1 Administrative Topics

- Course Evaluations. important. separate for lecture and lab. we are both invested in the projects. Bruce writes them. We take turns grading.
- For Wed, have some of your simple objects made so you can get our help making more complex objects

2 Drawing trees using recursion

Let’s draw a binary tree (using code developed for Project 11). We will require that the binary tree be drawn in exactly one orientation – with the top cube at the top, and the left cubes to the left, and the right cubes to the right. This will make it much more straight-forward to compute the locations of the subtrees.

Let’s use recursion. To do this, we will add a drawBinaryTree method to the Cube class. It will draw the current cube and then make two recursive calls to draw the left and right subtrees. The base case, which is more of a stopping case, will simply return from the function before anything is drawn. We will use a parameter levels to indicate how many levels the tree should have. The base case, is that levels is zero.
To compute the location of the subtrees, we need to use distance and our knowledge of what (x,y,z) means for a cube.

Here is the code for the method:

```python
# draw a binary tree with head node at (xpos,ypos,zpos)
# The tree should have levels levels.
def drawBinaryTree( self, xpos, ypos, zpos, levels ):
    if levels == 0:
        return

    # draw head node for this subtree
    self.draw( xpos, ypos, zpos=zpos )

    # draw left tree
    xd = self.distance
    yd = self.distance*2.0
    self.drawBinaryTree( xpos-xd, ypos-yd, zpos, levels-1 )

    # draw right tree
    self.drawBinaryTree( xpos+xd, ypos-yd, zpos, levels-1 )
```

Lines 4 and 5 implement the base/stopping case.

Lines 7 and 8 draw the top cube (head node) for the tree.

Lines 11 through 13 draw the left subtree, which we draw to the left of and below the head node. Likewise, line 16 draws the right subtree to the right of and below the head node.

Lines 17 through 20 repeat the process for the left subtree.
2.1 Adding Color

We decided it would be fun to give each level a color. So, let’s use a list of colors, and the level will help us index into the list.

```python
# draw a binary tree with the head node at (xpos, ypos, zpos)
# with levels levels
def drawBinaryTree( self, xpos, ypos, zpos, levels, 
    colors=None):
    if levels == 0:
        return
    if colors != None:
        self.setColor( colors[(levels-1)%len(colors)] )
    # draw head node
    self.draw( xpos, ypos, zpos=zpos, roll=45, scale=scale )
    # draw left subtree
    xd = self.distance
    yd = self.distance * 2
    self.drawBinaryTree( xpos-xd, ypos-yd, zpos, levels-1, 
        colors )
    # draw right subtree
    self.drawBinaryTree( xpos+xd, ypos-yd, zpos, levels-1, 
        colors )
```

If no list of colors is supplied, then no modification is made to the color. If it is, then we set the color according to levels. The indexing is the most complicated part here. First, we say that level 1 requires color 0 (i.e. the bottom layer requires the first color), so we use levels – 1. Then, we want to cycle though the colors if there aren’t enough colors for every level, so we use the mod (remainder) operator.
2.2 Scaling

To make cubes at lower levels smaller than cubes at higher levels, we can add a scaling factor. First, we use the scale to determine the size of the head node, then we shrink it to 80% for the next level.

```python
# draw a binary tree with the head node at (xpos, ypos, zpos)
# with levels levels
def drawBinaryTree(self, xpos, ypos, zpos, levels,
                   colors=None, scale=1.0):
    if levels == 0:
        return

    if colors != None:
        self.setColor( colors[(levels−1)%len(colors)] )

    # draw head node
    self.draw( xpos, ypos, zpos=zpos, roll=45, scale=scale )

    # draw left subtree
    xd = self.distance
    yd = self.distance * 2
    self.drawBinaryTree( xpos−xd, ypos−yd, zpos, levels−1,
                         colors, 0.8*scale )

    # draw right subtree
    self.drawBinaryTree( xpos+xd, ypos−yd, zpos, levels−1,
                         colors, 0.8*scale )
```

Note that in order to truly scale it, we should scale the distances xd and yd. We did that, but it looked ugly, so decided not to do it.
2.3 Quad Tree

We can give more depth to the tree by adding branches in the front and behind the current tree. Let's call that a quad tree. Here is the code for it:

```python
# draw a quad tree with the head node at (xpos, ypos, zpos)
# with levels levels
def drawQuadTree( self, xpos, ypos, zpos, levels ):
    if levels == 0:
        return

    # draw head node
    self.draw( xpos, ypos, zpos=zpos, roll=45 )

    # draw left subtree
    xd = self.distance
    yd = self.distance * 2
    zd = self.distance
    self.drawQuadTree( xpos-xd, ypos-yd, zpos, levels-1 )

    # draw right subtree
    self.drawQuadTree( xpos+xd, ypos-yd, zpos, levels-1 )

    # draw back tree
    self.drawQuadTree( xpos, ypos-yd, zpos-zd, levels-1 )

    # draw front tree
    self.drawQuadTree( xpos, ypos-yd, zpos+zd, levels-1 )
```
3 Proj 11 note

Here is another syntax for creating strings that makes the code a bit easier to read. It uses the % to perform string formatting. Let me start by explaining an example

\[
s = \left(\%(f)F(90) + \%(f)F(90) + \right) \% (\text{width}, \text{height})
\]

This creates a string that draws two sides of a rectangle. We can think of this as being divided into three parts: a format string, %, and the tuple containing the values. The format string tells us what the final string should look like, but uses place-holders for some of the parts. In particular, each ’%f’ string is a place-holder indicating that a number will appear there. Notice that there are two place-holders in the format string and two values in the final tuple (with and height). The \text{str(width)} replaces the first place-holder and \text{str(height)} replaces the second place-holder. So, if width is 40.0 and height is 45.0, then the value of \(s\) will be \(40.000000F(90)+(45.000000)F(90)+\). You may use this notation within parentheses if you want to concatenate its result with other strings. For example, the code to create a filled rectangle string is this:

\[
bw = '\{ ' + 2*(\%(f)F(90) + \%(f)F(90) + '%(width,height)'))+ ' \}'
\]