```
## Rubik's Cube Simulation
## MTH 44101 Fall 2011-Spring 2013
## A. O. Hausknecht
## Department of Mathematics
## UMass Dartmouth
## Udated for Visual Python 6.0.4 Spring 2013
from __future__ import division, print_function
from visual import *
from visual.controls import *
from visual.graph import *
import wx
import  wx.lib.buttons  as  buttons

class RubikColorGlobals(object):
    """ A class to to share the colors used throughout the program."""
    _cubeColor = color.black
    _orange = (1, 0.2, 0)
    _orangeStr = '#ff8c00'
    _markerColor = (0.5, 0, 0.5)
    _faceColors = { 'left': color.blue, 'front': color.white,
                    'right': color.green, 'back': color.yellow,
                       'up':color.red,    'down': _orange}

    _faceColorNames = { color.blue: 'Blue',   color.white: 'White',
                        color.green: 'Green', color.yellow: 'Yellow',
                          color.red: 'Red',        _orange: _orangeStr}

    def __init__(self):
        pass



class RubikCubePart(RubikColorGlobals):
    '''A generic part of a rubik's cube.'''

    partTypes = {'FLU': 'frontLeftUpCorner',     'FRU': 'frontRightUpCorner',
                 'FLD': 'frontLeftDownCorner',   'FRD': 'frontRightDownCorner',
                 'BLU': 'backLeftUpCorner',      'BRU': 'backRightUpCorner',
                 'BLD': 'backLeftDownCorner',    'BRD': 'backRightDownCorner',

                 'FLE': 'frontLeftEdge',         'FRE': 'frontRightEdge',
                 'FUE': 'frontUpEdge',           'FDE': 'frontDownEdge',

                 'BLE': 'backLeftEdge',          'BRE': 'backRightEdge',
                 'BUE': 'backUpEdge',            'BDE': 'backDownEdge',

                 'LUE': 'leftUpEdge',            'RUE': 'rightUpEdge',
                 'LDE': 'leftDownEdge',          'RDE': 'rightDownEdge',

                 'FCE': 'frontCenter',           'BCE': 'backCenter',
                 'LCE': 'frontCenter',           'RCE': 'backCenter',
                 'UCE': 'upCenter',              'DCE': 'downCenter' }

    __stickerPos = 0.5025
    __stickerSize = 0.9
    __stickerThickness = 0.05
    __markerPos  = 0.5525
    __markerThickness = 0.05
    __markerRadius = 0.3
```

```
65      def __init__(self):
66
67          self.partType = None
68          self.itsFrame = frame()
69          self.cube = box( frame = self.itsFrame,
70                           pos = (0, 0, 0),
71                         length = 1,
72                         height = 1,
73                          width = 1,
74                          color = RubikCubePart._cubeColor)
75          self.stickers = [];
76          self.fSticker = None; self.bSticker = None
77          self.lSticker = None; self.rSticker = None
78          self.uSticker = None; self.dSticker = None
79          self.markers = []
80          self.fMarker = None; self.bMarker = None
81          self.lMarker = None; self.rMarker = None
82          self.uMarker = None; self.dMarker = None
83
84      def _createFrontSticker(self, mark):
85          sticker = box( frame = self.itsFrame,
86                           pos = (0, 0, RubikCubePart.__stickerPos),
87                       length = RubikCubePart.__stickerSize,
88                       height = RubikCubePart.__stickerSize,
89                        width = RubikCubePart.__stickerThickness,
90                        color = RubikCubePart._faceColors['front'])
91          sticker.partName = 'fSticker'
92          self.stickers.append(sticker); self.fSticker = sticker
93
94          marker = cylinder( frame = self.itsFrame,
95                             pos = (0, 0, RubikCubePart.__markerPos),
96                            axis = (0, 0, 1),
97                          length = RubikCubePart.__markerThickness,
98                          radius = RubikCubePart.__markerRadius,
99                           color = RubikCubePart._markerColor)
100         marker.partName = 'fMarker'
101         marker.visible = mark
102         self.fMarker = marker; self.markers.append(marker)
103
104     def _createBackSticker(self, mark):
105         sticker = box( frame = self.itsFrame,
106                          pos = (0, 0, -RubikCubePart.__stickerPos),
107                      length = RubikCubePart.__stickerSize,
108                      height = RubikCubePart.__stickerSize,
109                       width = RubikCubePart.__stickerThickness,
110                       color = RubikCubePart._faceColors['back'])
111         sticker.partName = 'bSticker'
112         self.stickers.append(sticker); self.fSticker = sticker
113
114         marker = cylinder( frame = self.itsFrame,
115                            pos = (0, 0, -RubikCubePart.__markerPos),
116                           axis = (0,0,1),
117                         length = RubikCubePart.__markerThickness,
118                         radius = RubikCubePart.__markerRadius,
119                          color = RubikCubePart._markerColor)
120         marker.partName = 'bMarker'
121         marker.visible = mark
122         self.fMarker = marker; self.markers.append(marker)
123
124
125
126
127
128
```

```
129     def _createLeftSticker(self, mark):
130         sticker = box( frame = self.itsFrame,
131                           pos = (-RubikCubePart.__stickerPos, 0, 0),
132                        length = RubikCubePart.__stickerThickness,
133                        height = RubikCubePart.__stickerSize,
134                         width = RubikCubePart.__stickerSize,
135                         color = RubikCubePart._faceColors['left'] )
136         sticker.partName = 'lSticker'
137         self.lSticker = sticker; self.stickers.append(sticker)
138 
139         marker = cylinder(frame = self.itsFrame,
140                             pos = (-RubikCubePart.__markerPos, 0, 0),
141                            axis = (1,0,0),
142                          length = RubikCubePart.__markerThickness,
143                          radius = RubikCubePart.__markerRadius,
144                           color = RubikCubePart._markerColor)
145         marker.partName = 'lMarker'
146         marker.visible = mark
147         self.lMarker = marker; self.markers.append(marker)
148 
149     def _createRightSticker(self, mark):
150         sticker = box( frame = self.itsFrame,
151                           pos = (RubikCubePart.__stickerPos, 0, 0),
152                        length = RubikCubePart.__stickerThickness,
153                        height = RubikCubePart.__stickerSize,
154                         width = RubikCubePart.__stickerSize,
155                         color = RubikCubePart._faceColors['right'] )
156         sticker.partName = 'rSticker'
157         self.rSticker = sticker; self.stickers.append(sticker)
158 
159         marker = cylinder( frame = self.itsFrame,
160                             pos = (RubikCubePart.__markerPos, 0, 0),
161                            axis = (1,0,0),
162                          length = RubikCubePart.__markerThickness,
163                          radius = RubikCubePart.__markerRadius,
164                           color = RubikCubePart._markerColor)
165         marker.partName = 'rMarker'
166         marker.visible = mark
167         self.rMarker = marker; self.markers.append(marker)
168 
169     def _createUpSticker(self, mark):
170         sticker = box( frame = self.itsFrame,
171                           pos = (0, RubikCubePart.__stickerPos, 0),
172                        length = RubikCubePart.__stickerSize,
173                        height = RubikCubePart.__stickerThickness,
174                         width = RubikCubePart.__stickerSize,
175                         color =  RubikCubePart._faceColors['up'])
176         sticker.partName = 'uSticker'
177         self.uSticker = sticker
178         self.stickers.append(sticker)
179 
180         marker = cylinder( frame = self.itsFrame,
181                             pos = (0, RubikCubePart.__markerPos, 0),
182                            axis = (0,1,0),
183                          length = RubikCubePart.__markerThickness,
184                          radius = RubikCubePart.__markerRadius,
185                           color = RubikCubePart._markerColor)
186         marker.partName = 'uMarker'
187         marker.visible = mark
188         self.uMarker = marker; self.markers.append(marker)
189 
190 
191 
192 
```

```
193     def _createDownSticker(self, mark):
194         sticker = box( frame = self.itsFrame,
195                           pos = (0, -RubikCubePart.__stickerPos, 0),
196                        length = RubikCubePart.__stickerSize,
197                        height = RubikCubePart.__stickerThickness,
198                         width = RubikCubePart.__stickerSize,
199                         color =  RubikCubePart._faceColors['down'] )
200         sticker.partName = 'dSticker'
201         self.dSticker = sticker; self.stickers.append(sticker)
202 
203         marker = cylinder( frame = self.itsFrame,
204                               pos = (0, -RubikCubePart.__markerPos, 0),
205                              axis = (0,1,0),
206                            length = RubikCubePart.__markerThickness,
207                            radius = RubikCubePart.__markerRadius,
208                             color = RubikCubePart._markerColor)
209         marker.partName = 'dMarker'
210         marker.visible = mark
211         self.dMarker = marker; self.markers.append(marker)
212 
213 
214     def setColors(self, colors):
215         for s in self.stickers:
216             if s.partName == 'fSticker':
217                 s.color = colors['front']
218             elif s.partName == 'bSticker':
219                 s.color = colors['back']
220             elif s.partName == 'rSticker':
221                 s.color = colors['right']
222             elif s.partName == 'lSticker':
223                 s.color = colors['left']
224             elif s.partName == 'uSticker':
225                 s.color = colors['up']
226             elif s.partName == 'dSticker':
227                 s.color = colors['down']
228 
229 
230     def showHideMarkers(self, show):
231          for m in self.markers:
232             m.visible = show
233 
234 
235 class CornerPart(RubikCubePart):
236     '''A corner part of a Rubik's cube.'''
237 
238     def __init__(self, which, mark=False):
239         RubikCubePart.__init__(self)
240         self.partType = RubikCubePart.partTypes[which]
241         if which == 'FLU':
242             self._createFrontCorner(True, True, mark)
243         elif which == 'FRU':
244             self._createFrontCorner(False, True, mark)
245         elif which == 'FLD':
246             self._createFrontCorner(True, False, mark)
247         elif which == 'FRD':
248             self._createFrontCorner(False, False, mark)
249         elif which == 'BLU':
250             self._createBackCorner(True, True, mark)
251         elif which == 'BRU':
252             self._createBackCorner(False, True, mark)
253         elif which == 'BLD':
254             self._createBackCorner(True, False, mark)
255         elif which == 'BRD':
256             self._createBackCorner(False, False, mark)
```

```
257
258     def _createFrontCorner(self, isLeft, isTop, mark):
259         '''Create a front corner a Rubik's cube.'''
260         self._createFrontSticker(mark)
261         if isLeft:
262             self._createLeftSticker(mark)
263             cX = -1
264         else:
265             self._createRightSticker(mark)
266             cX = 1
267 
268         if isTop:
269             self._createUpSticker(mark)
270             cY = 1
271         else:
272             self._createDownSticker(mark)
273             cY = -1
274 
275         self.itsFrame.pos = (cX, cY, 1)
276         self.initPos = (cX, cY, 1)
277         self.initAxis = vector(self.itsFrame.axis)
278 
279 
280     def _createBackCorner(self, isLeft, isTop, mark):
281         '''Create a back corner a Rubik's cube.'''
282         self._createBackSticker(mark)
283         if isLeft:
284             self._createLeftSticker(mark)
285             cX = -1
286         else:
287             self._createRightSticker(mark)
288             cX = 1
289 
290         if isTop:
291             self._createUpSticker(mark)
292             cY = 1
293         else:
294             self._createDownSticker(mark)
295             cY = -1
296 
297         self.itsFrame.pos = (cX, cY, -1)
298         self.initPos = (cX, cY, -1)
299         self.initAxis = vector(self.itsFrame.axis)
300 
301 
302 class EdgePart(RubikCubePart):
303     '''Create an edge part of a rubik's cube.'''
304 
305     def __init__(self, which, mark=False):
306         RubikCubePart.__init__(self)
307         self.partType = RubikCubePart.partTypes[which]
308         if which.startswith('F'):
309             self._createFrontEdge(which, mark)
310         elif which.startswith('B'):
311             self._createBackEdge(which, mark)
312         else:
313             self._createLeftRightEdge(which, mark)
314 
315 
316 
317 
318 
319 
320 
```

```
321     def _createFrontEdge(self, which, mark):
322         '''Create a front edge part of a rubik's cube.'''
323         self._createFrontSticker(mark)
324         if which=='FLE':
325             self._createLeftSticker(mark)
326             cX = -1; cY = 0
327         elif which=='FRE':
328             self._createRightSticker(mark)
329             cX = 1; cY = 0
330         elif which=='FUE':
331             self._createUpSticker(mark)
332             cX = 0; cY = 1
333         elif which=='FDE':
334             self._createDownSticker(mark)
335             cX = 0; cY = -1
336 
337         self.itsFrame.pos = (cX, cY, 1)
338         self.initPos = (cX, cY, 1)
339         self.initAxis = vector(self.itsFrame.axis)
340 
341     def _createBackEdge(self, which, mark):
342         self._createBackSticker(mark)
343         if which == 'BLE':
344             self._createLeftSticker(mark)
345             cX = -1; cY = 0
346         elif which == 'BRE':
347             self._createRightSticker(mark)
348             cX = 1; cY = 0
349         elif which == 'BUE':
350             self._createUpSticker(mark)
351             cX = 0; cY = 1
352         elif which == 'BDE':
353             self._createDownSticker(mark)
354             cX = 0; cY = -1
355         self.itsFrame.pos = (cX, cY, -1)
356         self.initPos = (cX, cY, -1)
357         self.initAxis = vector(self.itsFrame.axis)
358 
359     def _createLeftRightEdge(self, which, mark):
360         if which.startswith('L'):
361             self._createLeftSticker(mark)
362             cX = -1
363         elif which.startswith('R'):
364             self._createRightSticker(mark)
365             cX = 1
366         else:
367             print("Error in '_createLeftRightEdge'")
368 
369         if which.endswith('UE'):
370             self._createUpSticker(mark)
371             cY = 1
372         elif which.endswith('DE'):
373             self._createDownSticker(mark)
374             cY = -1
375         else:
376             print("Error in '_createLeftRightEdge'")
377 
378         self.itsFrame.pos = (cX, cY, 0)
379         self.initPos = (cX, cY, 0)
380         self.initAxis = vector(self.itsFrame.axis)
381 
382 
383 
384 
```

```
385 class CenterPart(RubikCubePart):
386     '''Create center part of a rubik's cube.'''
387 
388     def __init__(self, which, mark=False):
389         RubikCubePart.__init__(self)
390         self.partType = RubikCubePart.partTypes[which]
391         cX = cY = cZ = 0
392         if which == 'LCE':
393             self._createLeftSticker(mark)
394             cX = -1
395         elif which == 'RCE':
396             self._createRightSticker(mark)
397             cX = 1
398         elif which == 'UCE':
399             self._createUpSticker(mark)
400             cY = 1
401         elif which == 'DCE':
402             self._createDownSticker(mark)
403             cY = -1
404         elif which == 'FCE':
405              self._createFrontSticker(mark)
406              cZ = 1
407         elif which == 'BCE':
408             self._createBackSticker(mark)
409             cZ = -1
410         self.itsFrame.pos = (cX, cY, cZ)
411         self.initPos = (cX, cY, cZ)
412         self.initAxis = vector(self.itsFrame.axis)
413 
414 
415 class RubiksCube(RubikColorGlobals):
416     '''A class representing a Rubik's cube.'''
417 
418     _frontFacePerm ={'FLU': 'FRU', 'FRU': 'FRD', 'FRD': 'FLD', 'FLD': 'FLU',
419                 'FLE': 'FUE', 'FUE': 'FRE', 'FRE': 'FDE', 'FDE': 'FLE', 'FCE': 'FCE'}
420     _backFacePerm = {'BLU': 'BLD', 'BLD': 'BRD', 'BRD': 'BRU', 'BRU': 'BLU',
421                 'BLE': 'BDE', 'BDE': 'BRE', 'BRE': 'BUE', 'BUE': 'BLE', 'BCE': 'BCE'}
422 
423     _leftFacePerm = {'FLU': 'FLD', 'FLD': 'BLD', 'BLD': 'BLU', 'BLU': 'FLU',
424                 'FLE': 'LDE', 'LDE': 'BLE', 'BLE': 'LUE', 'LUE': 'FLE', 'LCE': 'LCE'}
425     _rightFacePerm ={'FRU': 'BRU', 'BRU': 'BRD', 'BRD': 'FRD', 'FRD': 'FRU',
426                 'FRE': 'RUE', 'RUE': 'BRE', 'BRE': 'RDE', 'RDE': 'FRE', 'RCE': 'RCE'}
427 
428     _upFacePerm = {'FLU': 'BLU', 'BLU': 'BRU', 'BRU': 'FRU', 'FRU': 'FLU',
429             'LUE': 'BUE', 'BUE': 'RUE', 'RUE': 'FUE', 'FUE': 'LUE', 'UCE': 'UCE'}
430     _downFacePerm = {'FLD': 'FRD', 'FRD': 'BRD', 'BRD': 'BLD', 'BLD': 'FLD',
431                 'LDE': 'FDE', 'FDE': 'RDE', 'RDE': 'BDE', 'BDE': 'LDE', 'DCE': 'DCE'}
432 
433     def __init__(self):
434 
435         self.cFrontLeftUp    = CornerPart('FLU')
436         self.cFrontLeftDown  = CornerPart('FLD')
437         self.cFrontRightUp   = CornerPart('FRU')
438         self.cFrontRightDown = CornerPart('FRD')
439 
440         fColor = self._faceColors['back']
441         self.cBackLeftUp     = CornerPart('BLU')
442         self.cBackLeftDown   = CornerPart('BLD')
443         self.cBackRightUp    = CornerPart('BRU')
444         self.cBackRightDown  = CornerPart('BRD')
445 
446         fColor = self._faceColors['front']
447         self.cFrontLeftEdge  = EdgePart('FLE')
448         self.cFrontRightEdge = EdgePart('FRE')
```

```
449         self.cFrontUpEdge    = EdgePart('FUE')
450         self.cFrontDownEdge  = EdgePart('FDE')
451 
452         self.cBackLeftEdge  = EdgePart('BLE')
453         self.cBackRightEdge = EdgePart('BRE')
454         self.cBackUpEdge    = EdgePart('BUE')
455         self.cBackDownEdge  = EdgePart('BDE')
456 
457         self.cLeftUpEdge    = EdgePart('LUE')
458         self.cLeftDownEdge  = EdgePart('LDE')
459         self.cRightUpEdge   = EdgePart('RUE')
460         self.cRightDownEdge = EdgePart('RDE')
461 
462         self.cFrontCenter   = CenterPart('FCE')
463         self.cBackCenter    = CenterPart('BCE')
464         self.cLeftCenter    = CenterPart('LCE')
465         self.cRightCenter   = CenterPart('RCE')
466         self.cUpCenter      = CenterPart('UCE')
467         self.cDownCenter    = CenterPart('DCE')
468 
469         self.frontFace = [self.cFrontLeftUp,    self.cFrontRightUp,
470                           self.cFrontRightDown, self.cFrontLeftDown,
471                           self.cFrontLeftEdge,  self.cFrontUpEdge,
472                           self.cFrontRightEdge, self.cFrontDownEdge,
473                           self.cFrontCenter]
474 
475         self.backFace = [self.cBackLeftUp,    self.cBackLeftDown,
476                          self.cBackRightDown, self.cBackRightUp,
477                          self.cBackLeftEdge,  self.cBackDownEdge,
478                          self.cBackRightEdge, self.cBackUpEdge,
479                          self.cBackCenter]
480 
481         self.upFace   = [self.cFrontLeftUp, self.cBackLeftUp,
482                          self.cBackRightUp, self.cFrontRightUp,
483                          self.cLeftUpEdge,  self.cBackUpEdge,
484                          self.cRightUpEdge, self.cFrontUpEdge,
485                          self.cUpCenter]
486 
487         self.downFace = [self.cFrontLeftDown, self.cFrontRightDown,
488                          self.cBackRightDown, self.cBackLeftDown,
489                          self.cLeftDownEdge,  self.cFrontDownEdge,
490                          self.cRightDownEdge, self.cBackDownEdge,
491                          self.cDownCenter]
492 
493         self.leftFace = [self.cFrontLeftUp,   self.cFrontLeftDown,
494                          self.cBackLeftDown,  self.cBackLeftUp,
495                          self.cFrontLeftEdge, self.cLeftDownEdge,
496                          self.cBackLeftEdge,  self.cLeftUpEdge,
497                          self.cLeftCenter]
498 
499         self.rightFace = [self.cFrontRightUp,   self.cBackRightUp,
500                           self.cBackRightDown,  self.cFrontRightDown,
501                           self.cFrontRightEdge, self.cRightUpEdge,
502                           self.cBackRightEdge,  self.cRightDownEdge,
503                           self.cRightCenter]
504 
505         self._rCubeD = {
506            'FLU': self.cFrontLeftUp,      # 1
507            'FLD': self.cFrontLeftDown,    # 2
508            'FRU': self.cFrontRightUp,     # 3
509            'FRD': self.cFrontRightDown,   # 4
510            'FLE': self.cFrontLeftEdge,    # 5
511            'FRE': self.cFrontRightEdge,   # 6
512            'FUE': self.cFrontUpEdge,      # 7
```

```
513             'FDE': self.cFrontDownEdge,     # 8
514             'FCE': self.cFrontCenter,       # 9
515 
516             'BLU': self.cBackLeftUp,        #10
517             'BLD': self.cBackLeftDown,      #11
518             'BRU': self.cBackRightUp,       #12
519             'BRD': self.cBackRightDown,     #13
520             'BLE': self.cBackLeftEdge,      #14
521             'BRE': self.cBackRightEdge,     #15
522             'BUE': self.cBackUpEdge,        #16
523             'BDE': self.cBackDownEdge,      #17
524             'BCE': self.cBackCenter,        #18
525 
526             'LUE': self.cLeftUpEdge,        #19
527             'LDE': self.cLeftDownEdge,      #20
528             'LCE': self.cLeftCenter,        #21
529 
530             'RUE': self.cRightUpEdge,       #22
531             'RDE': self.cRightDownEdge,     #23
532             'RCE': self.cRightCenter,       #24
533 
534             'UCE': self.cUpCenter,          #25
535             'DCE': self.cDownCenter         #26
536 
537         }
538 
539         self._undoList = []
540         self._redo = None
541         self._currentFacePerm = None
542         self._animationKeys = []
543         self._tempParts = []
544         self._currentAngle = 0
545         self._endingAngle = 0
546         self._permAxis = None
547         self._shouldAnimate = False
548         self._dt = 0
549         self._permPower = 0
550 
551 
552     def _resetCubeD(self):
553         self._rCubeD['FLU'] = self.cFrontLeftUp       # 1
554         self._rCubeD['FLD'] = self.cFrontLeftDown     # 2
555         self._rCubeD['FRU'] = self.cFrontRightUp      # 3
556         self._rCubeD['FRD'] = self.cFrontRightDown    # 4
557         self._rCubeD['FLE'] = self.cFrontLeftEdge     # 5
558         self._rCubeD['FRE'] = self.cFrontRightEdge    # 6
559         self._rCubeD['FUE'] = self.cFrontUpEdge       # 7
560         self._rCubeD['FDE'] = self.cFrontDownEdge     # 8
561         self._rCubeD['FCE'] = self.cFrontCenter       # 9
562 
563         self._rCubeD['BLU'] = self.cBackLeftUp        #10
564         self._rCubeD['BLD'] = self.cBackLeftDown      #11
565         self._rCubeD['BRU'] = self.cBackRightUp       #12
566         self._rCubeD['BRD'] = self.cBackRightDown     #13
567         self._rCubeD['BLE'] = self.cBackLeftEdge      #14
568         self._rCubeD['BRE'] = self.cBackRightEdge     #15
569         self._rCubeD['BUE'] = self.cBackUpEdge        #16
570         self._rCubeD['BDE'] = self.cBackDownEdge      #17
571         self._rCubeD['BCE'] = self.cBackCenter        #18
572 
573         self._rCubeD['LUE'] = self.cLeftUpEdge        #19
574         self._rCubeD['LDE'] = self.cLeftDownEdge      #20
575         self._rCubeD['LCE'] = self.cLeftCenter        #21
576 
```

```
        self._rCubeD['RUE'] = self.cRightUpEdge      #22
        self._rCubeD['RDE'] = self.cRightDownEdge    #23
        self._rCubeD['RCE'] = self.cRightCenter      #24

        self._rCubeD['UCE'] = self.cUpCenter        #25
        self._rCubeD['DCE'] = self.cDownCenter      #26

        rCubeD = self._rCubeD
        keys = rCubeD.keys()
        for k in keys:
            f = rCubeD[k].itsFrame
            f.pos  = rCubeD[k].initPos
            f.axis = (1, 0, 0)
            f.up   = (0, 1, 0)


##    def _showHidePartMarkers(show, parts):
##        for p in parts: p.showHideMarkers(show)

    def showHideUpFaceMarkers(self, showIt, which):
        for x in self.upFace:
            if x.partType.endswith(which):
                x.showHideMarkers(showIt)

    def showHideUpFaceMarker(self, showIt, which):
        if which == 'FUE':
            self.cFrontUpEdge.showHideMarkers(showIt)
        elif which == 'BUE':
            self.cBackUpEdge.showHideMarkers(showIt)
        elif which == 'LUE':
            self.cLeftUpEdge.showHideMarkers(showIt)
        elif which == 'RUE':
            self.cLeftUpEdge.showHideMarkers(showIt)
        elif which == 'FLU':
            self.cFrontLeftUp.showHideMarkers(showIt)
        elif which == 'FRU':
            self.cFrontRightUp.showHideMarkers(showIt)
        elif which == 'BLU':
            self.cBackLeftUp.showHideMarkers(showIt)
        elif which == 'BRU':
            self.cBackRightUp.showHideMarkers(showIt)

    def applyPerm(self, perm, power):
        if type(power) == int and power in (-1, 1, 2):
            if perm =='F':
                self._rotateFrontFace(power)
            elif perm == 'B':
                 self._rotateBackFace(power)
            elif perm == 'U':
                 self._rotateUpFace(power)
            elif perm == 'D':
                 self._rotateDownFace(power)
            elif perm == 'L':
                 self._rotateLeftFace(power)
            elif perm == 'R':
                 self._rotateRightFace( power)
```

```
641     def doPerm(self, facePerm, pAxis, power):
642
643         self._currentFacePerm = facePerm
644         self._animationKeys = facePerm.keys()
645         self._permAxis = pAxis
646         self._permPower = power
647         self._tempParts = {}
648         for k in self._animationKeys:
649             self._tempParts[k] = self._rCubeD[k]
650         self._currentAngle = 0
651         self._shouldAnimate = True
652
653         if power == -1:
654             self._endingAngle = -pi/2
655             self._dt = -pi/8
656         else:
657             self._endingAngle = power*pi/2
658             self._dt = pi/8
659
660
661     def doAnimation(self):
662
663         if self._shouldAnimate:
664             aKeys = self._animationKeys
665             for k in aKeys:
666                 self._tempParts[k].itsFrame.rotate(angle = self._dt,
667                             axis = self._permAxis, origin = (0,0,0))
668             self._currentAngle += self._dt
669             if (abs(self._currentAngle - self._endingAngle) < 0.01):
670                 self._currentAngle = self._endingAngle
671                 self._shouldAnimate = False
672                 self.finishPermAnimation()
673
674     def finishPermAnimation(self):
675         perm = self._currentFacePerm
676         tempParts = self._tempParts
677         if self._permPower == -1:
678             for k in self._animationKeys:
679                 self._rCubeD[k] = tempParts[perm[k]]
680         elif self._permPower == 1:
681             for k in self._animationKeys:
682                 self._rCubeD[perm[k]] = tempParts[k]
683         else:
684             for k in self._animationKeys:
685                 self._rCubeD[perm[perm[k]]] = tempParts[k]
686
687
688     def _postUndo(self, which, power):
689         self._redo = None
690         if power == -1:
691             self._undoList.append((which,1))
692         elif power == 1:
693             self._undoList.append((which,-1))
694         elif power == 2:
695             self._undoList.append((which,2))
696
697     def _doFacePerm(self, which, power):
698         if which == 'front':
699             self.doPerm(self._frontFacePerm, (0, 0, -1), power)
700         elif which == 'back':
701             self.doPerm(self._backFacePerm, (0, 0, 1),  power)
702         elif which == 'left':
703             self.doPerm(self._leftFacePerm, (1, 0, 0), power)
704         elif which == 'right':
```

```
705             self.doPerm(self._rightFacePerm, (-1, 0, 0), power)
706         elif which == 'up':
707             self.doPerm(self._upFacePerm, (0, -1, 0), power)
708         elif which == 'down':
709             self.doPerm(self._downFacePerm, (0, 1, 0), power)
710 
711     def _doUndo(self):
712         if len(self._undoList)>0:
713             which, power = self._undoList.pop()
714             self._doFacePerm(which, power)
715             if power == -1:
716                 self._redo = (which, 1)
717             elif power == 1:
718                 self._doRedo = (which, -1)
719             elif power == 2:
720                 self._redo = (which, 2)
721 
722     def _doRedo(self):
723         if self._redo != None:
724             which, power = self._redo
725             self._doFacePerm(which, power)
726             self._postUndo(which, power)
727 
728     def rotateFrontFace(self, power):
729         self._doFacePerm('front',power)
730         self._postUndo('front', power)
731 
732     def rotateBackFace(self, power):
733         self._doFacePerm('back',power)
734         self._postUndo('back', power)
735 
736     def rotateUpFace(self, power):
737         self._doFacePerm('up',power)
738         self._postUndo('up', power)
739 
740     def rotateDownFace(self, power):
741         self._doFacePerm('down',power)
742         self._postUndo('down', power)
743 
744     def rotateLeftFace(self, power):
745         self._doFacePerm('left',power)
746         self._postUndo('left', power)
747 
748     def rotateRightFace(self, power):
749         self._doFacePerm('right',power)
750         self._postUndo('right', power)
751 
752     def getFaceColor(self, which):
753         if which in self._faceColors.keys():
754             return self._faceColors[which]
755         else:
756             return color.black
757 
758     def getFaceColorForNewButtons(self, which):
759         if which in self._faceColors.keys():
760             c = self._faceColors[which]
761             return self._faceColorNames[c]
762         else:
763             return 'Black'
764 
765 
766 
767 
768 
```

```
769     def resetFaceColors(self):
770         self._faceColors['left']  = color.blue
771         self._faceColors['front'] = color.white
772         self._faceColors['right'] = color.green
773         self._faceColors['back']  = color.yellow
774         self._faceColors['up']    = color.red
775         self._faceColors['down']  = _orange
776 
777     def rotateFaceColorsAroundZAxis(self):
778         self._resetCubeD()
779         zPerm = {'left':'front', 'front':'right', 'right': 'back', 'back':'left',
780                  'up': 'up', 'down':'down'}
781         fColors = self._faceColors
782         colors = fColors.copy()
783         for k in zPerm.keys(): colors[zPerm[k]] = fColors[k]
784         for k in fColors.keys(): fColors[k] = colors[k]
785 
786         rCube = self._rCubeD
787         for k in rCube.keys():
788             rCube[k].setColors(fColors)
789 
790     def rotateFaceColorsAroundXAxis(self):
791         self._resetCubeD()
792         xPerm = {'front':'down',  'down':'back',  'back': 'up',
793                  'up':'front', 'left':'left', 'right':'right'}
794         fColors = self._faceColors
795         colors = fColors.copy()
796         for k in xPerm.keys(): colors[xPerm[k]] = fColors[k]
797         for k in fColors.keys(): fColors[k] = colors[k]
798 
799         rCube = self._rCubeD
800         for k in rCube.keys():
801             rCube[k].setColors(fColors)
802 
803 
804 class Frame(wx.Frame):
805     def __init__(self, parent, title, width, height):
806         wx.Frame.__init__(self, parent, -1, title, pos = (801, 20),
807                           size = (width,height))
808 
809 
810 class RubiksCubeScene(RubikColorGlobals):
811     '''Class to setup and control a Visual Python scene containting
812     a Rubik's cube.'''
813 
814     _bSize = 30
815     _minX = -80; _maxY = 80
816     _deltaX = 32; _deltaY = 32
817     _cWindowX = 700; _cWindowY = 0
818     _cWindowWidth = 600; _cWindowHeignt = 600
819     _mainWindow = None
820 
821 
822     def __init__(self):
823         self._viewLabel = None
824         sWidth = 800
825         sHeight = 800
826         _mainWindow = window(menus=False, title="Rubik's Cube",
827                              x=0, y=0, width=sWidth, height=sWidth)
828         scene1 = display(window = _mainWindow, title="",
829                          x=0, y=0, width=sWidth, height=sHeight,
830                          background=(0.7,0.7,0.7))
831 
832         scene1.range = 4
```

```
833         self._scene = scene1
834         scene1.forward = (cos(pi/4), -cos(pi/4), -cos(pi/4))
835         distant_light(direction=(-1, -1, -1), color=color.white)
836         distant_light(direction=( 1, -1, -1), color=color.white)
837         distant_light(direction=(-1,  1, -1), color=color.white)
838         distant_light(direction=(-1, -1,  1), color=color.white)
839         distant_light(direction=( 1,  1,  1), color=color.white)
840 
841         self._rubiksCube = RubiksCube()
842         self._viewLabel = label(pos=(0,3,0), height=36, text='FLU-View',
843                                 color=color.white, linecolor = color.black)
844         self._createNewControlWindow(_mainWindow.win)
845 
846 
847     def _showView(self, which):
848         vlPos = (0,3,0)
849         if which == 'flu':
850             self._scene.forward = (1, -1, -1)
851         elif which == 'fru':
852             self._scene .forward = (-1, -1, -1)
853         elif which == 'blu':
854             self._scene.forward = (1, -1, 1)
855         elif which == 'bru':
856             self._scene.forward = (-1, -1, 1)
857         elif which == 'fld':
858             self._scene.forward = (1, 1, -1)
859             vlPos = (0, 6, 0)
860         elif which == 'frd':
861             self._scene.forward = (-1, 1, -1)
862             vlPos = (0, 6, 0)
863         elif which == 'bld':
864             self._scene.forward = (1, 1, 1)
865             vlPos = (0, 6, 0)
866         elif which == 'brd':
867             self._scene.forward = (-1, 1, 1)
868             vlPos = (0, 6, 0)
869         self._viewLabel.text = which.upper()+'-View'
870         self._viewLabel.pos = vlPos
871 
872 
873     def _setNewButtonsColor(self, faceButtons, which):
874         c = self._rubiksCube.getFaceColorForNewButtons(which)
875         if c == RubiksCube._orange: c = RubiksCube._orangeStr;
876 
877         for b in faceButtons.values():
878             b.SetBackgroundColour(c)
879             if c =='Blue':
880                 b.SetForegroundColour("White")
881             else:
882                 b.SetForegroundColour("Black")
883             b.Refresh()
884         if which == "front":
885             self._viewLabel.color = self._rubiksCube.getFaceColor(which)
886 
887 
888     def doRotateColorsAroundZAxis(self):
889         self._rubiksCube.rotateFaceColorsAroundZAxis()
890         self._setNewButtonsColor(self.frontFaceButtons, 'front')
891         self._setNewButtonsColor(self.rightFaceButtons, 'right')
892         self._setNewButtonsColor( self.backFaceButtons, 'back' )
893         self._setNewButtonsColor( self.leftFaceButtons, 'left' )
894 
895 
896 
```

```
897     def doRotateColorsAroundXAxis(self):
898         self._rubiksCube.rotateFaceColorsAroundXAxis()
899         self._setNewButtonsColor(self.frontFaceButtons, 'front')
900         self._setNewButtonsColor( self.downFaceButtons, 'down' )
901         self._setNewButtonsColor( self.backFaceButtons, 'back' )
902         self._setNewButtonsColor(   self.upFaceButtons, 'up' )
903 
904 
905     def _createNewButton(self, panel, name, color):
906         b = buttons.GenButton(panel, -1, name)
907         b.SetFont(wx.Font(16, wx.SWISS, wx.NORMAL, wx.BOLD, False))
908         b.SetBezelWidth(5)
909         b.SetMinSize(wx.DefaultSize)
910         b.SetBackgroundColour(color)
911         b.SetForegroundColour(wx.BLACK)
912         return b
913 
914     def _createGridOfButtons(self, panel, buttonNames, buttonColors, rows, cols):
915         gridS = wx.GridSizer(rows, cols, 5, 5)
916         buttons = {}
917         for name, color in zip(buttonNames, buttonColors):
918                 if name == '':
919                     b = wx.StaticText(panel, label='')
920                 else:
921                     b = self._createNewButton(panel, name, color)
922                     if color=='Blue':
923                         b.SetForegroundColour(wx.WHITE)
924 
925                 buttons[name] = b
926                 gridS.Add(b, proportion=1, flag=wx.ALL|wx.EXPAND, border=5)
927         return (buttons, gridS)
928 
929     def _createSliderBox(self, panel, prompt, sMin, sMax, sValue):
930         slider = wx.Slider(panel, size=(200,30), minValue=sMin, maxValue=sMax)
931         slider.SetValue(sValue)
932         label = wx.StaticText(panel, label=prompt)
933         label.SetForegroundColour('white')
934         label.SetFont(wx.Font(20, wx.SWISS, wx.NORMAL, wx.BOLD, False))
935         hbox = wx.BoxSizer(wx.HORIZONTAL)
936         hbox.Add(label, flag=wx.ALL, border = 10)
937         hbox.Add(slider, flag=wx.ALL, border = 10)
938         return (slider, hbox)
939 
940     def _bindButton(self, buttons, name, callback):
941         if name == 'animationRate':
942             self.animationSlider.Bind(wx.EVT_SCROLL, callback)
943         else:
944             buttons[name].Bind(wx.EVT_BUTTON,callback)
945 
946     def _createNewControlWindow(self, win):
947         rCube = self._rubiksCube
948         ctlFrame = Frame(win, "Transform Controls", 625,475)
949         panel = wx.Panel(ctlFrame)
950         ##panel.SetBackgroundColour("black")
951         bNames = ['F', 'F^2', 'F^(-1)', 'B', 'B^2', 'B^(-1)', 'L', 'L^2', 'L^(-1)',
952                   'R', 'R^2', 'R^(-1)', 'U', 'U^2', 'U^(-1)', 'D', 'D^2', 'D^(-1)']
953 
954         orangeStr = RubiksCube._orangeStr
955         bColors = ['White', 'White', 'White', 'Yellow', 'Yellow', 'Yellow',
956                    'Blue', 'Blue', 'Blue', 'Green', 'Green', 'Green',
957                    'Red', 'Red', 'Red', orangeStr, orangeStr, orangeStr]
958         rotateButtons, rotateGrid = self._createGridOfButtons(panel, bNames,
959                                                                bColors, 6, 3)
960         self.rotateButtons = rotateButtons
```

```
961
962         rotateButtons['F'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateFrontFace(1))
963         rotateButtons['F^2'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateFrontFace(2))
964         rotateButtons['F^(-1)'].Bind(wx.EVT_BUTTON,
965                                      lambda event: rCube.rotateFrontFace(-1))
966         self.frontFaceButtons = {'F': rotateButtons['F'], 'F2': rotateButtons['F^2'],
967                                  'F3': rotateButtons['F^(-1)']}
968
969         rotateButtons['B'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateBackFace(1))
970         rotateButtons['B^2'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateBackFace(2))
971         rotateButtons['B^(-1)'].Bind(wx.EVT_BUTTON,
972                                      lambda event: rCube.rotateBackFace(-1))
973         self.backFaceButtons = {'B': rotateButtons['B'], 'B2': rotateButtons['B^2'],
974                                 'B3': rotateButtons['B^(-1)']}
975
976         rotateButtons['L'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateLeftFace(1))
977         rotateButtons['L^2'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateLeftFace(2))
978         rotateButtons['L^(-1)'].Bind(wx.EVT_BUTTON,
979                                      lambda event: rCube.rotateLeftFace(-1))
980         self.leftFaceButtons = {'L': rotateButtons['L'], 'L2': rotateButtons['L^2'],
981                                 'L3': rotateButtons['L^(-1)']}
982
983         rotateButtons['R'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateRightFace(1))
984         rotateButtons['R^2'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateRightFace(2))
985         rotateButtons['R^(-1)'].Bind(wx.EVT_BUTTON,
986                                     lambda event: rCube.rotateRightFace(-1))
987         self.rightFaceButtons = {'R': rotateButtons['R'], 'R2': rotateButtons['R^2'],
988                                  'R3': rotateButtons['R^(-1)']}
989
990         rotateButtons['U'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateUpFace(1))
991         rotateButtons['U^2'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateUpFace(2))
992         rotateButtons['U^(-1)'].Bind(wx.EVT_BUTTON,
993                                      lambda event: rCube.rotateUpFace(-1))
994         self.upFaceButtons = {'U': rotateButtons['U'], 'U2': rotateButtons['U^2'],
995                               'U3': rotateButtons['U^(-1)']}
996
997         rotateButtons['D'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateDownFace(1))
998         rotateButtons['D^2'].Bind(wx.EVT_BUTTON, lambda event: rCube.rotateDownFace(2))
999         rotateButtons['D^(-1)'].Bind(wx.EVT_BUTTON,
1000                                      lambda event: rCube.rotateDownFace(-1))
1001        self.downFaceButtons = {'D': rotateButtons['D'], 'D2': rotateButtons['D^2'],
1002                                'D3': rotateButtons['D^(-1)']}
1003
1004        bNames = ['FLU', 'FRU', 'BLU', 'BRU',
1005                  'FLD', 'FRD', 'BLD', 'BRD']
1006        bColors = ['Pink', 'Pink', 'Pink', 'Pink', 'Pink', 'Pink','Pink', 'Pink']
1007        viewButtons, viewGrid = self._createGridOfButtons(panel, bNames,
1008                                                          bColors, 4, 2)
1009        self.viewButtons = viewButtons
1010
1011        viewButtons['FLU'].Bind(wx.EVT_BUTTON, lambda event: self._showView('flu'))
1012        viewButtons['FRU'].Bind(wx.EVT_BUTTON, lambda event: self._showView('fru'))
1013        viewButtons['BLU'].Bind(wx.EVT_BUTTON, lambda event: self._showView('blu'))
1014        viewButtons['BRU'].Bind(wx.EVT_BUTTON, lambda event: self._showView('bru'))
1015        viewButtons['FLD'].Bind(wx.EVT_BUTTON, lambda event: self._showView('fld'))
1016        viewButtons['FRD'].Bind(wx.EVT_BUTTON, lambda event: self._showView('frd'))
1017        viewButtons['BLD'].Bind(wx.EVT_BUTTON, lambda event: self._showView('bld'))
1018        viewButtons['BRD'].Bind(wx.EVT_BUTTON, lambda event: self._showView('brd'))
1019
1020        bNames = ['Undo', 'Redo', 'Reset']
1021        bColors = ['Magenta', 'Magenta', 'Cyan']
1022        editButtons, editGrid = self._createGridOfButtons(panel, bNames,
1023                                                          bColors, 1, 3)
1024        self.editButtons = editButtons
```

```
1025
1026        editButtons['Undo'].Bind(wx.EVT_BUTTON,  lambda event: rCube._doUndo())
1027        editButtons['Redo'].Bind(wx.EVT_BUTTON,  lambda event: rCube._doRedo())
1028        editButtons['Reset'].Bind(wx.EVT_BUTTON, lambda event: rCube._resetCubeD())
1029
1030        bNames = ['FRBL', 'FDBU']
1031        bColors = ['Cyan', 'Cyan']
1032        colorButtons, colorGrid = self._createGridOfButtons(panel, bNames,
1033                                                            bColors, 1, 2)
1034        self.colorButtons = colorButtons
1035        colorButtons['FRBL'].Bind(wx.EVT_BUTTON,
1036                    lambda event: self.doRotateColorsAroundZAxis())
1037        colorButtons['FDBU'].Bind(wx.EVT_BUTTON,
1038                     lambda event: self.doRotateColorsAroundXAxis())
1039
1040        box = wx.StaticBox(panel,-1,"Edit")
1041        editbox = wx.StaticBoxSizer(box, wx.HORIZONTAL)
1042        editbox.Add(editGrid)
1043
1044        box = wx.StaticBox(panel,-1,"Transforms")
1045        rotationbox = wx.StaticBoxSizer(box, wx.HORIZONTAL)
1046        rotationbox.Add(rotateGrid)
1047
1048        editRotationbox = wx.BoxSizer(wx.VERTICAL)
1049        editRotationbox.Add(editbox, flag=wx.EXPAND|wx.ALL, border=5)
1050        editRotationbox.Add(rotationbox, flag=wx.EXPAND|wx.ALL, border=5)
1051
1052        box = wx.StaticBox(panel,-1,"Change View")
1053        viewbox = wx.StaticBoxSizer(box, wx.HORIZONTAL)
1054        viewbox.Add(viewGrid)
1055
1056        box = wx.StaticBox(panel,-1,"Cycle Cube Colors")
1057        colorbox = wx.StaticBoxSizer(box, wx.HORIZONTAL)
1058        colorbox.Add(colorGrid)
1059
1060        viewColorbox = wx.BoxSizer(wx.VERTICAL)
1061        viewColorbox.Add(viewbox, flag=wx.EXPAND|wx.ALL, border=5)
1062        viewColorbox.Add(colorbox, flag=wx.EXPAND|wx.ALL, border=5)
1063
1064        hbox = wx.BoxSizer(wx.HORIZONTAL)
1065        hbox.Add(editRotationbox, flag=wx.EXPAND|wx.ALL, border=5)
1066        hbox.Add(viewColorbox, flag=wx.EXPAND|wx.ALL, border=5)
1067
1068        vbox = wx.BoxSizer(wx.VERTICAL)
1069        vbox.Add(hbox, flag=wx.EXPAND|wx.ALL, border=5)
1070
1071        panel.SetSizer(vbox)
1072
1073        ctlFrame.Show()
1074        return ctlFrame
1075
1076
1077    def _handleKey(self):
1078        if scene.kb.keys:
1079            s = scene.kb.getkey()
1080            power = 1
1081            if s.isupper():
1082                power = -1
1083                s = s.lower()
1084
1085            if s == 'f':
1086                rotateFrontFace(power)
1087            elif s == 'b':
1088                rotateBackFace(power)
```

```
1089            elif s == 'l':
1090                rotateLeftFace(power)
1091            elif s == 'r':
1092                rotateRightFace(power)
1093            elif s == 'u':
1094                rotateUpFace(power)
1095            else:
1096                rotateDownFace(power)
1097
1098    def doEventLoop(self):
1099        ##self._rubiksCube.showHideUpFaceMarkers(True, 'Edge')
1100        self._rubiksCube.showHideUpFaceMarker(True, 'FUE')
1101        while True:
1102            rate(10)
1103            self._rubiksCube.doAnimation()
1104            self._handleKey()
1105
1106
1107 rbs = RubiksCubeScene()
1108 rbs.doEventLoop()
1109
1110
1111
```