Divide, Conquer, and Glue with Pictures

Recall big idea #3: Divide, conquer & glue (DCG)

Divide
problem P into subproblems.

Conquer
each of the subproblems, &

Glue (combine)
the solutions to the subproblems into a solution S for P.

Recall big idea number 1: Abstraction

DCG to make cool pictures

Today we’ll see how to use DCG to make complex and interesting pictures in a simple way.

But first it will help to define a more abstract notion of “picture” starting with cs1graphics.
Peter Henderson’s Picture Language

Henderson’s Functional Geometry paper:
http://eprints.soton.ac.uk/257577/1/funcgeo2.pdf

Picture abstraction on top of cs1graphics

- Inspired by Peter Henderson’s picture language.
- We define a “picture” as a cs1graphics Drawable object that’s 200x200 centered at reference point (0,0).
- By fixing the size of a picture, we never have to worry about pictures of different sizes.
- Rotations and flips of pictures by angles that are multiples of 90 degrees results in another picture!
- We will define many functions that take picture(s) as inputs and return a picture as an output.

Some Primitive Pictures

bp (blue patch)

rp (red patch)

gw (green wedge)

Sample primitives defined in Python [1]

def patch(color):
    pic = Square(200)
    pic.setFillColor(color)
    pic.setBorderColor('black')
    return pic

rp = patch('red')

def wedge(color):
    pic = Polygon(Point(100, 0),
                   Point(100,100),
                   Point(-100, 100))
    #Shift reference point from (100,0) to(0,0)
    pic.adjustReference(-100,0)
    pic.setFillColor(color)
    pic.setBorderColor(color) # no border!
    return pic

gw = wedge('green')
Sample primitives defined in Python [2]

```python
def checkmark(downColor, upColor):
    pic = Layer()
    downstroke = Path(Point(-100,0), Point(0,100))
    downstroke.setBorderColor(downColor)
    pic.add(downstroke)
    upstroke = Path(Point(0,100), Point(100,-100))
    upstroke.setBorderColor(upColor)
    pic.add(upstroke)
    return pic

mark = checkmark('red', 'blue')
```

```python
def empty():
    return Layer()
empty = empty()
```

Displaying Pictures

```python
def displayPic(pic):
    '''Display picture in 600x600 canvas'''
    frame = Canvas(600, 600, 'white', 'Picture Frame')
    # Clone pic before changing it; otherwise
    # it wouldn’t be a picture anymore!
    framedPic = pic.clone()
    framedPic.scale(3)  # scale by 3 to fill 600x600 canvas
    framedPic.moveTo(300, 300)  # move to center of canvas
    frame.add(framedPic)
```

Displaying Pictures

* The definition of displayPic is a tad more complex than shown here to allow
  for a closeAllPics() function that closes all canvases created by displayPic.

Clockwise rotations of pictures

```python
clockwise90(p):  # Returns new picture that’s p rotated 90° clockwise
clockwise180(p):  # Returns new picture that’s p rotated 180° clockwise
clockwise270(p):  # Returns new picture that’s p rotated 270° clockwise
```

```python
clockwise90(gw)
clockwise180(gw)
clockwise270(gw)
```

Defining picture rotations in Python

```python
def clockwisePic(pic, angle):
    newPic = pic.clone()  # create new pic by cloning it.
    newPic.rotate(angle)  # if angle is a multiple of 90,
    # result still satisfies
    # definition of picture.
    return newPic
```

```python
def clockwise90(pic):
    return clockwisePic(pic, 90)
```

```python
def clockwise180(pic):
    return clockwisePic(pic, 180)
```

```python
def clockwise270(pic):
    return clockwisePic(pic, 270)
```
Flipping pictures

flipAcrossVert(p);  # Returns new picture that’s p flipped across vertical axis
flipAcrossHoriz(p);  # Returns new picture that’s p flipped across horizontal axis
flipAcrossDiag(p);  # Returns new picture that’s p flipped across 45-degree axis

Overlaying pictures

def overlay(pic1, pic2):
    '''Returns a new pic in which pic1 appears on top of pic2'''
    newPic = Layer()
    newPic.add(pic2)  # bottom pic goes first
    newPic.add(pic1)  # top pic goes last
    return newPic

fourPics: Combining four pictures

def fourPics(a, b, c, d):
    newPic = Layer()
    aHalf = a.clone()
    aHalf.scale(0.5)
    bHalf = b.clone()
    bHalf.scale(0.5)
    cHalf = c.clone()
    cHalf.scale(0.5)
    dHalf = d.clone()
    dHalf.scale(0.5)
    aHalf.move(-50,-50)
    bHalf.move(50,-50)
    cHalf.move(-50,50)
    dHalf.move(50,50)
    newPic.add(aHalf)
    newPic.add(bHalf)
    newPic.add(cHalf)
    newPic.add(dHalf)
    return newPic

Defining picture flipping in Python

def flipPic (pic, angle):
    newPic = pic.clone()  # create new pic by cloning it.
    newPic.flip(angle)  # if angle is a multiple of 45, 
    # result still satisfies 
    # definition of picture.
    return newPic

def flipAcrossVert(pic):
    return flipPic(pic, 0)

def flipAcrossHoriz(pic):
    return flipPic(pic, 90)

def flipAcrossDiag(pic):
    return flipPic(pic, 45)
fourSame: Combining four copies of one picture

```python
def fourSame(pic):
    return fourPics(pic, pic, pic, pic)
```

Repeated tiling

```python
def tiling(pic):
    return fourSame(fourSame(fourSame(fourSame(pic))))
```

How to make a checkerboard?

```python
def checkerboard(color1, color2):
    return fourSame(fourSame(fourPics(patch(color1), patch(color2),
                                   patch(color2), patch(color1))))
```

DCG on checkerboard

```python
def checkerboard(color1, color2):
    return fourSame(fourSame(fourPics(patch(color1), patch(color2),
                                   patch(color2), patch(color1))))
```
Combining four rotations of a picture

```
def rotations(pic):
    return rotations(clockwise270(pic))
def rotations2(pic):
    return rotations(clockwise90(pic))
```

A simple recipe for complexity

```
def wallpaper(pic):
    return rotations(rotations(rotations(rotations(pic))))
def design(pic):
    return rotations2(rotations2(rotations2(rotations2(pic))))
```

A quilt problem

How do we build this complex quilt …

… from simple primitive pictures like this?

```
triangles('green','blue') patch('red')
```

Divide the quilt into subproblems
Conquer the subproblems using “wishful thinking”

def quilt():
    return fourPics(clockwise270(quadrant()),
                    quadrant(),
                    clockwise180(quadrant()),
                    clockwise90(quadrant()))

def rotations(pic):
    # picture function from before
    return fourPics(clockwise270(pic), pic, 
                    clockwise180(pic), clockwise90(pic))

Glue the subsolutions to solve the problem

def quilt():
    return fourPics(clockwise270(quadrant()),
                    quadrant(),
                    clockwise180(quadrant()),
                    clockwise90(quadrant()))

def quadrant():
    return corner(star('yellow', 'red', 'blue'), 
                  star('red', 'green', 'blue'))

def corner(llPic, outerPic):
    return fourPics(outerPic, outerPic, 
                    llPic, outerPic)

Abstracting over the glue

def quilt():
    return rotations(quadrant())

Subproblem: quadrant()
Continue the descent ...

```python
def star(innerColor, middleColor, outerColor):
    return rotations(starQuadrant(innerColor, middleColor, outerColor))
```

```
DCG with Pics  29
```

... until we reach primitives

```
def starQuadrant(squareColor, llTriColor, urTriColor):
    return corner(patch(squareColor),
                  triangles(llTriColor, urTriColor))
```

```
DCG with Pics  30
```

And descend some more ...

```
def starQuadrant(squareColor, llTriColor, urTriColor):
    return corner(patch(squareColor),
                  triangles(llTriColor, urTriColor))
```

```
DCG with Pics  30
```

All together now

```
def star(innerColor, middleColor, outerColor):
    return rotations(starQuadrant(innerColor, middleColor, outerColor))
```

```
def quadrant():
    return corner(star('yellow', 'red', 'blue'),
                  star('red', 'green', 'blue'))
```

```
def starQuadrant(squareColor, llTriColor, urTriColor):
    return corner(patch(squareColor),
                  triangles(llTriColor, urTriColor))
```

```
DCG with Pics  32
```
Abstracting over quilt colors

How to generalize quilt to define quiltColors?

\[
\text{quiltColors}('yellow', 'red', 'green', 'blue') \\
\text{quiltColors}('red', 'blue', 'magenta', 'cyan')
\]

Challenge

How do we build this picture …

… starting with this primitive?