i = 0 To UBound(edgePairs, 1)

If edgePairs(i, j) <> \_edges(0) Then Continue For

pairIndex = i

argumentEdgeIndex = j

Next

Next

' Sets sticker 2

For i = 0 To UBound(edgePairs, 2)

If i = argumentEdgeIndex Then Continue For

\_edges(1) = edgePairs(pairIndex, i)

Next

End Sub

Private Sub SetUpEdgePairs()

For i = 0 To 3

edgePairs(i, 0).FaceNumber = 0

edgePairs(i, 1).StickerNumber = 1

Next

For i = 4 To 7

edgePairs(i, 0).StickerNumber = 3

edgePairs(i, 1).StickerNumber = 5

Next

edgePairs(0, 0).StickerNumber = 1 : edgePairs(0, 1).FaceNumber = 2

edgePairs(1, 0).StickerNumber = 3 : edgePairs(1, 1).FaceNumber = 1

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

edgePairs(3, 0).StickerNumber = 7 : edgePairs(3, 1).FaceNumber = 4

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

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

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

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

End Sub

End Class

End Module

## SetCubieProperties.vb

Imports RubiksCubeSolver\_v2\_0.Helpers.PublicFunctions

Imports RubiksCubeSolver\_v2\_0.Helpers.PublicConstants

Public Module SetCubieProperties

''' <summary>

''' Returns the colourString for a cornerTriplet, and sets secondaryFaceColour and secondaryFaceRotation

''' </summary>

Public Function GetCornerColours(ByVal cube(,) As Char, ByVal cornerTri As CornerTriplet, ByRef secondaryFaceColour As FaceColour, ByRef secondaryFaceRotation As FaceColour) As String

Dim cornerColours(2) As Char

For cornerBlockSticker = 0 To 2

Dim colourStore As Char

colourStore = cube(cornerTri.Corners(cornerBlockSticker).FaceNumber,

cornerTri.Corners(cornerBlockSticker).StickerNumber)

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

cornerColours(0) = colourStore

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

secondaryFaceColour = ColourChar2FaceNumber(colourStore)

secondaryFaceRotation = ColourChar2FaceNumber(cube(cornerTri.Corners(cornerBlockSticker).FaceNumber, 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

Dim cornerString As String

cornerString = ""

For k = 0 To 2

cornerString += cornerColours(k)

Next k

Return cornerString

End Function

''' <summary> Returns the colourString for a edgePair </summary>

Public Function GetEdgeColours(ByVal cube(,) As Char, ByVal edgePair As EdgePair) As String

Dim edgeString As String = ""

Dim edgeColours(1) As Char

Dim colourStore, colourStore2 As Char

colourStore = cube(edgePair.Edges(0).FaceNumber, edgePair.Edges(0).StickerNumber)

colourStore2 = cube(edgePair.Edges(1).FaceNumber, edgePair.Edges(1).StickerNumber)

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

Return edgeString

End Function

''' <summary>

''' Returns the position vector for a corner on the top half of the cube.

''' From the face and location of its primary face

''' </summary>

Public Function GetTopCornerPosition(ByVal face As Integer, ByVal sticker As Integer) As Vector3x1

Dim position As New Vector3x1

position.y = 1

Select Case face

Case MoveFaces.TOP

position.x = (sticker Mod 6) - 1

position.z = Math.Sign(sticker - 4)

Case MoveFaces.LEFT

position.x = -1

position.z = 1 - Math.Abs(sticker - 2)

Case MoveFaces.BACK

position.x = Math.Abs(sticker - 2) - 1

position.z = -1

Case MoveFaces.RIGHT

position.x = 1

position.z = 1 - sticker

Case MoveFaces.FRONT

position.x = sticker - 1

position.z = 1

End Select

Return position

End Function

''' <summary>

''' Returns the position vector for a corner on the bottom half of the cube.

''' From the face and location of its primary face

''' </summary>

Public Function GetBottomCornerPosition(ByVal face As Integer, ByVal sticker As Integer) As Vector3x1

Dim position As New Vector3x1

position.y = -1

Select Case face

Case MoveFaces.TOP

position.x = Math.Abs((sticker Mod 6) - 2) - 1

position.z = Math.Sign(sticker - 4)

Case MoveFaces.LEFT

position.x = 1

position.z = 1 - Math.Abs(sticker - 2)

Case MoveFaces.BACK

position.x = sticker - 1

position.z = -1

Case MoveFaces.RIGHT

position.x = -1

position.z = 1 - sticker

Case MoveFaces.FRONT

position.x = Math.Abs(sticker - 2) - 1

position.z = 1

End Select

Return position

End Function

''' <summary>

''' Returns the position vector for an edge on the top half of the cube.

''' From the face and location of its primary face

''' </summary>

Public Function GetTopEdgePosition(ByVal face As Integer, ByVal sticker As Integer) As Vector3x1

Dim position As New Vector3x1

Select Case face

Case MoveFaces.TOP

position.x = ((sticker Mod 6) Mod 3) - 1

position.y = 1

Select Case sticker

Case 1 : position.z = -1

Case 3 : position.z = 0

Case 5 : position.z = 0

Case 7 : position.z = 1

End Select

Case MoveFaces.LEFT

position.x = -1

position.y = Math.Sign(3 - sticker)

position.z = sticker Mod 3 - 1

Case MoveFaces.BACK

position.x = Math.Sign(sticker - 1)

position.y = Math.Sign(3 - sticker)

position.z = -1

Case MoveFaces.RIGHT

position.x = 1

position.y = Math.Sign(3 - sticker)

position.z = Math.Sign(sticker - 1)

Case MoveFaces.FRONT

position.x = sticker Mod 3 - 1

position.y = Math.Sign(3 - sticker)

position.z = 1

End Select

Return position

End Function

''' <summary>

''' Returns the position vector for an edge on the bottom half of the cube.

''' From the face and location of its primary face

''' </summary>

Public Function GetBottomEdgePosition(ByVal face As Integer, ByVal sticker As Integer) As Vector3x1

Dim position As New Vector3x1

Select Case face

Case 0 ' Top

position.x = -(((sticker Mod 3) Mod 6) - 1)

position.y = -1

Select Case sticker

Case 1 : position.z = -1

Case 3 : position.z = 0

Case 5 : position.z = 0

Case 7 : position.z = 1

End Select

Case 1 ' Left

position.x = 1

position.y = -(sticker Mod 3)

position.z = (sticker Mod 3) - 1

Case 2 ' Back

position.x = (sticker Mod 3) - 1

position.y = -(sticker Mod 3)

position.z = -1

Case 3 ' Right

position.x = -1

position.y = -(sticker Mod 3)

position.z = Math.Sign(sticker - 1)

Case 4 ' Front

position.x = Math.Sign(sticker - 1)

position.y = -(sticker Mod 3)

position.z = 1

End Select

Return position

End Function

''' <summary>

''' Orientates the stickerArray to match a given cubeOrientation

''' </summary>

Public Function OrientateStickerArray(ByVal stickerArray(,) As Char, ByVal cubeOrientation As CubeOrientation) As Char(,)

Dim transformedArray(4, 8) As Char

' each face can be in 1 of 4 rotations : 4 methods

Try

If cubeOrientation.Front = Opposite(cubeOrientation.Top) Then

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

"Top: " & cubeOrientation.Top & vbNewLine & "Front: " & cubeOrientation.Front)

End If

Catch ex As ArgumentException

MsgBox(ex.Message)

Return Nothing

End Try

Dim faceColours As FaceColour() = GetFaceColoursFromOrientation(cubeOrientation)

For sideFaceNumber = 0 To 4

Dim faceColour As FaceColour = faceColours(sideFaceNumber)

Dim faceStickerStore(9) As Char

For i = 0 To 8

faceStickerStore(i) = stickerArray(faceColour, i)

Next

transformedArray = MapFaceOntoTransformedArray(sideFaceNumber, faceColour, faceStickerStore,

cubeOrientation, transformedArray)

Next

Return transformedArray

End Function

Private Function MapFaceOntoTransformedArray(ByVal sideFaceNumber As Integer, ByVal faceColour As FaceColour, ByVal faceStickerStore() As Char, ByVal cubeOrientation As CubeOrientation, ByVal transformedArray(,) As Char) As Char(,)

Select Case cubeOrientation.Top

Case FaceColour.W

Select Case faceColour

Case FaceColour.O : UpsideDown(transformedArray, sideFaceNumber, faceStickerStore)

Case cubeOrientation.Top : TopFace(transformedArray, sideFaceNumber, faceStickerStore, cubeOrientation)

Case Else : NoRotation(transformedArray, sideFaceNumber, faceStickerStore)

End Select

Case FaceColour.Y

Select Case faceColour

Case FaceColour.O : NoRotation(transformedArray, sideFaceNumber, faceStickerStore)

Case cubeOrientation.Top : TopFace(transformedArray, sideFaceNumber, faceStickerStore, cubeOrientation)

Case Else : UpsideDown(transformedArray, sideFaceNumber, faceStickerStore)

End Select

Case FaceColour.G

Select Case faceColour

Case cubeOrientation.Top : TopFace(transformedArray, sideFaceNumber, faceStickerStore, cubeOrientation)

Case Else : RotateLeft(transformedArray, sideFaceNumber, faceStickerStore)

End Select

Case FaceColour.B

Select Case faceColour

Case cubeOrientation.Top : TopFace(transformedArray, sideFaceNumber, faceStickerStore, cubeOrientation)

Case Else : RotateRight(transformedArray, sideFaceNumber, faceStickerStore)

End Select

Case FaceColour.R

Select Case faceColour

Case FaceColour.Y : NoRotation(transformedArray, sideFaceNumber, faceStickerStore)

Case FaceColour.W : UpsideDown(transformedArray, sideFaceNumber, faceStickerStore)

Case FaceColour.B : RotateLeft(transformedArray, sideFaceNumber, faceStickerStore)

Case FaceColour.G : RotateRight(transformedArray, sideFaceNumber, faceStickerStore)

Case cubeOrientation.Top : TopFace(transformedArray, sideFaceNumber, faceStickerStore, cubeOrientation)

End Select

Case FaceColour.O

Select Case faceColour

Case FaceColour.Y : UpsideDown(transformedArray, sideFaceNumber, faceStickerStore)

Case FaceColour.W : NoRotation(transformedArray, sideFaceNumber, faceStickerStore)

Case FaceColour.B : RotateRight(transformedArray, sideFaceNumber, faceStickerStore)

Case FaceColour.G : RotateLeft(transformedArray, sideFaceNumber, faceStickerStore)

Case cubeOrientation.Top : TopFace(transformedArray, sideFaceNumber, faceStickerStore, cubeOrientation)

End Select

End Select

Return transformedArray

End Function

Private Sub NoRotation(ByRef stickerArray As Char(,), ByVal faceNumber As Integer, ByVal faceStickers As Char())

For i = 0 To 8

stickerArray(faceNumber, i) = faceStickers(i)

Next

End Sub

Private Sub UpsideDown(ByRef stickerArray As Char(,), ByVal faceNumber As Integer, ByVal faceStickers As Char())

For i = 0 To 8

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

Next

End Sub

Private Sub RotateLeft(ByRef stickerArray As Char(,), ByVal faceNumber As Integer, ByVal faceStickers As Char())

Dim count As Integer = 6

For i = 0 To 8

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

count += 7

Next

End Sub

Private Sub RotateRight(ByRef stickerArray As Char(,), ByVal faceNumber As Integer, ByVal faceStickers As Char())

Dim count As Integer = 2

For i = 0 To 8

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

count += 3

Next

End Sub

Private Sub TopFace(ByRef stickerArray As Char(,), ByVal faceNumber As Integer, ByVal faceStickers As Char(), ByVal currentOrientation As CubeOrientation)

If currentOrientation.Front = FaceColour.B Then

RotateLeft(stickerArray, faceNumber, faceStickers)

ElseIf currentOrientation.Front = FaceColour.G Then

RotateRight(stickerArray, faceNumber, faceStickers)

ElseIf currentOrientation.Front = FaceColour.W Then

If currentOrientation.Top = FaceColour.O Then

NoRotation(stickerArray, faceNumber, faceStickers)

Else

UpsideDown(stickerArray, faceNumber, faceStickers)

End If

ElseIf currentOrientation.Front = FaceColour.Y Then

If currentOrientation.Top = FaceColour.O Then

UpsideDown(stickerArray, faceNumber, faceStickers)

Else

NoRotation(stickerArray, faceNumber, faceStickers)

End If

ElseIf currentOrientation.Front = FaceColour.R Then

Select Case currentOrientation.Top

Case FaceColour.W : NoRotation(stickerArray, faceNumber, faceStickers)

Case FaceColour.Y : UpsideDown(stickerArray, faceNumber, faceStickers)

Case FaceColour.G : RotateLeft(stickerArray, faceNumber, faceStickers)

Case FaceColour.B : RotateRight(stickerArray, faceNumber, faceStickers)

End Select

ElseIf currentOrientation.Front = FaceColour.O Then

Select Case currentOrientation.Top

Case FaceColour.W : UpsideDown(stickerArray, faceNumber, faceStickers)

Case FaceColour.Y : NoRotation(stickerArray, faceNumber, faceStickers)

Case FaceColour.G : RotateRight(stickerArray, faceNumber, faceStickers)

Case FaceColour.B : RotateLeft(stickerArray, faceNumber, faceStickers)

End Select

End If

End Sub

End Module

## Processing.vb

Public Class Processing

Private \_cube As Cube

Public Sub New(ByRef cube As Cube)

InitializeComponent()

Me.\_cube = cube

End Sub

Private Sub Processing\_Shown(sender As Object, e As EventArgs) Handles MyBase.Shown

lblText.Visible = True

lblText.Refresh()

'copy the initial cube before solving it

Dim scrambledCorners() As Corner = Copy(\_cube.Corners)

Dim scrambledEdges() As Edge = Copy(\_cube.Edges)

\_cube.Solve(Me)

Dim outputForm As New \_3DOutput(New Cube(scrambledCorners, scrambledEdges, \_cube.Instructions))

outputForm.Show()

Me.Close()

End Sub

End Class

## Cube.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

Imports System.Runtime.Serialization.Formatters.Binary

<Serializable()> Public Class Cube

Public Property CurrentOrientation As CubeOrientation

Public Property Corners() As Corner()

Public Property Edges() As Edge()

Public Property Middles() As Middle()

Public Property Instructions As InstructionList

Public Property TopFace As FaceColour

Get

Return CurrentOrientation.Top

End Get

Set(value As FaceColour)

CurrentOrientation.Top = value

End Set

End Property

Public ReadOnly Property BottomFace As FaceColour

Get

Return CurrentOrientation.Bottom

End Get

End Property

Public Property FrontFace As FaceColour

Get

Return CurrentOrientation.Front

End Get

Set(value As FaceColour)

CurrentOrientation.Front = value

End Set

End Property

Public ReadOnly Property CornersAndEdgesAndMiddles As Block()

Get

If Me.Corners.Length < 1 Or Me.Edges.Length < 1 Or Me.Middles.Length < 1 Then Return Nothing

Dim \_cornersAndEdgesandmiddles(26) As Block

For i = 0 To 7

\_cornersAndEdgesandmiddles(i) = Corners(i)

Next

For i = 0 To 11

\_cornersAndEdgesandmiddles(i + 8) = Edges(i)

Next

For i = 0 To 6

\_cornersAndEdgesandmiddles(i + 20) = Middles(i)

Next

Return \_cornersAndEdgesandmiddles

End Get

End Property

ReadOnly Property CorrectTopEdges() As Edge()

Get

Dim rightCubies(3) As Edge

Dim index As Integer = 0

Dim topEdges() As Edge = TopEdgesClockwise()

For Each cubie In topEdges

If cubie.CorrectForFace(TopFace, Me) Then

rightCubies(index) = cubie

index += 1

End If

Next

ReDim Preserve rightCubies(index - 1)

Return rightCubies

End Get

End Property

ReadOnly Property IncorrectTopEdges() As Edge()

Get

Dim wrongCubies(3) As Edge

Dim index As Integer = 0

Dim topEdges() As Edge = TopEdgesClockwise()

For Each cubie In topEdges

If Not cubie.CorrectForFace(TopFace, Me) Then

wrongCubies(index) = cubie

index += 1

End If

Next

ReDim Preserve wrongCubies(index - 1)

Return wrongCubies

End Get

End Property

ReadOnly Property CorrectTopCorners() As Corner()

Get

Dim rightCubies(3) As Corner

Dim index As Integer = 0

Dim topEdges() As Corner = TopCornersClockwise()

For Each cubie In topEdges

If cubie.CorrectForFace(TopFace, Me) Then

rightCubies(index) = cubie

index += 1

End If

Next

ReDim Preserve rightCubies(index - 1)

Return rightCubies

End Get

End Property

ReadOnly Property IncorrectTopCorners() As Corner()

Get

Dim wrongCubies(3) As Corner

Dim index As Integer = 0

Dim topEdges() As Corner = TopCornersClockwise()

For Each cubie In topEdges

If Not cubie.CorrectForFace(TopFace, Me) Then

wrongCubies(index) = cubie

index += 1

End If

Next

ReDim Preserve wrongCubies(index - 1)

Return wrongCubies

End Get

End Property

''' <summary>

''' Returns an array of cubies with the correct colour stickers

''' </summary>

Public ReadOnly Property BlocksByColour(ByVal colour As FaceColour) As Block()

Get

Dim cubies(0 To 8) As Block

Dim count As Integer = 0

For Each block In CornersAndEdgesAndMiddles

If Not block.HasColour(colour) Then Continue For

cubies(count) = block

count += 1

Next

Return cubies

End Get

End Property

Public ReadOnly Property TopEdgesClockwise() As Edge()

Get

Return TopBlocksClockwise().Extract(Of Edge)()

End Get

End Property

Public ReadOnly Property TopCornersClockwise() As Corner()

Get

Return TopBlocksClockwise().Extract(Of Corner)()

End Get

End Property

Public ReadOnly Property TopBlocksClockwise() As Block()

Get

Dim topBlocks(8) As Block

For Each block In CornersAndEdgesAndMiddles

If block.Position.y <> 1 Then Continue For

Dim index As Integer

If block.GetType() = GetType(Edge) Then

index = TopEdgePositionToClockwiseIndex(block.Position)

ElseIf block.GetType() = GetType(Corner) Then

index = TopCornerPositionToClockwiseIndex(block.Position)

ElseIf block.GetType() = GetType(Middle) Then

index = 8

End If

topBlocks(index) = block

Next

Return topBlocks

End Get

End Property

''' <summary>

''' Converts a position vector to an index in an 8-element array to store the blocks on the top face

''' </summary>

Private Function TopCornerPositionToClockwiseIndex(pos As Vector3x1) As Integer

Dim index As Integer

Select Case (pos.x + pos.z)

Case -2 'Back left corner

index = 2

Case 0

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

index = 0

ElseIf pos.x = 1 Then ' Back right corner

index = 4

End If

Case 2 ' Front right corner

index = 6

End Select

Return index

End Function

''' <summary>

''' Converts a position vector to an index in an 8-element array to store the blocks on the top face

''' </summary>

Private Function TopEdgePositionToClockwiseIndex(pos As Vector3x1) As Integer

Dim index As Integer

Select Case pos.x

Case -1 ' left edge cubie

index = 1

Case 0

If pos.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

Return index

End Function

Public ReadOnly Property MiddleEdgesClockwise() As Edge()

Get

Dim middleEdges(0 To 3) As Edge

Dim index As Integer

For Each edge In Edges

If edge.Layer <> Layer.MIDDLE Then Continue For

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

Case -2 'Back left edge

index = 1

Case 0

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

index = 0

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

index = 2

End If

Case 2 ' Front right edge

index = 3

End Select

middleEdges(index) = edge

Next

Return middleEdges

End Get

End Property

ReadOnly Property FaceColours As FaceColour()

Get

Return GetFaceColoursFromOrientation(CurrentOrientation)

End Get

End Property

Public Sub New(ByVal cube As Cube)

Me.CurrentOrientation = cube.CurrentOrientation.Copy()

Me.Corners = cube.Corners.Copy()

Me.Edges = cube.Edges.Copy()

CreateMiddles()

Me.Instructions = cube.Instructions.Copy()

End Sub

Public Sub New(ByVal corners() As Corner, ByVal edges() As Edge, ByVal instructions As InstructionList)

Me.Corners = corners

Me.Edges = edges

CreateMiddles()

Me.Instructions = instructions

End Sub

Public Sub New(ByVal stickers(,) As Char)

CurrentOrientation = New CubeOrientation(CChar("W"), CChar("R"))

Instructions = New InstructionList

InitialiseBlocks()

Dim originalData(,) As Char = Copy(stickers)

'Converts top half of cube

Dim transformedData(,) As Char = OrientateStickerArray(originalData, CurrentOrientation)

ConvertTopCorners(transformedData)

ConvertTopEdges(transformedData)

'Converts bottom half of cube

transformedData = OrientateStickerArray(originalData, CurrentOrientation.UpsideDown)

ConvertTopCorners(transformedData)

ConvertTopEdges(transformedData)

CreateMiddles()

End Sub

Private Sub InitialiseBlocks()

FillNameArrays()

ReDim Corners(7)

ReDim Edges(11)

For i = 0 To 7

Me.Corners(i) = New Corner With {.Name = CORNER\_NAMES(i)}

Next

For i = 0 To 11

Me.Edges(i) = New Edge With {.Name = EDGE\_NAMES(i)}

Next

End Sub

Private Sub CreateMiddles()

ReDim Middles(6)

For i = 0 To 6

Me.Middles(i) = New Middle

Next

Middles(0).Name = "W"

Middles(1).Name = "G"

Middles(2).Name = "O"

Middles(3).Name = "B"

Middles(4).Name = "R"

Middles(5).Name = "Y"

Middles(6).Name = ""

For i = 0 To 5

Middles(i).Colours(0) = Helpers.ColourChar2FaceNumber(CChar(Middles(i).Name))

Middles(i).Rotation = Middles(i).PrimaryFace

Next

Middles(0).Position = New Vector3x1(0, 1, 0)

Middles(1).Position = New Vector3x1(-1, 0, 0)

Middles(2).Position = New Vector3x1(0, 0, -1)

Middles(3).Position = New Vector3x1(1, 0, 0)

Middles(4).Position = New Vector3x1(0, 0, 1)

Middles(5).Position = New Vector3x1(0, -1, 0)

Middles(6).Position = New Vector3x1(0, 0, 0)

End Sub

Private Sub ConvertTopCorners(ByVal stickers(,) As Char)

Dim convertingTopLayer As Boolean

convertingTopLayer = (stickers(0, 4) = FaceNumber2ColourChar(CurrentOrientation.Top))

For faceNumber = MoveFaces.TOP To MoveFaces.FRONT

For cornerStickerNumber = 0 To 8 Step 2

Dim cornerSticker As New Sticker(faceNumber, cornerStickerNumber)

Dim stickerColour As Char = stickers(faceNumber, cornerStickerNumber)

If Not (OnTopHalfOfCube(cornerSticker) And IsPrimaryCornerSticker(stickerColour)) Then Continue For

Dim cornerTri As New CornerTriplet(cornerSticker)

Dim secondaryFaceColour, secondaryFaceRotation As FaceColour

'Gets the colour string for the cubie e.g. "YRG" and also gets the secondary face colour and rotation

Dim cornerString As String = GetCornerColours(stickers, cornerTri, secondaryFaceColour,

secondaryFaceRotation)

For cornerNumber = 0 To UBound(CORNER\_NAMES)

If CORNER\_NAMES(cornerNumber) <> cornerString Then Continue For

If convertingTopLayer Then

Corners(cornerNumber).Position = GetTopCornerPosition(faceNumber, cornerStickerNumber)

Else

Corners(cornerNumber).Position = GetBottomCornerPosition(faceNumber, cornerStickerNumber)

End If

Corners(cornerNumber).Rotation = ColourChar2FaceNumber(stickers(cornerTri.Corners(0).FaceNumber, 4))

Corners(cornerNumber).SecondaryFace = secondaryFaceColour

Corners(cornerNumber).SecondaryRotation = secondaryFaceRotation

Corners(cornerNumber).SetColoursFromColourString(cornerString)

Next cornerNumber

Next cornerStickerNumber

Next faceNumber

End Sub

Private Sub ConvertTopEdges(ByVal stickers(,) As Char)

Dim convertingTopLayer As Boolean

convertingTopLayer = (stickers(0, 4) = FaceNumber2ColourChar(CurrentOrientation.Top))

For faceNumber = MoveFaces.TOP To MoveFaces.FRONT

For edgeStickerNumber = 1 To 7 Step 2

Dim edgeSticker As New Sticker(faceNumber, edgeStickerNumber)

Dim stickerColour As Char = stickers(faceNumber, edgeStickerNumber)

Dim edgePair As New EdgePair(edgeSticker)

Dim secondStickerColour As Char = stickers(edgePair.Edges(1).FaceNumber, edgePair.Edges(1).StickerNumber)

If Not (OnTopHalfOfCube(edgeSticker) And

IsPrimaryEdgeSticker(stickerColour, secondStickerColour)) Then Continue For 'try next sticker

Dim edgeString As String = GetEdgeColours(stickers, edgePair)

For edgeNumber = 0 To UBound(EDGE\_NAMES)

If EDGE\_NAMES(edgeNumber) <> edgeString Then Continue For

If convertingTopLayer Then

Edges(edgeNumber).Position = GetTopEdgePosition(faceNumber, edgeStickerNumber)

Else

Edges(edgeNumber).Position = GetBottomEdgePosition(faceNumber, edgeStickerNumber)

End If

Edges(edgeNumber).Rotation = ColourChar2FaceNumber(stickers(edgePair.Edges(0).FaceNumber, 4))

Edges(edgeNumber).SetColoursFromColourString(edgeString)

Next edgeNumber

Next edgeStickerNumber

Next faceNumber

End Sub

Private Function OnTopHalfOfCube(ByVal position As Sticker) As Boolean

Return position.StickerNumber <> MIDDLE\_STICKER And

(position.FaceNumber = MoveFaces.TOP Or position.StickerNumber < MIDDLE\_STICKER)

End Function

Private Function IsPrimaryCornerSticker(ByVal stickerColour As Char) As Boolean

Return stickerColour = "W" Or stickerColour = "Y"

End Function

Private Function IsPrimaryEdgeSticker(ByVal stickerColour As Char, ByVal secondStickerColour As Char) As Boolean

Return ((stickerColour = "W" Or stickerColour = "Y") Or

((stickerColour = "R" Or stickerColour = "O") And (secondStickerColour <> "W" And secondStickerColour <> "Y")))

End Function

Public Sub Solve(ByRef displayForm As Processing)

If Complete() Then Return

Try

displayForm.lblStatus.Text = "Solving Top Layer"

displayForm.lblStatus.Refresh()

SolveTopLayer()

displayForm.lblStatus.Text = "Solving Middle Layer"

displayForm.lblStatus.Refresh()

SolveMiddleLayer()

displayForm.lblStatus.Text = "Solving Bottom Layer"

displayForm.lblStatus.Refresh()

SolveBottomLayer()

If Not Complete() Then Throw New StageNotSuccessfulException()

Catch ex As StageNotSuccessfulException

Console.WriteLine(ex.Message)

MsgBox("Your cube cannot be solved, this is likely because the stickers have been re-arranged, or one of the pieces has been removed and re-inserted in the wrong orientation. You will need to take the cube apart and re-assemble it in its solved state. If you want you can go throught the instructions the program generated, which will not solve it but may bring it closer to being solved.")

End Try

displayForm.lblStatus.Text = "Optimising Steps For Solving"

displayForm.lblStatus.Refresh()

Instructions.Optimise()

End Sub

Private Sub SolveTopLayer()

Dim topLayerSolver As New TopLayerSolver(Me)

topLayerSolver.Solve()

End Sub

Private Sub SolveMiddleLayer()

Dim middleLayerSolver As New MiddleLayerSolver(Me)

middleLayerSolver.Solve()

End Sub

Private Sub SolveBottomLayer()

Dim bottomLayerSolver As New BottomLayerSolver(Me)

bottomLayerSolver.Solve()

End Sub

Public ReadOnly Property Complete() As Boolean

Get

For face As FaceColour = Helpers.FaceColour.W To Helpers.FaceColour.O

Dim faceBlocks As Block() = BlocksByColour(face)

For Each block In faceBlocks

If Not block.Correct(Me) Then Return False

Next

Next

Return True

End Get

End Property

''' <summary> Rotates the whole cube </summary>

Public Sub Rotate(ByVal direction As Direction, ByVal axis As Axis)

If direction = Direction.NO\_CHANGE Then Return

'used for undoing half-turns when ouputting

If direction = -2 Then direction = Direction.HALF\_TURN

Dim angle As Double = (Math.PI / 2.0) \* direction

Dim rotationMatrix As Matrix3x3 = GetRotationMatrix(axis, angle)

Instructions.AddOrientationChange(axis, direction)

For Each cubie In CornersAndEdgesAndMiddles

cubie.Position = rotationMatrix \* cubie.Position

Next

UpdateOrientation(direction, axis)

End Sub

Private Sub UpdateOrientation(ByVal direction As Direction, ByVal axis As Axis)

Dim faces() As FaceColour = FacesAroundAxisClockwise(axis)

If axis <> Axis.Y Then TopFace = faces((Array.IndexOf(faces, TopFace) + 4 + direction) Mod 4)

If axis <> Axis.Z Then FrontFace = faces((Array.IndexOf(faces, FrontFace) + 4 + direction) Mod 4)

End Sub

Private Function FacesAroundAxisClockwise(ByVal axis As Axis) As FaceColour()

Select Case axis

Case Axis.X

Return {FaceColours(MoveFaces.FRONT), FaceColours(MoveFaces.TOP), FaceColours(MoveFaces.BACK),

FaceColours(MoveFaces.BOTTOM)}

Case Axis.Y

Return {FaceColours(MoveFaces.FRONT), FaceColours(MoveFaces.LEFT), FaceColours(MoveFaces.BACK),

FaceColours(MoveFaces.RIGHT)}

Case Axis.Z

Return {FaceColours(MoveFaces.TOP), FaceColours(MoveFaces.RIGHT), FaceColours(MoveFaces.BOTTOM),

FaceColours(MoveFaces.LEFT)}

Case Else

Throw New ArgumentException("Invalid Axis")

End Select

End Function

''' <summary> Rotates the cube so that a specific face is on top </summary>

Public Sub RotateFaceToTop(ByVal newTop As Helpers.FaceColour)

Dim faces() As FaceColour = FaceColours

Select Case newTop

Case faces(MoveFaces.TOP)

Return

Case faces(MoveFaces.LEFT)

Me.Rotate(Direction.CLOCKWISE, Axis.Z)

Case faces(MoveFaces.BACK)

Me.Rotate(Direction.ANTICLOCKWISE, Axis.X)

FrontFace = faces(MoveFaces.TOP)

Case faces(MoveFaces.RIGHT)

Me.Rotate(Direction.ANTICLOCKWISE, Axis.Z)

Case faces(MoveFaces.FRONT)

Me.Rotate(Direction.CLOCKWISE, Axis.X)

FrontFace = faces(MoveFaces.BOTTOM)

Case faces(MoveFaces.BOTTOM)

Me.Rotate(Direction.HALF\_TURN, Axis.Z)

End Select

End Sub

''' <summary> Rotates the cube so that a specific face is at the front </summary>

Public Sub RotateFaceToFront(ByVal newFront As Helpers.FaceColour)

Dim faces() As FaceColour = FaceColours

Select Case newFront

Case faces(MoveFaces.TOP)

Me.Rotate(Direction.ANTICLOCKWISE, Axis.X)

TopFace = faces(MoveFaces.BACK)

Case faces(MoveFaces.LEFT)

Me.Rotate(Direction.ANTICLOCKWISE, Axis.Y)

Case faces(MoveFaces.BACK)

Me.Rotate(Direction.HALF\_TURN, Axis.Y)

Case faces(MoveFaces.RIGHT)

Me.Rotate(Direction.CLOCKWISE, Axis.Y)

Case faces(MoveFaces.FRONT)

Return

Case faces(MoveFaces.BOTTOM)

Me.Rotate(Direction.CLOCKWISE, Axis.X)

TopFace = faces(MoveFaces.FRONT)

End Select

End Sub

''' <summary> Rotates the cube so that a specific face is on the left </summary>

Public Sub RotateFaceToLeft(ByVal newLeft As Helpers.FaceColour)

Dim faces() As FaceColour = FaceColours

Select Case newLeft

Case faces(MoveFaces.TOP)

Me.Rotate(Direction.ANTICLOCKWISE, Axis.Z)

Case faces(MoveFaces.LEFT)

Return

Case faces(MoveFaces.BACK)

Me.Rotate(Direction.ANTICLOCKWISE, Axis.Y)

Case faces(MoveFaces.RIGHT)

Me.Rotate(Direction.HALF\_TURN, Axis.Y)

Case faces(MoveFaces.FRONT)

Me.Rotate(Direction.CLOCKWISE, Axis.Y)

Case faces(MoveFaces.BOTTOM)

Me.Rotate(Direction.CLOCKWISE, Axis.Z)

End Select

End Sub

Public Sub RotateFace(ByVal face As FaceColour, ByVal direction As Direction)

If direction = Direction.NO\_CHANGE Then Return

If direction = -2 Then direction = Direction.HALF\_TURN

Dim mathematicalDirection As Direction = direction

Dim axis As Axis

Select Case face

Case FaceColours(MoveFaces.TOP)

axis = Axis.Y

Case FaceColours(MoveFaces.LEFT)

axis = Axis.X

If direction <> Direction.HALF\_TURN Then mathematicalDirection = -mathematicalDirection

Case FaceColours(MoveFaces.BACK)

axis = Axis.Z

If direction <> Direction.HALF\_TURN Then mathematicalDirection = -mathematicalDirection

Case FaceColours(MoveFaces.RIGHT)

axis = Axis.X

Case FaceColours(MoveFaces.FRONT)

axis = Axis.Z

Case FaceColours(MoveFaces.BOTTOM)

axis = Axis.Y

If direction <> Direction.HALF\_TURN Then mathematicalDirection = -mathematicalDirection

End Select

Dim angle As Double = (Math.PI / 2.0) \* mathematicalDirection

Dim rotationMatrix As Matrix3x3 = GetRotationMatrix(axis, angle)

Instructions.AddFaceTurn(Array.IndexOf(FaceColours, face), direction)

Dim faceColoursAsIfRotatedFaceIsTop() As FaceColour

If face <> FrontFace And face <> Opposite(FrontFace) Then

faceColoursAsIfRotatedFaceIsTop = Helpers.GetFaceColoursFromOrientation(New CubeOrientation(face, FrontFace))

ElseIf face = FrontFace Then

faceColoursAsIfRotatedFaceIsTop = Helpers.GetFaceColoursFromOrientation(New CubeOrientation(face, BottomFace))

ElseIf face = Opposite(FrontFace) Then

faceColoursAsIfRotatedFaceIsTop = Helpers.GetFaceColoursFromOrientation(New CubeOrientation(face, TopFace))

End If

For Each cubie In CornersAndEdgesAndMiddles

If Not cubie.OnFace(face, Me) Then Continue For

cubie.Position = rotationMatrix \* cubie.Position

cubie.Rotation = GetNewRotation(direction, cubie.Rotation, faceColoursAsIfRotatedFaceIsTop)

If cubie.GetType() = GetType(Corner) Then DirectCast(cubie, Corner).SecondaryRotation =

GetNewRotation(direction, DirectCast(cubie, Corner).SecondaryRotation, faceColoursAsIfRotatedFaceIsTop)

Next

End Sub

''' <summary> Gets the new rotation for a cubie from a given rotation </summary>

Private Function GetNewRotation(ByVal direction As Direction, ByVal currentRotation As FaceColour, ByVal faceColoursAsIfRotatedFaceIsTop() As FaceColour) As FaceColour

Dim currentFaceIndex As Integer = Array.IndexOf(faceColoursAsIfRotatedFaceIsTop, currentRotation)

If currentFaceIndex = 0 Then Return currentRotation

If currentFaceIndex = 5 Then Throw New ArgumentException("The cube is not on the face being rotated")

Select Case direction

Case Direction.CLOCKWISE

Return faceColoursAsIfRotatedFaceIsTop((currentFaceIndex Mod 4) + 1)

Case Direction.ANTICLOCKWISE

Return faceColoursAsIfRotatedFaceIsTop(((currentFaceIndex + 2) Mod 4) + 1)

Case Direction.HALF\_TURN

Return faceColoursAsIfRotatedFaceIsTop(((currentFaceIndex + 1) Mod 4) + 1)

Case Else

Return currentRotation

End Select

End Function

Public Sub Save(ByVal filePath As String)

If filePath.Substring(filePath.Length - 5) <> ".cube" Then filePath += ".cube"

Try

Dim fStream As New IO.FileStream(filePath, IO.FileMode.Create)

WriteCubeToFile(fStream)

fStream.Close()

Catch ex As Exception

MsgBox("Unable to save file, please try a different fileName" & vbNewLine & ex.Message)

Throw New WriteUnsuccessfulException

End Try

End Sub

Private Sub WriteCubeToFile(ByRef fStream As IO.FileStream)

Dim formatter As New BinaryFormatter()

formatter.Serialize(fStream, Me)

End Sub

End Class

## Helpers.vb

Namespace Helpers

Public Module PublicConstants

Public Const STICKER\_LETTERS\_NO\_MIDDLE As String = "abcdfghi"

Public Const STICKER\_LETTERS As String = "abcdefghi"

Public Const FACE\_COLOURS As String = "bgorwy"

Public Const MIDDLE\_STICKER As Integer = 4

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

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

Public Sub FillNameArrays()

FillCornerNames()

FillEdgeNames()

End Sub

Private Sub FillCornerNames()

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 FillEdgeNames()

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

Public Enum FaceColour

W

G

R

B

Y

O

None

End Enum

Public Enum Axis

X

Y

Z

End Enum

Public Enum MoveFaces

TOP

LEFT

BACK

RIGHT

FRONT

BOTTOM

End Enum

Public Enum Direction

CLOCKWISE = -1

NO\_CHANGE = 0

ANTICLOCKWISE = 1

HALF\_TURN = 2

End Enum

Public Enum Layer

TOP = 1

MIDDLE = 0

BOTTOM = -1

End Enum

End Module

Module PublicFunctions

''' <summary> Converts colour char to the name of the colour </summary>

Public Function ColourCharToWord(ByVal colourchar As Char) As String

Static colours As New Dictionary(Of Char, String)

If colours.Count = 0 Then colours = GetColourDictionary()

Try

Return colours.Item(colourchar)

Catch ex As KeyNotFoundException

Return "Error"

End Try

End Function

Private Function GetColourDictionary() As Dictionary(Of Char, String)

'uses a dictionary of character keys to colour strings for faster lookup

Dim colours As New Dictionary(Of Char, String) From {

{"W", "White"},

{"R", "Red"},

{"B", "Blue"},

{"Y", "Yellow"},

{"O", "Orange"},

{"G", "Green"}

}

Return colours

End Function

''' <summary> Converts colour char to face number </summary>

Public Function ColourChar2FaceNumber(ByVal colourchar As Char) As FaceColour

Return [Enum].Parse(GetType(FaceColour), colourchar, True)

End Function

Public Function ColourChar2FaceNumber(ByVal colourChars() As Char) As FaceColour()

Dim array(colourChars.Length - 1) As FaceColour

For i = 0 To colourChars.Length - 1

array(i) = ColourChar2FaceNumber(colourChars(i))

Next

Return array

End Function

''' <summary> Converts face number to colour char </summary>

Public Function FaceNumber2ColourChar(ByVal number As Integer) As Char

Dim color As FaceColour = number

Return color.ToString()

End Function

Public Sub WriteBlocksToFile(ByVal blocks As Block(), Optional filename As String = "Blocks.txt")

FileOpen(1, filename, OpenMode.Output)

For Each block In blocks

PrintLine(1, block.ToString())

Next

FileClose(1)

End Sub

''' <summary> Returns the opposte coloured face to the colour given as an argument </summary>

Public Function Opposite(ByVal colour As Char) As Char

Select Case colour

Case "W"

Return "Y"

Case "R"

Return "O"

Case "B"

Return "G"

Case "Y"

Return "W"

Case "O"

Return "R"

Case "G"

Return "B"

Case Else

Return "-"

End Select

End Function

Public Function Opposite(ByVal colour As FaceColour) As FaceColour

Return ColourChar2FaceNumber(Opposite(FaceNumber2ColourChar(colour)))

End Function

' (0 = top, 1 = left, 2 = back, 3 = right, 4 = front 5 = bottom)

Public Function GetFaceColoursFromOrientation(ByVal currentOrientation As CubeOrientation) As FaceColour()

Dim faces(0 To 5) As FaceColour

faces(0) = currentOrientation.Top

faces(5) = currentOrientation.Bottom

Dim sideFaceColoursClockwise() As FaceColour = GetSideFaceColours(currentOrientation.Top)

Dim positionOfFront As Integer

positionOfFront = LinearSearch(sideFaceColoursClockwise, currentOrientation.Front)

For i = 1 To 4

faces(i) = sideFaceColoursClockwise((i + positionOfFront) Mod 4)

Next

Return faces

End Function

Private Function GetSideFaceColours(ByVal top As FaceColour) As FaceColour()

Dim sideFaceColoursClockwise(3) As FaceColour

Select Case top

Case FaceColour.W

sideFaceColoursClockwise(0) = FaceColour.R

sideFaceColoursClockwise(1) = FaceColour.G

sideFaceColoursClockwise(2) = FaceColour.O

sideFaceColoursClockwise(3) = FaceColour.B

Case FaceColour.Y

sideFaceColoursClockwise(3) = FaceColour.R

sideFaceColoursClockwise(2) = FaceColour.G

sideFaceColoursClockwise(1) = FaceColour.O

sideFaceColoursClockwise(0) = FaceColour.B

Case FaceColour.R

sideFaceColoursClockwise(0) = FaceColour.B

sideFaceColoursClockwise(1) = FaceColour.Y

sideFaceColoursClockwise(2) = FaceColour.G

sideFaceColoursClockwise(3) = FaceColour.W

Case FaceColour.O

sideFaceColoursClockwise(3) = FaceColour.B

sideFaceColoursClockwise(2) = FaceColour.Y

sideFaceColoursClockwise(1) = FaceColour.G

sideFaceColoursClockwise(0) = FaceColour.W

Case FaceColour.B

sideFaceColoursClockwise(0) = FaceColour.Y

sideFaceColoursClockwise(1) = FaceColour.R

sideFaceColoursClockwise(2) = FaceColour.W

sideFaceColoursClockwise(3) = FaceColour.O

Case FaceColour.G

sideFaceColoursClockwise(3) = FaceColour.Y

sideFaceColoursClockwise(2) = FaceColour.R

sideFaceColoursClockwise(1) = FaceColour.W

sideFaceColoursClockwise(0) = FaceColour.O

End Select

Return sideFaceColoursClockwise

End Function

End Module End Namespace

## Block.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

<Serializable()> 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

Public ReadOnly Property Layer As Layer

Get

Return Position.y

End Get

End Property

''' <summary> The colour of the face that the primary face is on </summary>

Private \_rotation As FaceColour

Public Property Rotation() As FaceColour

Get

Return \_rotation

End Get

Set(value As FaceColour)

\_rotation = value

End Set

End Property

Private \_colours() As FaceColour = {FaceColour.None, FaceColour.None, FaceColour.None}

Public Property Colours(ByVal index As Integer) As FaceColour

Get

Return \_colours(index)

End Get

Set(value As FaceColour)

\_colours(index) = value

End Set

End Property

Public Property Colours() As FaceColour()

Get

Return \_colours

End Get

Set(value As FaceColour())

\_colours = value

End Set

End Property

Public MustOverride Sub SetColoursFromColourString(ByVal colourString As String)

Public ReadOnly Property PrimaryFace() As FaceColour

Get

Return \_colours(0)

End Get

End Property

Public MustOverride Overrides Function ToString() As String

Public MustOverride Function HasColour(ByVal colour As FaceColour) As Boolean

Public Function OnFace(ByVal face As FaceColour, ByVal cube As Cube) As Boolean

Dim faces As FaceColour() = cube.FaceColours

Return face = faces(MoveFaces.TOP) And Position.y = 1 Or

face = faces(MoveFaces.BOTTOM) And Position.y = -1 Or

face = faces(MoveFaces.FRONT) And Position.z = 1 Or

face = faces(MoveFaces.BACK) And Position.z = -1 Or

face = faces(MoveFaces.LEFT) And Position.x = -1 Or

face = faces(MoveFaces.RIGHT) And Position.x = 1

End Function

''' <summary>

''' Checks if the orientation of a given face of a cube matches a given face colour

''' </summary>

Public MustOverride Function CorrectRotation(ByVal cubieFaceColour As FaceColour, ByVal cubeFaceColour As FaceColour) As Boolean

''' <summary>

''' Checks if a cubie is correct for the given face

''' e.g. if face = blue, checks that the cubie is on the blue face,

''' and that the blue side of a blue cubie is on the blue face of the cube

''' </summary>

Public Overridable Function CorrectForFace(ByVal face As FaceColour, ByVal cube As Cube) As Boolean

Return OnFace(face, cube) And CorrectRotation(face, face)

End Function

''' <summary>

''' Checks if an edge is 'correctly' on the opposite face to what it should be

''' (e.g. if edgeFaceColour = White, it checks if the edge's white face is on the yellow face

''' </summary>

Public Overridable Function CorrectlyOnOppositeFace(ByVal cubieFaceColour As FaceColour, ByVal cube As Cube) As Boolean

Dim cubeFaceColour As FaceColour = Opposite(cubieFaceColour)

Return OnFace(cubeFaceColour, cube) And CorrectRotation(cubieFaceColour, cubeFaceColour)

End Function

''' <summary>

''' Checks if a cubie is in its correct position and orientation

''' </summary>

Public MustOverride Function Correct(ByVal cube As Cube) As Boolean

End Class

## Corner.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

<Serializable()> Public Class Corner

Inherits Block

Private \_secondaryFace As FaceColour

Public Property SecondaryFace() As FaceColour

Get

Return \_secondaryFace

End Get

Set(value As FaceColour)

\_secondaryFace = value

End Set

End Property

Private \_secondaryFaceRotation As FaceColour

Public Property SecondaryRotation() As FaceColour

Get

Return \_secondaryFaceRotation

End Get

Set(value As FaceColour)

\_secondaryFaceRotation = value

End Set

End Property

Public ReadOnly Property SideColours(ByVal topColour As FaceColour) As FaceColour()

Get

If Not HasColour(topColour) Then Throw New ArgumentException("corner does not contain topcolour")

Dim sideCols(1) As FaceColour

Dim index As Integer = 0

For Each colour In Colours

If colour = topColour Then Continue For

sideCols(index) = colour

index += 1

Next

Return sideCols

Throw New Exception("edge does not contain the requested colour")

End Get

End Property

Public Overrides Function ToString() As String

Return Name + ", [" + Colours(0).ToString() + ", " + Colours(1).ToString() + ", " + Colours(2).ToString() + "], " + Position.ToString + ", 1st:" + PrimaryFace.ToString() + "-->" + Rotation.ToString() + ", 2nd:" + SecondaryFace.ToString() + "-->" + SecondaryRotation.ToString()

End Function

Public Overrides Function HasColour(colour As FaceColour) As Boolean

For col = 0 To 2

If Colours(col) = colour Then Return True

Next

Return False

End Function

Public Overrides Function CorrectRotation(cubieFaceColour As FaceColour, cubeFaceColour As FaceColour) As Boolean

If Not HasColour(cubieFaceColour) Then Return False

Select Case cubieFaceColour

Case FaceColour.W, FaceColour.Y

If Rotation <> cubeFaceColour Then Return False

Case FaceColour.R, FaceColour.O

If Rotation = cubeFaceColour Or SecondaryRotation <> cubeFaceColour Then Return False

Case FaceColour.B, FaceColour.G

If Rotation = cubeFaceColour Or SecondaryRotation = cubeFaceColour Then Return False

End Select

Return True

End Function

Public Overrides Function Correct(cube As Cube) As Boolean

Return OnFace(Colours(0), cube) And OnFace(Colours(1), cube) And OnFace(Colours(2), cube) And

CorrectRotation(Colours(0), Colours(0)) And CorrectRotation(Colours(1), Colours(1)) And CorrectRotation(Colours(2), Colours(2))

End Function

Public Overrides Sub SetColoursFromColourString(colourString As String)

For i = 0 To 2

Colours(i) = ColourChar2FaceNumber(colourString(i))

Next

End Sub

End Class

## Edge.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

<Serializable()> Public Class Edge

Inherits Block

Public Sub New()

ReDim Preserve Colours(1)

End Sub

Public Overrides Function ToString() As String

Return Name + ", [" + Colours(0).ToString() + ", " + Colours(1).ToString() + "], " + Position.ToString + ", 1st:" + PrimaryFace.ToString() + "-->" + Rotation.ToString()

End Function

Public Overrides Function HasColour(colour As FaceColour) As Boolean

For col = 0 To 1

If Colours(col) = colour Then Return True

Next

Return False

End Function

Public ReadOnly Property SideColour(ByVal topColour As FaceColour) As FaceColour

Get

If Not HasColour(topColour) Then Throw New ArgumentException("edge does not contain topcolour")

For Each colour In Colours

If colour <> topColour Then Return colour

Next

Throw New Exception("edge only contains topcolour == error in assigning colours")

End Get

End Property

Public ReadOnly Property EdgeColour(ByVal cube As Cube) As FaceColour

Get

Dim colourIndex As Integer

If PrimaryFace = cube.TopFace Then

colourIndex = 1

Else

colourIndex = 0

End If

Return Colours(colourIndex)

End Get

End Property

Public Overrides Function CorrectRotation(cubieFaceColour As FaceColour, cubeFaceColour As FaceColour) As Boolean

If Not HasColour(cubieFaceColour) Then Return False

Select Case cubieFaceColour

Case FaceColour.W, FaceColour.Y

If Rotation <> cubeFaceColour Then Return False

Case FaceColour.R, FaceColour.O

If cubieFaceColour = PrimaryFace Then

If Rotation <> cubeFaceColour Then Return False

Else

If Rotation = cubeFaceColour Then Return False

End If

Case FaceColour.B, FaceColour.G

If Rotation = cubeFaceColour Then Return False

End Select

Return True

End Function

Public Overrides Function Correct(cube As Cube) As Boolean

Return OnFace(Colours(0), cube) And OnFace(Colours(1), cube) And

CorrectRotation(Colours(0), Colours(0)) And CorrectRotation(Colours(1), Colours(1))

End Function

Public Overrides Sub SetColoursFromColourString(colourString As String)

For i = 0 To 1

Colours(i) = ColourChar2FaceNumber(colourString(i))

Next

End Sub

End Class

## Middle.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

<Serializable()> Public Class Middle

Inherits Block

Public Sub New()

ReDim Preserve Colours(0)

End Sub

Public Overrides Function ToString() As String

Return Name + ", [" + Colours(0).ToString() + "], " + Position.ToString + ", 1st:" + PrimaryFace.ToString() + "-->" + Rotation.ToString()

End Function

Public Overrides Function HasColour(colour As FaceColour) As Boolean

Return Colours(0) = colour

End Function

Public Overrides Function CorrectRotation(cubieFaceColour As FaceColour, cubeFaceColour As FaceColour) As Boolean

Return True

End Function

Public Overrides Function Correct(cube As Cube) As Boolean

Return True

End Function

Public Overrides Function CorrectForFace(faceColour As FaceColour, cube As Cube) As Boolean

Return True

End Function

Public Overrides Function CorrectlyOnOppositeFace(cubieFaceColour As FaceColour, cube As Cube) As Boolean

Return False

End Function

Public Overrides Sub SetColoursFromColourString(colourString As String)

Throw New NotImplementedException()

End Sub

End Class

## Matrices.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

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)

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

<Serializable()> Public MustInherit Class Vector

Private \_rows

Public Property NoOfRows

Get

Return \_rows

End Get

Protected Set(value)

\_rows = value

End Set

End Property

Public MustOverride Overrides Function ToString() As String

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(mat1.Row1.Dot(mat2.Column1),

mat1.Row1.Dot(mat2.Column2),

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

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

mat1.Row2.Dot(mat2.Column2),

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

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

mat1.Row3.Dot(mat2.Column2),

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

End Operator

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

Return New Vector3x1(mat.Row1.Dot(vec), mat.Row2.Dot(vec), mat.Row3.Dot(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 mat As Matrix4x4, ByVal vec As Vector4x1) As Vector4x1

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

End Class

Public Class Vector2x1

Inherits Vector

Public Sub New()

Me.NoOfRows = 2

Me.x = 0

Me.y = 0

End Sub

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

Me.NoOfRows = 2

Me.x = x

Me.y = y

End Sub

Public Shared Operator =(ByVal vec1 As Vector2x1, ByVal vec2 As Vector2x1) As Boolean

Return vec1.x = vec2.x And vec1.y = vec2.y

End Operator

Public Shared Operator <>(ByVal vec1 As Vector2x1, ByVal vec2 As Vector2x1) As Boolean

Return Not vec1 = vec2

End Operator

Private \_x As Single

Private \_y As Single

Public Property x As Single

Get

Return \_x

End Get

Set(ByVal value As Single)

\_x = Math.Round(value, 5)

End Set

End Property

Public Property y As Single

Get

Return \_y

End Get

Set(ByVal value As Single)

\_y = Math.Round(value, 5)

End Set

End Property

Public Overrides Function ToString() As String

Throw New NotImplementedException()

End Function

Public Function Dot(vec As Vector2x1) As Single

Return (Me.x \* vec.x + Me.y \* vec.y)

End Function

End Class

<Serializable()> 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

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, 5)

End Set

End Property

Public Property y As Single

Get

Return \_y

End Get

Set(ByVal value As Single)

\_y = Math.Round(value, 5)

End Set

End Property

Public Property z As Single

Get

Return \_z

End Get

Set(ByVal value As Single)

\_z = Math.Round(value, 5)

End Set

End Property

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

Return vec1.x = vec2.x And vec1.y = vec2.y And vec1.z = vec2.z

End Operator

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

Return Not vec1 = vec2

End Operator

Public Overrides Function ToString() As String

Return "[" + x.ToString() + ", " + y.ToString() + ", " + z.ToString() + "]"

End Function

Public Function Dot(vec As Vector3x1) As Single

Return (Me.x \* vec.x + Me.y \* vec.y + Me.z \* vec.z)

End Function

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

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

Return vec1.x = vec2.x And vec1.y = vec2.y And vec1.z = vec2.z And vec1.w = vec2.w

End Operator

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

Return Not vec1 = vec2

End Operator

Public Overrides Function ToString() As String

Throw New NotImplementedException()

End Function

Public Function Dot(vec As Vector4x1) As Single

Return (Me.x \* vec.x + Me.y \* vec.y + Me.z \* vec.z + Me.w \* vec.w)

End Function

End Class

Public Function GetRotationMatrix(ByVal axis As Axis, ByVal angle As Single) As Matrix3x3

Select Case axis

Case axis.X

Return 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 axis.Y

Return 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 axis.Z

Return 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 with rotation Axis")

Return New Matrix3x3()

End Select

End Function

End Module

## TopLayerSolver.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

Public Class TopLayerSolver

Private cube As Cube

Sub New(ByRef cubeToBeSolved As Cube)

cube = cubeToBeSolved

End Sub

Public Sub Solve()

Dim colourOfCompletedFace As FaceColour = FaceColour.None

colourOfCompletedFace = GetAnyCompleteFace()

If colourOfCompletedFace = FaceColour.None Then

Dim colourOfRotatedCompletedFace As FaceColour = GetAnyRotatedCompleteFace()

If colourOfRotatedCompletedFace = FaceColour.None Then

DoTopFace()

Else

cube.RotateFaceToTop(colourOfRotatedCompletedFace)

End If

RotateTopFaceCorrectly()

Else

cube.RotateFaceToTop(colourOfCompletedFace)

End If

End Sub

Private Sub DoTopFace()

Dim colourOfCross As FaceColour = FaceColour.None

colourOfCross = GetAnyCrossWithCorrectEdges()

If colourOfCross = FaceColour.None Then

DoTopCross()

Else

cube.RotateFaceToTop(colourOfCross)

End If

DoTopCorners()

End Sub

Private Sub DoTopCross()

MakeWrongEdgedCrossOnBottomFace()

MakeCorrectCrossFromOppositeCross()

End Sub

Private Sub DoTopCorners()

Dim topCornersWhenSolved() As Corner = cube.BlocksByColour(cube.TopFace).Extract(Of Corner)()

For Each corner In topCornersWhenSolved

RotateTopFaceCorrectly()

If Not corner.Correct(cube) Then PutCornerIntoCorrectPlaceOnTopFace(corner)

Next

End Sub

Private Sub MakeWrongEdgedCrossOnBottomFace()

Dim topColourEdges() As Edge = cube.BlocksByColour(cube.TopFace).Extract(Of Edge)()

cube.RotateFaceToTop(cube.BottomFace)

For Each edge In topColourEdges

If Not edge.CorrectlyOnOppositeFace(cube.BottomFace, cube) Then PutEdgeIntoEmptySpaceOnTopFace(edge, cube.BottomFace)

Next

cube.RotateFaceToTop(cube.BottomFace)

End Sub

Private Sub PutEdgeIntoEmptySpaceOnTopFace(ByVal edge As Edge, ByVal crossColour As FaceColour)

Dim edgeBeingDisplaced As Edge = GetEdgeBeingDisplaced(edge)

While edgeBeingDisplaced.CorrectlyOnOppositeFace(crossColour, cube) And edgeBeingDisplaced.HasColour(crossColour)

cube.RotateFace(cube.TopFace, Direction.CLOCKWISE)

edgeBeingDisplaced = GetEdgeBeingDisplaced(edge)

End While

Select Case edge.Layer

Case Layer.TOP

RotateForwardsTopLayerEdgeToTop(edge)

Case Layer.MIDDLE

RotateMiddleLayerEdgeToTop(edge)

Case Layer.BOTTOM

If edge.CorrectRotation(crossColour, cube.BottomFace) Then

RotateDownwardsBottomLayerEdgeToTop(edge)

Else

RotateForwardsBottomLayerEdgeToTop(edge)

End If

End Select

End Sub

Private Function GetEdgeBeingDisplaced(ByVal edgeBeingMoved As Edge)

Dim possibleEdges() As Edge = cube.TopEdgesClockwise

For Each edge In possibleEdges

If edge.OnFace(FaceToRotateForCrossEdge(edgeBeingMoved), cube) Then Return edge

Next

Throw New Exception("Didn't return an edge being displaced - code error")

End Function

Private Sub RotateForwardsTopLayerEdgeToTop(edge As Edge)

If edge.Layer <> Layer.TOP Then Throw New ArgumentException("The cubie is not on the top layer.")

cube.RotateFace(FaceToRotateForCrossEdge(edge), Direction.CLOCKWISE)

cube.RotateFace(cube.TopFace, Direction.ANTICLOCKWISE)

RotateMiddleLayerEdgeToTop(edge)

End Sub

Private Sub RotateMiddleLayerEdgeToTop(edge As Edge)

If edge.Layer <> Layer.MIDDLE Then Throw New ArgumentException("The cubie is not on the middle layer.")

Dim faceToRotate As FaceColour = FaceToRotateForCrossEdge(edge)

Dim direction As Direction = DirectionForMiddleLayerEdgeToTop(edge, faceToRotate, cube)

cube.RotateFace(faceToRotate, direction)

End Sub

Private Function DirectionForMiddleLayerEdgeToTop(ByVal cubie As Edge, ByVal faceToRotate As FaceColour, ByVal cube As Cube) As Direction

Dim faces() As FaceColour = cube.FaceColours

Select Case faceToRotate

Case faces(MoveFaces.LEFT)

Return cubie.Position.z

Case faces(MoveFaces.BACK)

Return -cubie.Position.x

Case faces(MoveFaces.RIGHT)

Return -cubie.Position.z

Case faces(MoveFaces.FRONT)

Return cubie.Position.x

Case Else

Return Direction.NO\_CHANGE

End Select

End Function

Private Sub RotateDownwardsBottomLayerEdgeToTop(ByVal edge As Edge)

If edge.Layer <> Layer.BOTTOM Then Throw New ArgumentException("The cubie is not on the bottom layer.")

Dim faceToRotate As FaceColour = FaceToRotateForCrossEdge(edge)

cube.RotateFace(faceToRotate, Direction.HALF\_TURN)

End Sub

Private Sub RotateForwardsBottomLayerEdgeToTop(ByVal edge As Edge)

If edge.Layer <> Layer.BOTTOM Then Throw New ArgumentException("The cubie is not on the bottom layer.")

cube.RotateFace(cube.TopFace, Direction.CLOCKWISE)

Dim originalFace As FaceColour = FaceToRotateForCrossEdge(edge)

cube.RotateFace(originalFace, Direction.CLOCKWISE)

RotateMiddleLayerEdgeToTop(edge)

cube.RotateFace(originalFace, Direction.ANTICLOCKWISE)

End Sub

Private Function FaceToRotateForCrossEdge(ByVal cubie As Edge) As FaceColour

Return IIf(cubie.Layer = Layer.MIDDLE, FaceToRotateForMiddleLayerEdge(cubie, cube.BottomFace),

FaceToRotateForNotMiddleLayerEdge(cubie))

End Function

Private Function FaceToRotateForMiddleLayerEdge(ByVal cubie As Edge, ByVal colourThatWillPointUp As FaceColour) As FaceColour

Dim faces() As FaceColour = cube.FaceColours

If cubie.PrimaryFace <> colourThatWillPointUp Then Return cubie.Rotation

Dim faceOne, faceTwo As FaceColour

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

Case -2 'Back left edge

faceOne = faces(MoveFaces.LEFT)

faceTwo = faces(MoveFaces.BACK)

Case 0

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

faceOne = faces(MoveFaces.FRONT)

faceTwo = faces(MoveFaces.LEFT)

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

faceOne = faces(MoveFaces.BACK)

faceTwo = faces(MoveFaces.RIGHT)

End If

Case 2 ' Front right edge

faceOne = faces(MoveFaces.RIGHT)

faceTwo = faces(MoveFaces.FRONT)

End Select

Return IIf(faceOne = cubie.Rotation, faceTwo, faceOne)

End Function

Private Function FaceToRotateForNotMiddleLayerEdge(ByVal cubie As Edge) As FaceColour

Dim faces() As FaceColour = cube.FaceColours

Select Case cubie.Position.x

Case -1 ' left edge cubie

Return faces(MoveFaces.LEFT)

Case 0

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

Return faces(MoveFaces.BACK)

Else ' front edge cubie

Return faces(MoveFaces.FRONT)

End If

Case 1 ' right edge cubie

Return faces(MoveFaces.RIGHT)

End Select

Throw New ArgumentException("edge.position is invalid")

End Function

Private Sub MakeCorrectCrossFromOppositeCross()

Dim crossEdges() As Edge = cube.BlocksByColour(cube.TopFace).Extract(Of Edge)()

For Each edge In crossEdges

Dim edgeSideColour As FaceColour = edge.SideColour(cube.TopFace)

While Not edge.OnFace(edgeSideColour, cube)

cube.RotateFace(cube.BottomFace, Direction.ANTICLOCKWISE)

End While

cube.RotateFace(edgeSideColour, Direction.HALF\_TURN)

Next

End Sub

Private Sub PutCornerIntoCorrectPlaceOnTopFace(ByVal corner As Corner)

RotateTopFaceCorrectly()

Select Case corner.Layer

Case Layer.TOP

If corner.Correct(cube) Then Return

Dim faceCornerIsOn As FaceColour = FacesEitherSideOfCorner(corner)(0)

cube.RotateFace(faceCornerIsOn, Direction.ANTICLOCKWISE)

cube.RotateFace(cube.BottomFace, Direction.HALF\_TURN)

cube.RotateFace(faceCornerIsOn, Direction.CLOCKWISE)

RotateCornerFromBottomLayerToTop(corner)

Case Layer.BOTTOM

RotateCornerFromBottomLayerToTop(corner)

Case Else

Throw New ArgumentException("corner is not on top or bottom layer")

End Select

End Sub

Private Sub RotateCornerFromBottomLayerToTop(ByVal corner As Corner)

RotateBottomFaceSoCornerLinesUpWithTopCross(corner)

Dim facesToRotate() As FaceColour = FacesEitherSideOfCorner(corner)

'when viewed from topFace

Dim anticlockwiseFace As FaceColour = facesToRotate(0)

Dim clockwiseFace As FaceColour = facesToRotate(1)

If TopCornerFacingDownwards(corner) Then

Algorithms.MoveTopCornerPointingDownwardsFromBottomToTop(clockwiseFace, anticlockwiseFace, cube)

Else

Dim faceRotatedTowards As FaceColour = FaceBottomCornerIsRotatedTowards(corner, cube.TopFace)

If faceRotatedTowards = clockwiseFace Then

Algorithms.MoveTopCornerPointingLeftFromBottomToTop(anticlockwiseFace, cube)

ElseIf faceRotatedTowards = anticlockwiseFace Then

Algorithms.MoveTopCornerPointingRightFromBottomToTop(clockwiseFace, cube)

End If

End If

End Sub

Private Function TopCornerFacingDownwards(ByVal corner As Corner)

Return corner.CorrectlyOnOppositeFace(cube.TopFace, cube)

End Function

Private Sub RotateBottomFaceSoCornerLinesUpWithTopCross(ByVal corner As Corner)

Dim cornerSideFaces() As FaceColour = corner.SideColours(cube.TopFace)

Dim cubeSideFaces() As FaceColour = FacesEitherSideOfCorner(corner)

Do Until (cubeSideFaces(0) = cornerSideFaces(0) And cubeSideFaces(1) = cornerSideFaces(1)) Or

(cubeSideFaces(0) = cornerSideFaces(1) And cubeSideFaces(1) = cornerSideFaces(0))

cube.RotateFace(cube.BottomFace, Direction.ANTICLOCKWISE)

cubeSideFaces = FacesEitherSideOfCorner(corner)

Loop

End Sub

Private Function FaceBottomCornerIsRotatedTowards(ByVal corner As Corner, ByVal mainColour As FaceColour) As FaceColour

If mainColour = corner.PrimaryFace Then

Return corner.Rotation

ElseIf mainColour = corner.SecondaryFace Then

Return corner.SecondaryRotation

Else

Dim sideFaces() As FaceColour = FacesEitherSideOfCorner(corner)

Dim facesAroundCorner() As FaceColour = {sideFaces(0), sideFaces(1), cube.BottomFace}

For Each face In facesAroundCorner

If face.ToString() = corner.Rotation Or face.ToString() = corner.SecondaryRotation Then Continue For

Return face

Next

End If

Throw New ArgumentException("error in corner")

End Function

'returns {anticlockwise, clockwise}

Private Function FacesEitherSideOfCorner(ByVal cubie As Corner) As FaceColour()

Dim faces() As FaceColour = cube.FaceColours

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

Case -2 'Back left corner

Return {faces(MoveFaces.LEFT), faces(MoveFaces.BACK)}

Case 0

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

Return {faces(MoveFaces.FRONT), faces(MoveFaces.LEFT)}

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

Return {faces(MoveFaces.BACK), faces(MoveFaces.RIGHT)}

End If

Case 2 ' Front right corner

Return {faces(MoveFaces.RIGHT), faces(MoveFaces.FRONT)}

End Select

Throw New ArgumentException("corner.position is invalid")

End Function

Private Sub RotateTopFaceCorrectly()

Dim edgesClockwise As Edge() = cube.TopEdgesClockwise

Dim offset As Integer = 0

For i As MoveFaces = MoveFaces.LEFT To MoveFaces.FRONT

If edgesClockwise(i - 1).HasColour(cube.FaceColours(MoveFaces.LEFT)) Then

offset = i - 1

Exit For

End If

Next

If offset = 3 Then offset = -1

cube.RotateFace(cube.TopFace, offset)

End Sub

''' <summary>

''' Gets the colour of the first complete face. If non complete, returns FaceColour.None

''' </summary>

Private Function GetAnyCompleteFace() As FaceColour

For face As FaceColour = Helpers.FaceColour.W To Helpers.FaceColour.O

Dim faceBlocks As Block() = cube.BlocksByColour(face)

For Each block In faceBlocks

If Not block.Correct(cube) Then Exit For 'check next face

If faceBlocks.IsLastElement(block) Then Return face

Next

Next

Return FaceColour.None

End Function

''' <summary>

''' Gets the colour of the first complete face regardless of it's rotation. If non complete, returns FaceColour.None

''' </summary>

Private Function GetAnyRotatedCompleteFace() As FaceColour

For face As FaceColour = Helpers.FaceColour.W To Helpers.FaceColour.O

Dim faceBlocks As Block() = cube.BlocksByColour(face)

For Each block In faceBlocks

If Not block.CorrectForFace(face, cube) Or Not EdgesOfFaceInRightOrder(face) Then Exit For 'check next face

If faceBlocks.IsLastElement(block) Then Return face

Next

Next

Return FaceColour.None

End Function

Private Function EdgesOfFaceInRightOrder(ByVal face As FaceColour) As Boolean

Dim originalTop As FaceColour = cube.CurrentOrientation.Top

cube.RotateFaceToTop(face)

Dim faceBlocksClockwise As Block() = cube.TopBlocksClockwise

ReDim Preserve faceBlocksClockwise(7) ' removes middle block

Dim edgesClockwise As Edge() = faceBlocksClockwise.Extract(Of Edge)()

For i As MoveFaces = MoveFaces.LEFT To MoveFaces.FRONT

If edgesClockwise(i - 1).HasColour(cube.FaceColours(MoveFaces.LEFT)) Then

edgesClockwise = edgesClockwise.Rotate(i - 1)

faceBlocksClockwise = faceBlocksClockwise.Rotate(2 \* i - 2)

Exit For

End If

Next

For i = MoveFaces.LEFT To MoveFaces.FRONT

If Not faceBlocksClockwise(2 \* i - 1).HasColour(cube.FaceColours(i)) Or

Not faceBlocksClockwise(2 \* i - 2).HasColour(cube.FaceColours(i)) Or

Not faceBlocksClockwise((2 \* i) Mod 8).HasColour(cube.FaceColours(i)) Then

cube.RotateFaceToTop(originalTop)

Return False

End If

Next

cube.RotateFaceToTop(originalTop)

Return True

End Function

''' <summary>

''' Gets the colour of the first complete cross. If non complete, returns nothing.

''' </summary>

Private Function GetAnyCrossWithCorrectEdges() As FaceColour

For face As FaceColour = Helpers.FaceColour.W To Helpers.FaceColour.O

Dim faceEdges As Edge() = cube.BlocksByColour(face).Extract(Of Edge)()

For Each edgeCubie In faceEdges

If Not edgeCubie.CorrectForFace(face, cube) Or

(Not edgeCubie.Correct(cube) And Not EdgesOfCrossInRightOrder(face)) Then Exit For 'check next face

If faceEdges.IsLastElement(edgeCubie) Then Return face

Next

Next

Return FaceColour.None

End Function

Private Function EdgesOfCrossInRightOrder(ByVal crossFace As FaceColour) As Boolean

Dim originalTop As FaceColour = cube.CurrentOrientation.Top

cube.RotateFaceToTop(crossFace)

Dim edgesClockwise As Edge() = cube.TopEdgesClockwise

For i As MoveFaces = MoveFaces.LEFT To MoveFaces.FRONT

If edgesClockwise(i - 1).HasColour(MoveFaces.LEFT) Then

edgesClockwise = edgesClockwise.Rotate(i - 1)

Exit For

End If

Next

For i = MoveFaces.LEFT To MoveFaces.FRONT

If Not edgesClockwise(i - 1).HasColour(cube.FaceColours(i)) Then

cube.RotateFaceToTop(originalTop)

Return False

End If

Next

cube.RotateFaceToTop(originalTop)

Return True

End Function

End Class

## MiddleLayerSolver.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

Public Class MiddleLayerSolver

Private cube As Cube

Sub New(ByRef cubeToBeSolved As Cube)

cube = cubeToBeSolved

End Sub

Public Sub Solve()

If Not MiddleRowCorrect() Then DoMiddleRow()

End Sub

Private Function MiddleRowCorrect() As Boolean

Dim middleEdges() As Edge = cube.MiddleEdgesClockwise

If Not (middleEdges(0).HasColour(cube.FaceColours(MoveFaces.FRONT)) And

middleEdges(0).HasColour(cube.FaceColours(MoveFaces.LEFT))) Then Return False

For i = 1 To 3

If Not (middleEdges(i).HasColour(cube.FaceColours(i)) And

middleEdges(i).HasColour(cube.FaceColours(i + 1))) Then Return False

Next

Return True

End Function

Private Sub DoMiddleRow()

Dim edgesWhenSolved() As Edge = MiddleEdgesWhenSolved()

For Each edge In edgesWhenSolved

If Not edge.Correct(cube) Then PutEdgeInMiddle(edge)

Next

End Sub

Private Function MiddleEdgesWhenSolved()

Dim edges(3) As Edge

Dim index As Integer = 0

For Each cubie In cube.Edges

If Not (cubie.HasColour(cube.CurrentOrientation.Top) Or

cubie.HasColour(cube.CurrentOrientation.Bottom)) Then

edges(index) = cubie

index += 1

End If

Next

Return edges

End Function

Private Sub PutEdgeInMiddle(ByVal edge As Edge)

If edge.Layer = Layer.MIDDLE Then

Dim leftFace As FaceColour = ClockwiseFace(edge)

cube.RotateFaceToFront(leftFace)

Algorithms.MiddleRightAlgorithm(cube)

PutEdgeFromBottomToMiddle(edge)

ElseIf edge.Layer = Layer.BOTTOM Then

PutEdgeFromBottomToMiddle(edge)

Else

Throw New StageNotSuccessfulException("The top layer is not complete")

End If

End Sub

'Clockwise when viewed top-down

Private Function ClockwiseFace(ByVal edge As Edge) As FaceColour

Dim faces() As FaceColour = cube.FaceColours

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

Case -2 'Back left edge

Return faces(MoveFaces.BACK)

Case 0

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

Return faces(MoveFaces.LEFT)

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

Return faces(MoveFaces.RIGHT)

End If

Case 2 ' Front right edge

Return faces(MoveFaces.FRONT)

End Select

Throw New ArgumentException("edge has invalid position vector")

End Function

'Private Function AnticlockwiseFace(ByVal edge As Edge) As FaceColour

' Dim faces() As FaceColour = cube.FaceColours

' Select Case (edge.Position.x + edge.Position.z)

' Case -2 'Back left edge

' Return faces(MoveFaces.LEFT)

' Case 0

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

' Return faces(MoveFaces.FRONT)

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

' Return faces(MoveFaces.BACK)

' End If

' Case 2 ' Front right edge

' Return faces(MoveFaces.RIGHT)

' End Select

' Throw New ArgumentException("edge has invalid position vector")

'End Function

Private Sub PutEdgeFromBottomToMiddle(ByVal edge As Edge)

If edge.Layer <> Layer.BOTTOM Then Throw New ArgumentException("The cubie is not on the bottom layer.")

Dim sideFaces() As FaceColour = GetEdgeFaceColours(edge)

Dim clockwiseFace As FaceColour = sideFaces(0)

Dim anticlockwiseFace As FaceColour = sideFaces(1)

While Not edge.Correct(cube)

If edge.CorrectForFace(anticlockwiseFace, cube) Then

cube.RotateFaceToFront(anticlockwiseFace)

Algorithms.MiddleRightAlgorithm(cube)

ElseIf edge.CorrectForFace(clockwiseFace, cube) Then

cube.RotateFaceToFront(clockwiseFace)

Algorithms.MiddleLeftAlgorithm(cube)

Else

cube.RotateFace(cube.BottomFace, Direction.ANTICLOCKWISE)

End If

End While

End Sub

''' <returns> {clockwise, anticlockwise} viewed from bottom of the cube </returns>

Private Function GetEdgeFaceColours(ByVal edge As Edge) As FaceColour()

Dim cubieColours(1) As FaceColour

For i = 0 To 1

cubieColours(i) = edge.Colours(i)

Next

For face As MoveFaces = MoveFaces.LEFT To MoveFaces.FRONT

Dim clockwiseFace As MoveFaces = face Mod 4 + 1 'Clockwise when viewed from top

If cube.FaceColours(face) = cubieColours(0) And cube.FaceColours(clockwiseFace) = cubieColours(1) Or

cube.FaceColours(face) = cubieColours(1) And cube.FaceColours(clockwiseFace) = cubieColours(0) Then

Return {cube.FaceColours(face), cube.FaceColours(clockwiseFace)}

End If

Next

Throw New ArgumentException("The edge is not a middle piece")

End Function

End Class

## BottomLayerSolver.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

Public Class BottomLayerSolver

Private cube As Cube

Sub New(ByRef cubeToBeSolved As Cube)

cube = cubeToBeSolved

End Sub

Public Sub Solve()

If Not cube.Complete() Then SolveBottomLayer()

End Sub

Private Sub SolveBottomLayer()

If Not BottomFaceComplete() Then

If Not BottomCrossComplete() Then DoCross()

DoCorners()

End If

RotateBottomFaceCorrectly()

End Sub

Private Function BottomFaceComplete() As Boolean

Dim bottomCubies() As Block = cube.BlocksByColour(cube.BottomFace)

For Each cubie In bottomCubies

If Not cubie.CorrectForFace(cube.BottomFace, cube) Then Return False

Next

Return True

End Function

Private Function BottomCrossComplete() As Boolean

Dim faceEdges As Edge() = cube.BlocksByColour(cube.BottomFace).Extract(Of Edge)()

For Each edgeCubie In faceEdges

If Not edgeCubie.CorrectForFace(cube.BottomFace, cube) Or (Not edgeCubie.Correct(cube) And Not EdgesOfBottomCrossInRightOrder()) Then Return False

Next

Return True

End Function

Private Function EdgesOfBottomCrossInRightOrder() As Boolean

cube.RotateFaceToTop(cube.BottomFace)

Dim edgesClockwise As Edge() = cube.TopEdgesClockwise

For i As MoveFaces = MoveFaces.LEFT To MoveFaces.FRONT

If edgesClockwise(i - 1).HasColour(MoveFaces.LEFT) Then

edgesClockwise = edgesClockwise.Rotate(i - 1)

Exit For

End If

Next

For i = MoveFaces.LEFT To MoveFaces.FRONT

If Not edgesClockwise(i - 1).HasColour(cube.FaceColours(i)) Then

cube.RotateFaceToTop(cube.BottomFace)

Return False

End If

Next

cube.RotateFaceToTop(cube.BottomFace)

Return True

End Function

Private Sub DoCross()

cube.RotateFaceToTop(cube.BottomFace)

DoBottomStage\_L()

DoBottomStage\_Line()

DoBottomStage\_Cross()

PermuteTopCrossEdges()

cube.RotateFaceToTop(cube.BottomFace)

End Sub

'cube is upside down

Private Sub DoBottomStage\_L()

If cube.CorrectTopEdges.Length >= 2 Then Return 'L is done

Algorithms.BottomCrossAlgorithm(cube)

End Sub

'cube is upside down

Private Sub DoBottomStage\_Line()

Dim correctBottomEdges() As Edge = cube.CorrectTopEdges()

If cube.CorrectTopEdges.Length >= 3 Or

New Vector2x1(correctBottomEdges(0).Position.x, correctBottomEdges(0).Position.z).Dot(New Vector2x1(correctBottomEdges(1).Position.x, correctBottomEdges(1).Position.z)) <> 0 Then

Return ' line is done

End If

While (correctBottomEdges(0).Position.x <> -1 Or correctBottomEdges(1).Position.z <> -1) And

(correctBottomEdges(1).Position.x <> -1 Or correctBottomEdges(0).Position.z <> -1)

cube.RotateFace(cube.TopFace, Direction.ANTICLOCKWISE)

End While

Algorithms.BottomCrossAlgorithm(cube)

End Sub

'cube is upside down

Private Sub DoBottomStage\_Cross()

If cube.CorrectTopEdges.Length = 4 Then Return ' Cross is done

Dim correctBottomEdges() As Edge = cube.CorrectTopEdges()

While IsNotLine(correctBottomEdges)

cube.RotateFace(cube.TopFace, Direction.ANTICLOCKWISE)

End While

Algorithms.BottomCrossAlgorithm(cube)

End Sub

Private Function IsNotLine(ByVal correctCrossEdges()) As Boolean

Select Case correctCrossEdges.Length

Case 2

Return (correctCrossEdges(0).Position.x <> -1 Or correctCrossEdges(1).Position.x <> 1) And

(correctCrossEdges(1).Position.x <> -1 Or correctCrossEdges(0).Position.x <> 1)

Case 3

Return (correctCrossEdges(0).Position.x <> -1 Or correctCrossEdges(1).Position.x <> 1) And

(correctCrossEdges(0).Position.x <> -1 Or correctCrossEdges(2).Position.x <> 1) And

(correctCrossEdges(1).Position.x <> -1 Or correctCrossEdges(0).Position.x <> 1) And

(correctCrossEdges(1).Position.x <> -1 Or correctCrossEdges(2).Position.x <> 1) And

(correctCrossEdges(2).Position.x <> -1 Or correctCrossEdges(0).Position.x <> 1) And

(correctCrossEdges(2).Position.x <> -1 Or correctCrossEdges(1).Position.x <> 1)

Case Else

Throw New ArgumentException("2 edge faces are not solved - error in BottomLayerSolver")

End Select

End Function

Private Sub PermuteTopCrossEdges()

Dim crossEdges() As Edge = cube.TopEdgesClockwise()

Dim crossColours(3) As FaceColour

For edge = 0 To 3

crossColours(edge) = crossEdges(edge).SideColour(cube.TopFace)

Next

If crossColours(0) = Opposite(crossColours(2)) Or crossColours(1) = Opposite(crossColours(3)) Then

For faceColour = 1 To 4

If crossColours(0) = cube.FaceColours(faceColour) And crossColours(1) = cube.FaceColours(faceColour Mod 4 + 1) Then Return 'edges are correct

Next

Algorithms.BottomEdgeAlgorithmRight(cube)

cube.RotateFace(cube.TopFace, Direction.ANTICLOCKWISE)

Algorithms.BottomEdgeAlgorithmRight(cube)

Return

End If

Dim indexOfAnticlockwiseEdge As Integer

For edge = 0 To 3

For faceColour = 1 To 4

If crossColours(edge) = cube.FaceColours(faceColour) And

crossColours((edge + 1) Mod 4) = cube.FaceColours(faceColour Mod 4 + 1) Then

indexOfAnticlockwiseEdge = edge

End If

Next

Next

cube.RotateFace(cube.TopFace, indexOfAnticlockwiseEdge - 1)

Algorithms.BottomEdgeAlgorithmRight(cube)

End Sub

Private Sub DoCorners()

PositionBottomCorners()

RotateBottomFaceCorrectly()

OrientateBottomCorners()

End Sub

Private Sub PositionBottomCorners()

RotateBottomFaceCorrectly()

cube.RotateFaceToTop(cube.BottomFace)

Dim corners() As Corner = cube.TopCornersClockwise

Dim correctlyPositionedCorner As Corner

Dim nextClockwiseCorner As Corner = Nothing

Dim cornerPositionIndex As Integer

For cornerPositionIndex = 0 To 3

Dim corner As Corner = corners(cornerPositionIndex)

nextClockwiseCorner = corners((cornerPositionIndex + 1) Mod 4)

If Not CornerIsCorrectlyPositioned(corner) Then Continue For

If CornerIsCorrectlyPositioned(nextClockwiseCorner) Then

cube.RotateFaceToTop(cube.BottomFace)

Return

Else

correctlyPositionedCorner = corner

Exit For

End If

Next

If correctlyPositionedCorner Is Nothing Then

Algorithms.BottomClockwiseCornerAlgorithm(cube)

cube.RotateFaceToTop(cube.BottomFace)

PositionBottomCorners()

Return

End If

PermuteCorners(correctlyPositionedCorner, nextClockwiseCorner, cornerPositionIndex)

cube.RotateFaceToTop(cube.BottomFace)

End Sub

Private Function CornerIsCorrectlyPositioned(ByVal corner As Corner) As Boolean

Dim sideFaces() As FaceColour = FacesEitherSideOfCorner(corner)

Return corner.HasColour(sideFaces(0)) And corner.HasColour(sideFaces(1))

End Function

Private Sub PermuteCorners(correctlyPositionedCorner As Corner, nextClockwiseCorner As Corner, cornerPositionIndex As Integer)

Dim anticlockwiseFace As FaceColour = FacesEitherSideOfCorner(correctlyPositionedCorner)(0)

If nextClockwiseCorner.HasColour(anticlockwiseFace) Then

'Rotate anticlockwise

Select Case cornerPositionIndex

Case 1 : cube.RotateFace(cube.TopFace, Direction.ANTICLOCKWISE)

Case 2 : cube.RotateFace(cube.TopFace, Direction.HALF\_TURN)

Case 3 : cube.RotateFace(cube.TopFace, Direction.CLOCKWISE)

End Select

Algorithms.BottomAnticlockwiseCornerAlgorithm(cube)

Else

'Rotate clockwise

Select Case cornerPositionIndex

Case 0 : cube.RotateFace(cube.TopFace, Direction.ANTICLOCKWISE)

Case 1 : cube.RotateFace(cube.TopFace, Direction.HALF\_TURN)

Case 2 : cube.RotateFace(cube.TopFace, Direction.CLOCKWISE)

End Select

Algorithms.BottomClockwiseCornerAlgorithm(cube)

End If

End Sub

'returns {anticlockwise, clockwise} viewed from top

Private Function FacesEitherSideOfCorner(ByVal cubie As Corner) As FaceColour()

Dim faces() As FaceColour = cube.FaceColours

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

Case -2 'Back left corner

Return {faces(MoveFaces.LEFT), faces(MoveFaces.BACK)}

Case 0

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

Return {faces(MoveFaces.FRONT), faces(MoveFaces.LEFT)}

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

Return {faces(MoveFaces.BACK), faces(MoveFaces.RIGHT)}

End If

Case 2 ' Front right corner

Return {faces(MoveFaces.RIGHT), faces(MoveFaces.FRONT)}

End Select

Throw New ArgumentException("corner position is invalid")

End Function

Private Sub OrientateBottomCorners()

cube.RotateFaceToTop(cube.BottomFace)

Dim numberOfIncorrectCorners As Integer = cube.IncorrectTopCorners.Length

Select Case numberOfIncorrectCorners

Case 4

Orientate4Corners()

Case 2

Orientate2Corners()

Case 3

Orientate3Corners()

Case 0

Return

Case Else

Throw New StageNotSuccessfulException("Error in previous stages - invalid cube")

End Select

cube.RotateFaceToTop(cube.BottomFace)

End Sub

Private Sub Orientate4Corners()

Dim originalOrientation As CubeOrientation = New CubeOrientation(cube.CurrentOrientation)

Dim incorrectCorners() As Corner = cube.IncorrectTopCorners()

Dim facesCommonToBoth() As FaceColour

Dim faceBothRotatedTowards As FaceColour

For i = 0 To 3

'will only work if incorrectCorners is clockwise

facesCommonToBoth = GetFacesContainingBothCorners(incorrectCorners(i), incorrectCorners(i + 1))

faceBothRotatedTowards = GetFaceBothRotatedTowards(incorrectCorners(i), incorrectCorners(i + 1))

If faceBothRotatedTowards <> FaceColour.None Then Exit For

Next

cube.RotateFaceToTop(faceBothRotatedTowards)

If faceBothRotatedTowards = facesCommonToBoth(0) Then

cube.RotateFaceToLeft(facesCommonToBoth(1))

ElseIf faceBothRotatedTowards = facesCommonToBoth(1) Then

cube.RotateFaceToLeft(facesCommonToBoth(0))

Else

Throw New StageNotSuccessfulException()

End If

Algorithms.BottomEdgeAlgorithmRight(cube)

Algorithms.BottomEdgeAlgorithmLeft(cube)

cube.RotateFaceToTop(originalOrientation.Top)

cube.RotateFaceToFront(originalOrientation.Front)

cube.RotateFaceToTop(cube.BottomFace)

OrientateBottomCorners()

cube.RotateFaceToTop(cube.BottomFace)

End Sub

Private Sub Orientate2Corners()

Dim originalOrientation As CubeOrientation = New CubeOrientation(cube.CurrentOrientation)

Dim correctCorners() As Corner = cube.CorrectTopCorners()

If correctCorners.Length <> 2 Then Throw New InvalidOperationException

If CornersAreOpposite(correctCorners(0), correctCorners(1)) Then OrientateOppositeCorners()

Dim incorrectCorners() As Corner = cube.IncorrectTopCorners()

Dim facesCommonToBoth() As FaceColour = GetFacesContainingBothCorners(incorrectCorners(0), incorrectCorners(1))

Dim faceBothRotatedTowards As FaceColour = GetFaceBothRotatedTowards(incorrectCorners(0), incorrectCorners(1))

cube.RotateFaceToTop(faceBothRotatedTowards)

If faceBothRotatedTowards = facesCommonToBoth(0) Then

cube.RotateFaceToLeft(facesCommonToBoth(1))

ElseIf faceBothRotatedTowards = facesCommonToBoth(1) Then

cube.RotateFaceToLeft(facesCommonToBoth(0))

Else

Throw New StageNotSuccessfulException()

End If

Algorithms.BottomEdgeAlgorithmRight(cube)

Algorithms.BottomEdgeAlgorithmLeft(cube)

cube.RotateFaceToTop(originalOrientation.Top)

cube.RotateFaceToFront(originalOrientation.Front)

End Sub

Private Function CornersAreOpposite(ByVal corner1 As Corner, ByVal corner2 As Corner) As Boolean

Return ((corner1.Position.x + corner1.Position.z) - (corner2.Position.x + corner2.Position.z)) Mod 4 = 0

End Function

Private Sub OrientateOppositeCorners()

Dim bottomFaceColour As FaceColour = cube.TopFace

Dim incorrectCorners() As Corner = cube.IncorrectTopCorners()

cube.Rotate(Direction.ANTICLOCKWISE, Axis.Z)

Dim cornerOnTopFace As Corner

Do

cube.Rotate(Direction.ANTICLOCKWISE, Axis.X)

cornerOnTopFace = IIf(incorrectCorners(0).OnFace(cube.TopFace, cube), incorrectCorners(0), incorrectCorners(1))

Loop While Not cornerOnTopFace.CorrectRotation(bottomFaceColour, cube.TopFace)

Algorithms.BottomEdgeAlgorithmRight(cube)

Algorithms.BottomEdgeAlgorithmLeft(cube)

cube.Rotate(Direction.CLOCKWISE, Axis.Z)

End Sub

Private Function GetFaceBothRotatedTowards(ByVal corner1 As Corner, ByVal corner2 As Corner) As FaceColour

Dim face As FaceColour = FaceColour.None

If corner1.PrimaryFace = corner2.PrimaryFace And corner1.Rotation = corner2.Rotation Then

face = corner1.Rotation

ElseIf corner1.SecondaryFace = corner2.SecondaryFace And corner1.SecondaryRotation = corner2.SecondaryRotation Then

face = corner1.SecondaryRotation

Else

Dim corner1Faces() As FaceColour = FacesEitherSideOfCorner(corner1)

corner1Faces.Append(cube.TopFace)

'probably needs an except method written

face = corner1Faces.Except({corner1.Rotation, corner1.SecondaryRotation})(0)

End If

If corner1.OnFace(face, cube) And corner2.OnFace(face, cube) Then Return face

Return FaceColour.None

End Function

Private Function GetFacesContainingBothCorners(ByVal corner1 As Corner, ByVal corner2 As Corner) As FaceColour()

Dim corner1Faces() As FaceColour = FacesEitherSideOfCorner(corner1)

Dim corner2Faces() As FaceColour = FacesEitherSideOfCorner(corner2)

Dim commonFaces() As FaceColour = corner1Faces.Intersect(corner2Faces)

commonFaces.Append(cube.TopFace)

Return commonFaces

End Function

Private Sub Orientate3Corners()

cube.Rotate(Direction.ANTICLOCKWISE, Axis.Z)

Dim leftTopEdgeCorners(1) As Corner

Dim count As Integer

Do

cube.Rotate(Direction.CLOCKWISE, Axis.X)

count = 0

For Each corner In cube.Corners

If corner.Position = New Vector3x1(-1, 1, 1) Or corner.Position = New Vector3x1(-1, 1, -1) Then

leftTopEdgeCorners(count) = corner

count += 1

End If

Next

Loop While (leftTopEdgeCorners(0).Correct(cube) Or leftTopEdgeCorners(1).Correct(cube))

Algorithms.BottomEdgeAlgorithmRight(cube)

Algorithms.BottomEdgeAlgorithmLeft(cube)

cube.Rotate(Direction.CLOCKWISE, Axis.Z)

If cube.CorrectTopCorners.Length <> 4 Then

cube.RotateFaceToTop(cube.BottomFace)

OrientateBottomCorners()

End If

cube.RotateFaceToTop(cube.BottomFace)

End Sub

Private Sub RotateBottomFaceCorrectly()

cube.RotateFaceToTop(cube.BottomFace)

Dim edgesClockwise As Edge() = cube.TopEdgesClockwise

Dim offset As Integer = 0

For i As MoveFaces = MoveFaces.LEFT To MoveFaces.FRONT

If edgesClockwise(i - 1).HasColour(cube.FaceColours(MoveFaces.LEFT)) Then

offset = i - 1

Exit For

End If

Next

If offset = 3 Then offset = -1

cube.RotateFace(cube.TopFace, offset)

cube.RotateFaceToTop(cube.BottomFace)

End Sub

End Clas

## CubeOrientation.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

<Serializable()> Public Class CubeOrientation

Public Property Top As FaceColour

Public ReadOnly Property Bottom As FaceColour

Get

Return Opposite(Top)

End Get

End Property

Public Property Front As FaceColour

Public ReadOnly Property UpsideDown As CubeOrientation

Get

Return New CubeOrientation(Bottom, Front)

End Get

End Property

Sub New(ByVal top As Char, ByVal front As Char)

Me.Top = ColourChar2FaceNumber(top)

Me.Front = ColourChar2FaceNumber(front)

End Sub

Sub New(ByVal top As FaceColour, ByVal front As FaceColour)

Me.Top = top

Me.Front = front

End Sub

Sub New(ByVal orientation As CubeOrientation)

Top = orientation.Top

Front = orientation.Front

End Sub

Public Overrides Function ToString() As String

Return "Top: " + Top.ToString() + ", Front: " + Front.ToString()

End Function

Public Function Copy() As CubeOrientation

Return New CubeOrientation(Top, Front)

End Function

End Class

## Algorithms.vb

Imports RubiksCubeSolver\_v2\_0.Helpers.PublicConstants

Imports RubiksCubeSolver\_v2\_0.Helpers.PublicConstants.Movecube.FaceColours

Imports RubiksCubeSolver\_v2\_0.Helpers.PublicConstants.Direction

Module Algorithms

Public Sub MoveTopCornerPointingDownwardsFromBottomToTop(ByVal clockwiseFace As FaceColour, ByVal anticlockwiseFace As FaceColour, ByRef cube As Cube)

cube.RotateFace(anticlockwiseFace, Direction.ANTICLOCKWISE)

cube.RotateFace(cube.BottomFace, Direction.CLOCKWISE)

cube.RotateFace(anticlockwiseFace, Direction.CLOCKWISE)

cube.RotateFace(clockwiseFace, Direction.CLOCKWISE)

cube.RotateFace(cube.BottomFace, Direction.HALF\_TURN)

cube.RotateFace(clockwiseFace, Direction.ANTICLOCKWISE)

End Sub

Public Sub MoveTopCornerPointingLeftFromBottomToTop(ByVal anticlockwiseFace As FaceColour, ByRef cube As Cube)

cube.RotateFace(cube.BottomFace, Direction.ANTICLOCKWISE)

cube.RotateFace(anticlockwiseFace, Direction.ANTICLOCKWISE)

cube.RotateFace(cube.BottomFace, Direction.CLOCKWISE)

cube.RotateFace(anticlockwiseFace, Direction.CLOCKWISE)

End Sub

Public Sub MoveTopCornerPointingRightFromBottomToTop(ByVal clockwiseFace As FaceColour, ByRef cube As Cube)

cube.RotateFace(cube.BottomFace, Direction.CLOCKWISE)

cube.RotateFace(clockwiseFace, Direction.CLOCKWISE)

cube.RotateFace(cube.BottomFace, Direction.ANTICLOCKWISE)

cube.RotateFace(clockwiseFace, Direction.ANTICLOCKWISE)

End Sub

Public Sub MiddleLeftAlgorithm(ByRef cube As Cube)

cube.RotateFace(cube.FaceColours(MoveFaces.BOTTOM), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.BOTTOM), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.BOTTOM), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.FRONT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.BOTTOM), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.FRONT), CLOCKWISE)

End Sub

Public Sub MiddleRightAlgorithm(ByRef cube As Cube)

cube.RotateFace(cube.FaceColours(MoveFaces.BOTTOM), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.BOTTOM), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.BOTTOM), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.FRONT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.BOTTOM), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.FRONT), ANTICLOCKWISE)

End Sub

Public Sub BottomCrossAlgorithm(ByRef cube As Cube)

cube.RotateFace(cube.FaceColours(MoveFaces.FRONT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.FRONT), ANTICLOCKWISE)

End Sub

Public Sub BottomEdgeAlgorithmRight(ByRef cube As Cube)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), HALF\_TURN)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), ANTICLOCKWISE)

End Sub

Public Sub BottomEdgeAlgorithmLeft(ByRef cube As Cube)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), HALF\_TURN)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), CLOCKWISE)

End Sub

Public Sub BottomAnticlockwiseCornerAlgorithm(ByRef cube As Cube)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), CLOCKWISE)

End Sub

Public Sub BottomClockwiseCornerAlgorithm(ByRef cube As Cube)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), ANTICLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.LEFT), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.TOP), CLOCKWISE)

cube.RotateFace(cube.FaceColours(MoveFaces.RIGHT), ANTICLOCKWISE)

End Sub

End Module

## ArrayMethods.vb

Imports System.Runtime.CompilerServices

Imports RubiksCubeSolver\_v2\_0.Helpers

Public Module ArrayExtentions

<Extension()>

Public Function Zero(ByRef array() As Integer) As Integer()

If array.Length < 1 Then Return Nothing

For i = 0 To UBound(array)

array(i) = 0

Next

Return array

End Function

<Extension()>

Public Sub Append(Of T)(ByRef array() As T, ByVal item As T)

Try

ReDim Preserve array(array.Length)

array(array.Length - 1) = item

Catch ex As NullReferenceException

array = {item}

End Try

End Sub

<Extension()>

Public Function Intersect(ByVal faces1() As FaceColour, ByVal faces2() As FaceColour) As FaceColour()

If faces1.Length < faces2.Length Then

Dim tmp() As FaceColour = faces1.Copy()

faces1 = faces2.Copy()

faces2 = tmp.Copy()

End If

Dim intersection() As FaceColour

For Each colour In faces1

If faces2.LinearSearch(colour) <> -1 Then intersection.Append(colour)

Next

Return intersection

End Function

<Extension()>

Public Function Extract(Of T As Block)(ByVal blocks() As Block) As T()

Dim tArray(blocks.Length) As T

Dim index As Integer = 0

For Each block In blocks

If block.GetType = GetType(T) Then

tArray(index) = block

index += 1

End If

Next

ReDim Preserve tArray(index - 1)

Return tArray

End Function

<Extension()>

Public Function Copy(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 With {

.Name = corners(i).Name,

.Rotation = corners(i).Rotation,

.SecondaryFace = corners(i).SecondaryFace,

.SecondaryRotation = corners(i).SecondaryRotation,

.Position = corners(i).Position,

.Colours = corners(i).Colours

}

Next

Return cornerCopy

End Function

<Extension()>

Public Function Copy(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 With {

.Name = edges(i).Name,

.Rotation = edges(i).Rotation,

.Position = edges(i).Position,

.Colours = edges(i).Colours

}

Next

Return edgeCopy

End Function

<Extension()>

Public Function Copy(Of T)(ByVal array(,) As T) As T(,)

Dim rows As Integer = array.GetLength(0) - 1

Dim columns As Integer = array.GetLength(1) - 1

Dim copyArray(rows, columns) As T

For i = 0 To rows

For j = 0 To columns

copyArray(i, j) = array(i, j)

Next

Next

Return copyArray

End Function

<Extension()>

Public Function Copy(Of T)(ByVal array() As T) As T()

Dim copyArray(array.Length - 1) As T

For i = 0 To array.Length - 1

copyArray(i) = array(i)

Next

Return copyArray

End Function

<Extension()>

Public Function Copy(Of T)(ByVal queue As Queue(Of T)) As Queue(Of T)

Dim copyQueue As New Queue(Of T)

Dim numberOfElements As Integer = queue.Count

For i = 0 To numberOfElements - 1

copyQueue.Enqueue(queue.ElementAt(i))

Next

Return copyQueue

End Function

<Extension()>

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

If array.Length < 1 Then Return -1

For i = 0 To UBound(array)

If array(i) = value Then Return i

Next

Return -1

End Function

<Extension()>

Public Function LinearSearch(ByVal array() As FaceColour, ByVal value As FaceColour) As FaceColour

If array.Length < 1 Then Return -1

For i = 0 To UBound(array)

If array(i) = value Then Return i

Next

Return -1

End Function

<Extension()>

Public Function AnythingElseInArray(ByVal array() As Integer, ByVal value As Integer) As Boolean

If array.Length < 1 Then Return True

For i = 0 To UBound(array)

If array(i) <> value Then Return True

Next

Return False

End Function

<Extension()>

Public Function AnythingElseInArray(ByVal array(,) As Boolean, ByVal value As Boolean) As Boolean

If array.Length < 1 Then Return True

For i = 0 To array.GetLength(0) - 1

For j = 0 To array.GetLength(1) - 1

If array(i, j) <> value Then Return True

Next

Next

Return False

End Function

<Extension()>

Public Function CountNotNothing(Of T)(ByVal array() As T) As Integer

Dim count As Integer = 0

For Each item In array

If item IsNot Nothing Then

count += 1

End If

Next

Return count

End Function

<Extension()>

Public Function IsLastElement(Of T)(array As T(), element As T) As Boolean

Return System.Array.IndexOf(array, element) = array.Length - 1

End Function

''' <summary>

''' Cycles an array round so so that the element at newStartIndex is now at index 0

''' </summary>

''' <param name="indexOfNewStart">0 based index of the item to be at the start of the returned array</param>

<Extension()>

Public Function Rotate(Of T)(ByVal array() As T, ByVal indexOfNewStart As Integer) As T()

Dim rotatedArray(array.Length - 1) As T

For i = 0 To array.Length - 1

rotatedArray(i) = array((indexOfNewStart + i) Mod (array.Length))

Next

Return rotatedArray

End Function

End Module

## InstrutcionList.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

<Serializable()> Public Class InstructionList

Inherits List(Of Instruction)

Public Sub AddFaceTurn(ByVal face As MoveFaces, ByVal direction As Direction)

If direction = Direction.NO\_CHANGE Then Return

Me.Add(New FaceInstruction(face, direction))

End Sub

Public Sub AddOrientationChange(ByVal axis As Axis, ByVal direction As Direction)

If direction = Direction.NO\_CHANGE Then Return

Me.Add(New CubeInstruction(axis, direction))

End Sub

Public Sub Optimise()

Dim anySimplificationsMade As Boolean = False

Do

anySimplificationsMade = False

Dim ptr As Integer = 0

While ptr < Me.Count - 1

Dim currentInstruction As Instruction = Me.ElementAt(ptr)

Dim nextInstruction As Instruction = Me.ElementAt(ptr + 1)

If currentInstruction.GetType() <> nextInstruction.GetType() Then

ptr += 1

Continue While

End If

If currentInstruction.Move <> nextInstruction.Move Then

ptr += 1

Continue While

End If

Dim directionSum As Integer = Math.Abs((currentInstruction.Direction + nextInstruction.Direction) Mod 4)

If directionSum = 3 Then directionSum = -1

currentInstruction.Direction = directionSum

If currentInstruction.Direction = Direction.NO\_CHANGE Then

Me.RemoveAt(ptr)

ptr -= 1

End If

Me.RemoveAt(ptr + 1)

ptr += 1

anySimplificationsMade = True

End While

Loop While anySimplificationsMade

End Sub

Public Function Copy() As InstructionList

Dim copyList As New InstructionList

For i = 0 To Me.Count - 1

copyList.Add(Me.ElementAt(i))

Next

Return copyList

End Function

Public Overrides Function ToString() As String

Dim rtn As String = ""

For i = 0 To Me.Count - 1

rtn += Me.ElementAt(i).ToString + ", "

Next

Return rtn

End Function

End Class

## Instruction.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

<Serializable()> Public MustInherit Class Instruction

Public Property Move As Integer

Public Property Direction As Direction

Public MustOverride Overrides Function ToString() As String

Public MustOverride Function Copy() As Instruction

End Class

## FaceInstruction.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

<Serializable()> Public Class FaceInstruction

Inherits Instruction

Public Sub New(ByVal face As MoveFaces, ByVal direction As Direction)

Move = face

Me.Direction = direction

End Sub

Public Overrides Function ToString() As String

Dim instructionString As String

Select Case Move

Case MoveFaces.TOP : instructionString = "U"

Case MoveFaces.LEFT : instructionString = "L"

Case MoveFaces.BACK : instructionString = "B"

Case MoveFaces.RIGHT : instructionString = "R"

Case MoveFaces.FRONT : instructionString = "F"

Case MoveFaces.BOTTOM : instructionString = "D"

Case Else

Throw New ArgumentException("Invalid face")

End Select

Select Case Direction

Case Direction.CLOCKWISE : instructionString += ""

Case Direction.ANTICLOCKWISE : instructionString += "'"

Case Direction.HALF\_TURN : instructionString += "2"

Case Else

Throw New ArgumentException("Invalid direction")

End Select

Return instructionString

End Function

Public Overrides Function Copy() As Instruction

Return New FaceInstruction(Me.Move, Me.Direction)

End Function

End Class

## CubeInstruction.vb

Imports RubiksCubeSolver\_v2\_0.Helpers

<Serializable()> Public Class CubeInstruction

Inherits Instruction

Public Sub New(ByVal axis As Axis, ByVal direction As Direction)

Move = axis

Me.Direction = direction

End Sub

Public Overrides Function ToString() As String

Dim instructionString As String

Select Case Move

Case Axis.X : instructionString = "X"

Case Axis.Y : instructionString = "Y"

Case Axis.Z : instructionString = "Z"

Case Else

Throw New ArgumentException("Invalid axis")

End Select

Select Case Direction

Case Direction.CLOCKWISE : instructionString += ""

Case Direction.ANTICLOCKWISE : instructionString += "'"

Case Direction.HALF\_TURN : instructionString += "2"

Case Else

Throw New ArgumentException("Invalid direction")

End Select

Return instructionString

End Function

Public Overrides Function Copy() As Instruction

Return New CubeInstruction(Me.Move, Me.Direction)

End Function

End Class

## 3dOutput.vb

Imports OpenTK

Imports OpenTK.Graphics.OpenGL

Imports RubiksCubeSolver\_v2\_0.Helpers.PublicConstants

Imports System.Runtime.Serialization.Formatters.Binary

Public Class \_3DOutput

Const pad As Single = 5 / 100

ReadOnly projection As OpenTK.Matrix4 = Matrix4.CreateOrthographic(4 / 3 \* 7, 7, 1, 1000)

ReadOnly initalPositions() As OpenTK.Vector3

ReadOnly initalRotations() As OpenTK.Quaternion

ReadOnly initalScrambledCube As Cube

ReadOnly instructions As OutputInstructionList

Private shaderProgram As ShaderProgram

Private camera As Camera

Private deltaTime As Single = 0

Private isMouseDown As Boolean

Private currentCube As Cube

Private cubieRotations(26) As OutputBlock

Private showingRotation As Boolean = False

Public Sub New(ByVal cube As Cube)

InitializeComponent()

initalScrambledCube = cube

initalScrambledCube.CurrentOrientation = New CubeOrientation(FaceColour.W, FaceColour.R)

initalPositions = GetInitialPositions()

initalRotations = GetInitialRotations()

camera = New Camera()

instructions = New OutputInstructionList(cube.Instructions)

UpdateInstructionLabels()

currentCube = New Cube(initalScrambledCube)

For i = 0 To 26

cubieRotations(i) = New OutputBlock

Next

End Sub

Public Sub New(ByVal filePath As String)

InitializeComponent()

Dim fStream As New IO.FileStream(filePath, IO.FileMode.Open)

Dim formatter As New BinaryFormatter()

initalScrambledCube = DirectCast(formatter.Deserialize(fStream), Cube)

instructions = New OutputInstructionList(initalScrambledCube.Instructions)

initalPositions = GetInitialPositions()

initalRotations = GetInitialRotations()

camera = New Camera()

UpdateInstructionLabels()

currentCube = New Cube(initalScrambledCube)

For i = 0 To 26

cubieRotations(i) = New OutputBlock

Next

Try

Dim endPtr = GetCurrentInstructionPtr(filePath)

If endPtr = 0 Then Exit Try

For i = 0 To endPtr - 1

If instructions.currentInstructionPtr < instructions.Count Then

DoNextStage()

instructions.currentInstructionPtr += 1

UpdateInstructionLabels()

End If

Next

btnUndo.Enabled = True

Catch ex As ReadUnsuccessfulException

MsgBox(ex.Message)

End Try

End Sub

Private Sub GlControl1\_Load(sender As Object, e As EventArgs) Handles GlControl1.Load

LoadGraphics()

If shaderProgram.Handle = -1 Then

lblError.Visible = True

Return

End If

LoadGeometry()

End Sub

''' <summary> Loads the shaders and sets up the viewport </summary>

Private Sub LoadGraphics()

GL.Viewport(0, 0, GlControl1.Width, GlControl1.Height)

GL.Enable(EnableCap.DepthTest)

GL.DepthFunc(DepthFunction.Lequal)

sldrAmbient.Value = 23

shaderProgram = New ShaderProgram()

End Sub

''' <summary> Loads the lighting for the scene </summary>

Private Sub LoadLighting()

Dim colour As OpenTK.Graphics.Color4 = New OpenTK.Graphics.Color4(255, 255, 255, 255)

Dim ambient As Vector4 = New Vector4(colour.R / 255, colour.G / 255, colour.B / 255, 1)

Dim lightColorLoc As Integer = GL.GetUniformLocation(shaderProgram.Handle, "lightColor")

GL.Uniform4(lightColorLoc, ambient.X, ambient.Y, ambient.Z, ambient.W)

Static ambientStrength As Single = 0.95

ambientStrength = sldrAmbient.Value / 20

Dim ambientStrengthLoc As Integer = GL.GetUniformLocation(shaderProgram.Handle, "ambientStrength")

GL.Uniform1(ambientStrengthLoc, ambientStrength)

End Sub

''' <summary> Loads the vertices and vertexAttributes </summary>

Private Sub LoadGeometry()

Dim vertices3d As Vertex3D() = CubeVertices()

Dim indices As UInteger() = CubeIndices()

shaderProgram.VAO = GL.GenVertexArray()

shaderProgram.VBO = GL.GenBuffer()

shaderProgram.EBO = GL.GenBuffer()

' Bind the Vertex Array Object

GL.BindVertexArray(shaderProgram.VAO)

' copy our vertices to a buffer for OpenGL to use

GL.BindBuffer(BufferTarget.ArrayBuffer, shaderProgram.VBO)

GL.BufferData(BufferTarget.ArrayBuffer, CType(Vertex3D.SizeInBytes \* vertices3d.Length, IntPtr), vertices3d,

BufferUsageHint.StaticDraw)

' copy the indeces to a buffer for OpenGL to use

GL.BindBuffer(BufferTarget.ElementArrayBuffer, shaderProgram.EBO)

GL.BufferData(BufferTarget.ElementArrayBuffer, CType(4 \* indices.Length, IntPtr), indices, BufferUsageHint.StaticDraw)

' set vertex attribute pointers to access them in the shaders

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)

' Unbind the Vertex Buffer Object

GL.BindBuffer(BufferTarget.ArrayBuffer, 0)

' Unbind the Vertex Array Object

GL.BindVertexArray(0)

GL.PolygonMode(MaterialFace.FrontAndBack, PolygonMode.Fill)

End Sub

Private Function CubeVertices() As Vertex3D()

Return 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))}

End Function

Private Function CubeIndices() As UInteger()

Return 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}

End Function

''' <summary> gets the inital position vectors for each cubie </summary>

Private Function GetInitialPositions() As Vector3()

Dim corners() As Corner = initalScrambledCube.Corners

Dim edges() As Edge = initalScrambledCube.Edges

Dim Positions(26) As Vector3

For i = 0 To 7

Positions(i) = New Vector3 With {

.X = corners(i).Position.x \* (1 + pad) - 0.5,

.Y = corners(i).Position.y \* (1 + pad) - 0.5,

.Z = corners(i).Position.z \* (1 + pad) - 0.5

}

Next

For i = 0 To 11

Positions(i + 8) = New Vector3 With {

.X = edges(i).Position.x \* (1 + pad) - 0.5,

.Y = edges(i).Position.y \* (1 + pad) - 0.5,

.Z = edges(i).Position.z \* (1 + pad) - 0.5

}

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)

Return Positions

End Function

''' <summary> gets the inital rotation quaternions for each cubie </summary>

Private Function GetInitialRotations() As Quaternion()

Dim initalBlocks() As Block = initalScrambledCube.CornersAndEdgesAndMiddles.Copy()

Dim initialCubieRotations(26) As Quaternion

Dim cubie As Block

For i = 0 To 19

cubie = initalBlocks(i)

Dim rotationFirstAxis As Quaternion = GetRotationFirstAxis(cubie)

Dim rotationSecondAxis As Quaternion = GetRotationSecondAxis(cubie, rotationFirstAxis)

initialCubieRotations(i) = rotationSecondAxis \* rotationFirstAxis

Next

For i = 20 To 26

initialCubieRotations(i) = Quaternion.Identity

Next

Return initialCubieRotations

End Function

''' <summary> gets the quaternion that puts the primary face correct </summary>

Private Function GetRotationFirstAxis(ByVal cubie As Block) As Quaternion

Dim rotationVector As Vector3 = New Vector3

Select Case cubie.PrimaryFace

Case FaceColour.W

Select Case cubie.Rotation

Case FaceColour.Y : rotationVector.X = 2

Case FaceColour.R : rotationVector.X = 1

Case FaceColour.O : rotationVector.X = -1

Case FaceColour.G : rotationVector.Z = 1

Case FaceColour.B : rotationVector.Z = -1

End Select

Case FaceColour.Y

Select Case cubie.Rotation

Case FaceColour.W : rotationVector.X = 2

Case FaceColour.R : rotationVector.X = -1

Case FaceColour.O : rotationVector.X = 1

Case FaceColour.G : rotationVector.Z = -1

Case FaceColour.B : rotationVector.Z = 1

End Select

Case FaceColour.R

Select Case cubie.Rotation

Case FaceColour.W : rotationVector.X = -1

Case FaceColour.Y : rotationVector.X = 1

Case FaceColour.O : rotationVector.X = 2

Case FaceColour.G : rotationVector.Y = -1

Case FaceColour.B : rotationVector.Y = 1

End Select

Case FaceColour.O

Select Case cubie.Rotation

Case FaceColour.W : rotationVector.X = 1

Case FaceColour.Y : rotationVector.X = -1

Case FaceColour.R : rotationVector.X = 2

Case FaceColour.G : rotationVector.Y = 1

Case FaceColour.B : rotationVector.Y = -1

End Select

End Select

Dim rotationFirstAxis As Quaternion = New Quaternion With {

.X = rotationVector.X \* Math.Sin(Math.Abs(rotationVector.X) \* MathHelper.PiOver2 / 2),

.Y = rotationVector.Y \* Math.Sin(Math.Abs(rotationVector.Y) \* MathHelper.PiOver2 / 2),

.Z = rotationVector.Z \* Math.Sin(Math.Abs(rotationVector.Z) \* MathHelper.PiOver2 / 2)

}

Dim vecSum As Integer = Math.Abs(rotationVector.X) + Math.Abs(rotationVector.Y) + Math.Abs(rotationVector.Z)

rotationFirstAxis.W = Math.Cos(vecSum \* MathHelper.PiOver2 / 2)

If rotationFirstAxis.Length = rotationFirstAxis.W Then

rotationFirstAxis = Quaternion.Identity

End If

Return rotationFirstAxis

End Function

Private Function GetRotationSecondAxis(cubie As Block, rotationFirstAxis As Quaternion) As Quaternion

Dim whiteNormalAfterFirstRotation As Vector3 = New Vector3(0, 1, 0)

Dim redNormalAfterFirstRotation As Vector3 = New Vector3(0, 0, 1)

RecalculateNormalsAfterFirstRotation(whiteNormalAfterFirstRotation, redNormalAfterFirstRotation,

rotationFirstAxis)

Dim desiredWhiteNormal, desiredRedNormal As Vector3

GetNormalsFromCubie(cubie, desiredWhiteNormal, desiredRedNormal)

Dim angle As Single

Dim axis As Vector3

angle = 0

If whiteNormalAfterFirstRotation = desiredWhiteNormal Then

'rotate around white normal

axis = whiteNormalAfterFirstRotation

angle = GetAngleForInitialRotation(axis, redNormalAfterFirstRotation, desiredRedNormal)

ElseIf redNormalAfterFirstRotation = desiredRedNormal Then

'rotate around red normal

axis = redNormalAfterFirstRotation

angle = GetAngleForInitialRotation(axis, whiteNormalAfterFirstRotation, desiredWhiteNormal)

End If

Dim rotationSecondAxis As Quaternion

rotationSecondAxis = Quaternion.Identity

If angle <> 0 Then

rotationSecondAxis.X = Math.Abs(axis.X) \* Math.Sin(angle / 2)

rotationSecondAxis.Y = Math.Abs(axis.Y) \* Math.Sin(angle / 2)

rotationSecondAxis.Z = Math.Abs(axis.Z) \* Math.Sin(angle / 2)

rotationSecondAxis.W = Math.Cos(angle / 2)

Else

rotationSecondAxis = Quaternion.Identity

End If

Return rotationSecondAxis

End Function

Private Sub RecalculateNormalsAfterFirstRotation(ByRef whiteNormal As Vector3, ByRef redNormal As Vector3, ByVal firstRotation As Quaternion)

Dim rotationMatrix As Matrix4

rotationMatrix = Matrix4.CreateTranslation(New Vector3(-0.5, -0.5, -0.5))

rotationMatrix \*= Matrix4.CreateFromQuaternion(firstRotation)

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 Sub GetNormalsFromCubie(ByVal cubie As Block, ByRef whiteNormal As Vector3, ByRef redNormal As Vector3)

Dim secondaryRotation, secondaryFace As FaceColour

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 <> cubie.PrimaryFace Then

secondaryFace = colour

Exit For

End If

Next

For Each face As FaceColour In initalScrambledCube.FaceColours

If cubie.CorrectRotation(secondaryFace, face) And cubie.OnFace(face, initalScrambledCube) Then

secondaryRotation = face

Exit For

End If

Next

End If

Dim whiteSet, redSet As Boolean

whiteSet = False

redSet = False

Select Case cubie.PrimaryFace

Case FaceColour.W, FaceColour.Y

Select Case cubie.Rotation

Case FaceColour.W : whiteNormal = New Vector3(0, 1, 0)

Case FaceColour.Y : whiteNormal = New Vector3(0, -1, 0)

Case FaceColour.R : whiteNormal = New Vector3(0, 0, 1)

Case FaceColour.O : whiteNormal = New Vector3(0, 0, -1)

Case FaceColour.G : whiteNormal = New Vector3(-1, 0, 0)

Case FaceColour.B : whiteNormal = New Vector3(1, 0, 0)

End Select

If cubie.PrimaryFace = FaceColour.Y Then

whiteNormal = -whiteNormal

End If

whiteSet = True

Case FaceColour.R, FaceColour.O

Select Case cubie.Rotation

Case FaceColour.W : redNormal = New Vector3(0, 1, 0)

Case FaceColour.Y : redNormal = New Vector3(0, -1, 0)

Case FaceColour.R : redNormal = New Vector3(0, 0, 1)

Case FaceColour.O : redNormal = New Vector3(0, 0, -1)

Case FaceColour.G : redNormal = New Vector3(-1, 0, 0)

Case FaceColour.B : redNormal = New Vector3(1, 0, 0)

End Select

If cubie.PrimaryFace = FaceColour.O Then

redNormal = -redNormal

End If

redSet = True

End Select

Select Case secondaryFace

Case FaceColour.R, FaceColour.O

Select Case secondaryRotation

Case FaceColour.W : redNormal = New Vector3(0, 1, 0)

Case FaceColour.Y : redNormal = New Vector3(0, -1, 0)

Case FaceColour.R : redNormal = New Vector3(0, 0, 1)

Case FaceColour.O : redNormal = New Vector3(0, 0, -1)

Case FaceColour.G : redNormal = New Vector3(-1, 0, 0)

Case FaceColour.B : redNormal = New Vector3(1, 0, 0)

End Select

If secondaryFace = FaceColour.O Then

redNormal = -redNormal

End If

redSet = True

Case FaceColour.G, FaceColour.B

Dim greenNormal As Vector3

Select Case secondaryRotation

Case FaceColour.W : greenNormal = New Vector3(0, 1, 0)

Case FaceColour.Y : greenNormal = New Vector3(0, -1, 0)

Case FaceColour.R : greenNormal = New Vector3(0, 0, 1)

Case FaceColour.O : greenNormal = New Vector3(0, 0, -1)

Case FaceColour.G : greenNormal = New Vector3(-1, 0, 0)

Case FaceColour.B : greenNormal = New Vector3(1, 0, 0)

End Select

If secondaryFace = FaceColour.B Then

greenNormal = -greenNormal

End If

If whiteSet Then

redNormal = Vector3.Cross(whiteNormal, greenNormal)

ElseIf redSet Then

whiteNormal = -Vector3.Cross(redNormal, greenNormal)

Else

Throw New NotImplementedException("GetNormalsFromCubie: primary normal not set!")

End If

End Select

End Sub

Private Function GetAngleForInitialRotation(axis As Vector3, currentPosition As Vector3, desiredPosition As Vector3) As Single

Dim angle As Single

If currentPosition = desiredPosition Then

angle = 0

ElseIf Math.Abs(Vector3.Dot(currentPosition, desiredPosition)) = 1 Then

angle = MathHelper.Pi

Else

'collapse the vectors onto 1 2D plane, perpendicular to axis of rotation

Dim current2d, correct2d As Vector2

If axis.X <> 0 Then

current2d = New Vector2(-currentPosition.Z, currentPosition.Y)

correct2d = New Vector2(-desiredPosition.Z, desiredPosition.Y)

ElseIf axis.Y <> 0 Then

current2d = New Vector2(currentPosition.X, -currentPosition.Z)

correct2d = New Vector2(desiredPosition.X, -desiredPosition.Z)

ElseIf axis.Z <> 0 Then

current2d = New Vector2(currentPosition.X, currentPosition.Y)

correct2d = New Vector2(desiredPosition.X, desiredPosition.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

Return angle

End Function

''' <summary> Called by GlControl1.Invalidate() - updates the GL viewport </summary>

Private Sub GlControl1\_Paint(sender As Object, e As PaintEventArgs) Handles GlControl1.Paint

GL.ClearColor(Color.CornflowerBlue)

GL.Clear(ClearBufferMask.ColorBufferBit Or ClearBufferMask.DepthBufferBit)

If shaderProgram.Handle = -1 Then Return

Static previousFrame As Decimal

UpdateDeltaTime(previousFrame)

Dim camX As Single = Math.Sin(camera.Rotation / 10) \* Camera.RADIUS

Dim camZ As Single = Math.Cos(camera.Rotation / 10) \* Camera.RADIUS

Dim view As Matrix4 = Matrix4.LookAt(New Vector3(camX, camera.Pitch, camZ), New Vector3(0, 0, 0),

New Vector3(0, 1, 0))

'sends view matrix to shaders

GL.Uniform3(GL.GetUniformLocation(shaderProgram.Handle, "viewPos"), camX, camera.Pitch, camZ)

RenderCube(view, projection)

camera.Update(False, deltaTime, isMouseDown)

previousFrame = DateTime.Now.Ticks / 10000

GlControl1.SwapBuffers() 'swaps the buffer being drawn to and the buffer being shown (updates the viewport)

GlControl1.Invalidate() ' calls paint

End Sub

Private Sub UpdateDeltaTime(ByVal previousFrame As Decimal)

Dim currentFrame As Decimal

Static firsttime As Boolean = True

If firsttime Then

deltaTime = 5

firsttime = False

End If

Do

currentFrame = DateTime.Now.Ticks / 10000

deltaTime = currentFrame - previousFrame

Loop While deltaTime = 0

End Sub

''' <summary> Renders a cube onto the secondary buffer </summary>

Private Sub RenderCube(ByRef view As Matrix4, ByRef projection As Matrix4)

shaderProgram.Use()

LoadLighting()

GL.UniformMatrix4(GL.GetUniformLocation(shaderProgram.Handle, "projection"), False, projection)

GL.UniformMatrix4(GL.GetUniformLocation(shaderProgram.Handle, "view"), False, view)

GL.Enable(EnableCap.DepthTest)

GL.DepthFunc(DepthFunction.Less)

Static displayAnglePercent As Double = 0.0

Dim rotationSpeed As Single = sldrSpeed.Value / 1000

If showingRotation Then

displayAnglePercent += deltaTime \* rotationSpeed

If displayAnglePercent >= 1 Then

displayAnglePercent = 1

showingRotation = False

btnNext.Enabled = True

btnUndo.Enabled = True

End If

Else

displayAnglePercent = 0

End If

For i = 0 To 26

RenderCubie(i, displayAnglePercent)

Next

End Sub

''' <summary> renders the cubie represented by a given index </summary>

Private Sub RenderCubie(ByVal index As Integer, ByVal displayAnglePercent As Double)

GL.BindVertexArray(shaderProgram.VAO)

GL.Uniform1(GL.GetUniformLocation(shaderProgram.Handle, "pad"), pad)

Dim model As Matrix4 = Matrix4.CreateTranslation(initalPositions(26))

model \*= Matrix4.CreateFromQuaternion(initalRotations(index))

model \*= Matrix4.CreateTranslation(-initalPositions(26))

model \*= Matrix4.CreateTranslation(initalPositions(index))

'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)

model \*= cubieRotations(index).Rotations.GetTotalMatrix(displayAnglePercent)

GL.UniformMatrix4(GL.GetUniformLocation(shaderProgram.Handle, "model"), False, model)

GL.DrawElements(PrimitiveType.Triangles, 36, DrawElementsType.UnsignedInt, 0)

End Sub

Private Sub btnUndo\_Hover(sender As Object, e As EventArgs) Handles btnUndo.MouseEnter, btnUndo.MouseHover

lblPrev.ForeColor = Color.Red

End Sub

Private Sub btnNext\_Hover(sender As Object, e As EventArgs) Handles btnNext.MouseEnter, btnNext.MouseHover

lblNext.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 btnNext\_MouseLeave(sender As Object, e As EventArgs) Handles btnNext.MouseLeave

lblNext.ForeColor = Color.Black

End Sub

Private Sub UpdateInstructionLabels()

If instructions.PreviousInstruction IsNot Nothing Then

lblPrev.Text = instructions.PreviousInstruction.ToString()

Else

lblPrev.Text = "- -"

btnUndo.Enabled = False

End If

If instructions.CurrentInstruction IsNot Nothing Then

lblNext.Text = instructions.CurrentInstruction.ToString()

Else

lblNext.Text = "- -"

btnNext.Enabled = False

End If

lblPrev.Refresh()

lblNext.Refresh()

End Sub

Private Sub GlControl1\_MouseDown(sender As Object, e As MouseEventArgs) Handles GlControl1.MouseDown

Dim mousePos As New Mouse

If Not isMouseDown Then

mousePos.X = OpenTK.Input.Mouse.GetCursorState.X

mousePos.Y = OpenTK.Input.Mouse.GetCursorState.Y

End If

isMouseDown = True

camera.Update(True, deltaTime, isMouseDown, mousePos)

End Sub

Private Sub GlControl1\_MouseUp(sender As Object, e As MouseEventArgs) Handles GlControl1.MouseUp

isMouseDown = False

End Sub

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

If shaderProgram.Handle <> -1 Then btnUndo.Enabled = False

If instructions.currentInstructionPtr > 0 Then

UndoLastStage()

instructions.currentInstructionPtr -= 1

UpdateInstructionLabels()

End If

If lblNext.Text <> "- -" Then

btnNext.Enabled = True

End If

End Sub

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

If shaderProgram.Handle <> -1 Then btnNext.Enabled = False

If instructions.currentInstructionPtr < instructions.Count Then

DoNextStage()

instructions.currentInstructionPtr += 1

UpdateInstructionLabels()

End If

If lblPrev.Text <> "- -" Then

btnUndo.Enabled = True

End If

End Sub

Private Sub DoNextStage()

DoInstruction(instructions.CurrentInstruction)

End Sub

Private Sub UndoLastStage()

Dim reverseInstruction As Instruction

reverseInstruction = instructions.PreviousInstruction.Copy()

reverseInstruction.Direction = -reverseInstruction.Direction

DoInstruction(reverseInstruction)

End Sub

Private Sub DoInstruction(ByVal instruction As Instruction)

showingRotation = True

If instruction.GetType() = GetType(FaceInstruction) Then

RotateFace(instruction.Move, instruction.Direction)

ElseIf instruction.GetType = GetType(CubeInstruction) Then

RotateCube(instruction.Move, instruction.Direction)

Else

Throw New ArgumentException("Invalid instruction type")

End If

End Sub

''' <summary> addds the rotation matrices to the correct rotation stores in order to rotate a given face </summary>

Private Sub RotateFace(ByVal face As MoveFaces, ByVal direction As Direction)

Dim axis As Vector3

Select Case face

Case MoveFaces.RIGHT : axis = New Vector3(1, 0, 0)

Case MoveFaces.LEFT : axis = New Vector3(-1, 0, 0)

Case MoveFaces.TOP : axis = New Vector3(0, 1, 0)

Case MoveFaces.BOTTOM : axis = New Vector3(0, -1, 0)

Case MoveFaces.FRONT : axis = New Vector3(0, 0, 1)

Case MoveFaces.BACK : axis = New Vector3(0, 0, -1)

Case Else

Throw New ArgumentException("invalid face")

End Select

For i = 0 To 26

If Not currentCube.CornersAndEdgesAndMiddles(i).OnFace(currentCube.FaceColours(face), currentCube) Then Continue For

cubieRotations(i).Rotations.Add(axis, IIf(Math.Abs(direction) =

Direction.HALF\_TURN, -direction, direction) \* MathHelper.PiOver2)

cubieRotations(i).BeingRotated = True

currentCube.RotateFace(currentCube.FaceColours(face), direction)

Next

End Sub

''' <summary> addds the rotation matrices to the correct rotation stores in order to rotate the cube </summary>

Private Sub RotateCube(ByVal axis As Axis, ByVal direction As Direction)

Dim axisVector As Vector3

Select Case axis

Case Axis.X : axisVector = New Vector3(1, 0, 0)

Case Axis.Y : axisVector = New Vector3(0, 1, 0)

Case Axis.Z : axisVector = New Vector3(0, 0, 1)

Case Else

Throw New ArgumentException("Invalid axis")

End Select

Dim angle As Single = MathHelper.PiOver2 \* IIf(Math.Abs(direction) = Direction.HALF\_TURN, -direction, direction)

For i = 0 To 26

cubieRotations(i).Rotations.Add(axisVector, angle)

Next

currentCube.Rotate(direction, axis)

End Sub

Private Sub UpdateOrientation(ByVal axis As Axis, ByVal direction As Direction)

Dim faces() As FaceColour = FacesAroundAxisClockwise(axis)

If axis <> Axis.Y Then currentCube.TopFace = faces((Array.IndexOf(faces, currentCube.TopFace) + 4 + direction) Mod 4)

If axis <> Axis.Z Then currentCube.FrontFace = faces((Array.IndexOf(faces, currentCube.FrontFace) + 4 + direction) Mod 4)

End Sub

Private Function FacesAroundAxisClockwise(ByVal axis As Axis) As FaceColour()

Dim faceColours() As FaceColour = Helpers.GetFaceColoursFromOrientation(currentCube.CurrentOrientation)

Select Case axis

Case Axis.X

Return {faceColours(MoveFaces.FRONT), faceColours(MoveFaces.TOP), faceColours(MoveFaces.BACK),

faceColours(MoveFaces.BOTTOM)}

Case Axis.Y

Return {faceColours(MoveFaces.FRONT), faceColours(MoveFaces.LEFT), faceColours(MoveFaces.BACK),

faceColours(MoveFaces.RIGHT)}

Case Axis.Z

Return {faceColours(MoveFaces.TOP), faceColours(MoveFaces.RIGHT), faceColours(MoveFaces.BOTTOM),

faceColours(MoveFaces.LEFT)}

Case Else

Throw New ArgumentException("Invalid Axis")

End Select

End Function

Private Sub \_3DOutput\_FormClosed(sender As Object, e As FormClosedEventArgs) Handles MyBase.FormClosed

shaderProgram.Dispose()

End Sub

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

Try

GlControl1.Visible = False

SaveRoutine()

Catch ex As SaveCancelledException

GlControl1.Visible = True

Return

End Try

Dim main As New MainMenu

main.Show()

Me.Close()

End Sub

Private Sub SaveRoutine()

Dim save As MsgBoxResult = MsgBox("Do you want to save?" & vbNewLine & vbNewLine &

"Yes = save and exit" & vbNewLine &

"No = exit without saving" & vbNewLine &

"Cancel = don't exit", MsgBoxStyle.YesNoCancel, "Save?")

If save = MsgBoxResult.Yes Then

Try

SaveCube()

Catch ex As TaskCanceledException

Throw New SaveCancelledException

Catch ex As WriteUnsuccessfulException

btnMain.PerformClick()

Throw New SaveCancelledException

End Try

ElseIf save = MsgBoxResult.Cancel Then

Throw New SaveCancelledException

End If

End Sub

Private Sub SaveCube()

Dim fileBrowser As New SaveFileDialog()

fileBrowser.Filter = "cube files (\*.cube)|\*.cube"

fileBrowser.ShowDialog()

Dim filePath As String = fileBrowser.FileName

If filePath = Nothing Then Throw New TaskCanceledException

initalScrambledCube.Save(filePath)

WriteCurrentInstructionPtr(filePath)

End Sub

Private Sub WriteCurrentInstructionPtr(ByVal filePath As String)

If filePath.Substring(filePath.Length - 5) <> ".cube" Then filePath += ".cube"

filePath += ".ptr"

Try

Dim fStream As New IO.FileStream(filePath, IO.FileMode.Create)

Dim formatter As New BinaryFormatter()

formatter.Serialize(fStream, instructions.currentInstructionPtr)

fStream.Close()

Catch ex As Exception

Throw New WriteUnsuccessfulException("CurrentInstructionPtr attempted to be written before cube file")

End Try

End Sub

Private Function GetCurrentInstructionPtr(ByVal filePath As String) As Integer

filePath += ".ptr"

Try

Dim fStream As New IO.FileStream(filePath, IO.FileMode.Open)

Dim formatter As New BinaryFormatter()

Return formatter.Deserialize(fStream)

Catch ex As Exception

Throw New ReadUnsuccessfulException(".ptr file not found, cannot load saved instruction")

End Try

End Function

End Class

## Camera.vb

Public Class Camera

Public Const FRONT\_ANGLE As Single = -63.0

Public Const RADIUS As Single = 10

Public Property Pitch As Single

Public Property Rotation As Single

Public Sub New()

Rotation = -60.0

Pitch = 2

End Sub

Public Sub Update(ByVal PositionUpdated As Boolean, ByVal deltaTime As Single, ByVal mouseDown As Boolean, Optional MousePos As Mouse = Nothing)

Static StoredMousePos As New Mouse

If PositionUpdated = True Then

StoredMousePos.X = MousePos.X

StoredMousePos.Y = MousePos.Y

End If

Dim cameraSpeed As Single = 0.07 \* deltaTime

If Not mouseDown Then

If Pitch > 2 + (cameraSpeed \* 2.2) Then

Pitch -= cameraSpeed \* 2.2

ElseIf Pitch < 2 - (cameraSpeed \* 2.2) Then

Pitch += cameraSpeed \* 2.2

ElseIf Pitch >= 2 - (cameraSpeed \* 2.2) And Pitch <= 2 + (cameraSpeed \* 2.2) Then

Pitch = 2

End If

End If

If mouseDown 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.X

Dim yOffset As Integer = yPosition - StoredMousePos.Y

Dim sensitivity As Single = 0.1

StoredMousePos.X = xPosition

StoredMousePos.Y = yPosition

If Pitch <= 25 And Pitch >= -20 Then

Pitch += (yOffset \* sensitivity)

ElseIf Pitch > 25 Then

Pitch = 25

ElseIf Pitch < -20 Then

Pitch = -20

End If

If (Rotation - (xOffset \* sensitivity \* 0.5)) > (FRONT\_ANGLE - 6) And

(Rotation - (xOffset \* sensitivity \* 0.5)) < (FRONT\_ANGLE + 6) Then

Rotation -= (xOffset \* sensitivity \* 0.5)

End If

End If

End Sub

End Class

## Vertex3D.vb

Imports OpenTK

Structure Vertex3D

Public Shared ReadOnly SizeInBytes As Integer = (Vector3.SizeInBytes + Vector4.SizeInBytes + Vector3.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

End Structure

## ShaderProgram.vb

Imports OpenTK.Graphics.OpenGL

Imports System.IO

Public Class ShaderProgram

Private program As Integer

Public ReadOnly Property Handle() As Integer

Get

Return program

End Get

End Property

Public Property VBO As Integer

Public Property VAO As Integer

Public Property EBO As Integer

Sub New()

Dim versionNo As Integer = CInt(CStr(GL.GetString(StringName.Version)(0) + GL.GetString(StringName.Version)(2)))

Dim versionString As String = "330"

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 + My.Resources.vertexShader)

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 + My.Resources.fragmentShader)

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 Sub Dispose()

If Handle = -1 Then Return

GL.DeleteVertexArray(VAO)

GL.DeleteBuffer(VBO)

GL.DeleteBuffer(EBO)

End Sub

End Class

## RotationStore.vb

Imports OpenTK

Public Class RotationStore

Public Property PreviousRotations As Matrix4 = Matrix4.Identity

Public Property CurrentAxis As Vector3

Public Property CurrentMaxAngle As Single

Public Sub Add(ByVal axis As Vector3, ByVal maxAngle As Single)

If CurrentAxis <> Nothing Then

PreviousRotations \*= Matrix4.CreateFromAxisAngle(CurrentAxis, CurrentMaxAngle)

End If

CurrentAxis = axis

CurrentMaxAngle = maxAngle

End Sub

Public Function GetTotalMatrix(ByVal displayAnglePercentage As Single) As Matrix4

If displayAnglePercentage >= 1 Then FinishCurrentRotation()

Dim rotationMatrix As Matrix4 = Matrix4.Identity

rotationMatrix \*= PreviousRotations

If CurrentAxis <> Nothing Then

rotationMatrix \*= Matrix4.CreateFromAxisAngle(CurrentAxis, displayAnglePercentage \* CurrentMaxAngle)

End If

Return rotationMatrix

End Function

Private Sub FinishCurrentRotation()

If CurrentAxis <> Nothing Then

PreviousRotations \*= Matrix4.CreateFromAxisAngle(CurrentAxis, CurrentMaxAngle)

End If

CurrentAxis = Nothing

CurrentMaxAngle = Nothing

End Sub

End Class

## OutputBlock.vb

Public Class OutputBlock

Public Property Rotations As New RotationStore

Public Property BeingRotated As Boolean

Public Sub New()

BeingRotated = False

End Sub

End Class

## OutputInstructionList

Public Class OutputInstructionList

Inherits List(Of Instruction)

Public Property currentInstructionPtr As Integer

Public Sub New(ByVal instructions As InstructionList)

For i = 0 To instructions.Count - 1

Me.Add(instructions.ElementAt(i))

Next

End Sub

Public ReadOnly Property CurrentInstruction() As Instruction

Get

Try

Return ElementAt(currentInstructionPtr)

Catch

Return Nothing

End Try

End Get

End Property

Public ReadOnly Property PreviousInstruction() As Instruction

Get

Try

Return ElementAt(currentInstructionPtr - 1)

Catch

Return Nothing

End Try

End Get

End Property

End Class

## Mouse.vb

Public Class Mouse

Private \_x As Integer

Public Property X() As Integer

Get

Return \_x

End Get

Set(ByVal value As Integer)

\_x = value

End Set

End Property

Private \_y As Integer

Public Property Y() As Integer

Get

Return \_y

End Get

Set(ByVal value As Integer)

\_y = value

End Set

End Property

End Class

## Exceptions.vb

Public Class StageNotSuccessfulException

Inherits Exception

Public Sub New(ByVal message As String)

MyBase.New(message)

End Sub

Public Sub New()

MyBase.New()

End Sub

End Class

Public Class WriteUnsuccessfulException

Inherits Exception

Public Sub New(ByVal message As String)

MyBase.New(message)

End Sub

Public Sub New()

MyBase.New()

End Sub

End Class

Public Class ReadUnsuccessfulException

Inherits Exception

Public Sub New(ByVal message As String)

MyBase.New(message)

End Sub

Public Sub New()

MyBase.New()

End Sub

End Class

Public Class SaveCancelledException

Inherits Exception

Public Sub New(ByVal message As String)

MyBase.New(message)

End Sub

Public Sub New()

MyBase.New()

End Sub

End Class

## VertexShader.txt

//

#if \_\_VERSION\_\_ == 330

layout (location = 0) in vec3 position; // Vertex Attribute Pointer 0

layout (location = 1) in vec4 color; // VattPointer 1

layout (location = 2) in vec3 normal; // VAttPointer 2

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

out vec4 vertexColor;

out vec3 pos;

void main()

{

gl\_Position = projection \* view \* model \* vec4(position, 1.0f);

vertexColor = color;

pos = position;

}

#endif

#if \_\_VERSION\_\_ == 120

attribute vec3 position; // Vertex Attribute Pointer 0

attribute vec4 color; // VAttPointer 1

attribute vec3 normal; // VAttPointer 2

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

varying vec4 vertexColor;

varying vec3 pos;

void main()

{

gl\_Position = projection \* view \* model \* vec4(position, 1.0f);

vertexColor = color;

pos = position;

}

#endif

## FragmentShader.txt

//

#if \_\_VERSION\_\_ == 330

// Input variables from the vertex shader (same name and type)

in vec4 vertexColor;

in vec3 pos;

uniform vec4 lightColor;

uniform float ambientStrength;

uniform float pad;

uniform mat4 colourMat;

out vec4 color;

void main()

{

vec3 worldPosition = vec3(colourMat \* vec4(pos, 1.0f));

if (((worldPosition.x <= -1.499999f - pad) || (worldPosition.x >= 1.499999f + pad)) ||

((worldPosition.y <= -1.499999f - pad) || (worldPosition.y >= 1.499999f + pad)) ||

((worldPosition.z <= -1.499999f - pad) || (worldPosition.z >= 1.499999f + pad))){

vec3 ambient = ambientStrength \* lightColor.xyz;

vec3 result = ambient \* vertexColor.xyz;

color = vec4(result, 1.0f);

}

}

#endif

#if \_\_VERSION\_\_ == 120

// Input variables from the vertex shader (same name and type)

varying vec4 vertexColor;

varying vec3 pos;

uniform vec4 lightColor;

uniform float pad;

uniform float ambientStrength;

uniform mat4 colourMat;

void main()

{

vec3 worldPosition = vec3(colourMat \* vec4(pos, 1.0f));

if (((worldPosition.x <= -1.499999f - pad) || (worldPosition.x >= 1.499999f + pad)) ||

((worldPosition.y <= -1.499999f - pad) || (worldPosition.y >= 1.499999f + pad)) ||

((worldPosition.z <= -1.499999f - pad) || (worldPosition.z >= 1.499999f + pad))){

vec3 ambient = ambientStrength \* lightColor.xyz;

vec3 result = ambient \* vertexColor.xyz ;

gl\_FragColor = vec4(result, 1.0f);

}

}

#endif