1 Administrative Topics

- Return the quizzes.
- There is no quiz this Friday.

2 Memory model of graphics code

The goal for today’s class was to gain a mental picture of the objects and lists that will be used in Project 6. We wrote code to handle scene components, which are really just lists of graphics objects. We developed functions that create specific, aggregate, objects and a function to draw any aggregate object.

2.1 Code

The component code is

```python
# Stephanie Taylor
# component.py
# A "component" is a list of graphics objects.
import graphics as gr
```
# draw a list of graphics object in the given window

def draw( objlist, win ):
    for item in objlist:
        item.draw( win )

# draw a funky ball at (x,y).
def ball_init( x, y, scale ):
    shapes = []
    b = gr.Circle( gr.Point(x,y), 20*scale )
    b.setFill( "blue" )
    b.setOutline( 'yellow' )
    b.setLineWidth( 4*scale )
    shapes.append(b)
    b = gr.Circle( gr.Point(x+15*scale,y), 20*scale )
    b.setFill( "blue" )
    b.setOutline( 'yellow' )
    b.setLineWidth( 4*scale )
    shapes.append(b)
    print "printing shapes"
    print shapes
    return shapes

def beach_init( x, y, scale ):
    shapes = []
    r = gr.Rectangle( gr.Point(x,y),
                     gr.Point(x+300*scale,y+130*scale) )
    r.setFill( gr.color_rgb(210,180,140) )
    shapes.append( r )
    return shapes
# Stephanie Taylor
# scene1.py

```python
import graphics as gr
import component as cmp

def main():
    win = gr.GraphWin( "Beach Scene", 300, 300 )

    ball = cmp.ball_init( 150, 150, 1 )
    beach = cmp.beach_init( 0, 170, 1 )
    scene_list = [beach, ball]

    for part in scene_list:
        cmp.draw( part, win )

    # close
    win.getMouse()
    win.close()

if __name__ == "__main__":
    main()
```

2.2 Stepping through the code

We begin by executing the top-level code. That puts gr, cmp, and main in the symbol table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr</td>
<td></td>
</tr>
<tr>
<td>cmp</td>
<td></td>
</tr>
<tr>
<td>main</td>
<td></td>
</tr>
</tbody>
</table>
Then line 22 of the scene module calls the main function. A new symbol table appears.

The first line of code in main is

```python
win = gr.GraphWin( "Beach Scene", 300, 300 )
```

After executing this line, the main function’s symbol is updated to include an arrow to the graphics window object:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr</td>
<td></td>
</tr>
<tr>
<td>cmp</td>
<td></td>
</tr>
<tr>
<td>main</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>win</td>
<td></td>
</tr>
</tbody>
</table>
```

![Diagram of symbol table updates](image)

5
The next line of code in `main` is

```python
ball = cmp.ball_init( 150, 150, 1 )
```

We will step through the call to `cmp.ball_init`. A new symbol table is created for it. The parameter values are entered into the table. The first line of code initializes the variable `shapes` to an empty list, so that is also added to the table:

![Symbol Table Diagram](image-url)
The next few lines of `ball_init` create the first circle and change its properties.

```python
b = gr.Circle( gr.Point(x,y), 20*scale )
b.setFill( "blue" )
b.setOutline( 'yellow' )
b.setWidth( 4*scale )
```

The first circle is created and stored with the name `b`:
The next line of `ball_init` is

```
shapes.append(b)
```

which updates the contents of the `shapes` list:
The next few lines of `ball_init` create the second circle and change its properties.

```python
b = gr.Circle( gr.Point(x+15*scale, y), 20*scale )
b.setFill( "blue" )
b.setOutline( 'yellow' )
b.setWidth( 4*scale )
```

The second circle is created and stored with the name `b`:
The next line of `ball_init` is

```
shapes.append(b)
```

which updates the contents of the `shapes` list:
The value of the `shapes` list is returned to the caller (`main`). That code adds it to the symbol table, using the name `ball`:
And we redraw the memory in a more compact way:
The next line of code in `main` is

```python
beach = cmp.beach_init( 0, 170, 1 )
```

cmp.beach_init is called. It returns a list of objects and the arrow to that list is stored with the name `beach`:

![Diagram showing the structure of `main` and `beach` objects]

```
main (top-level for scene1.py)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr</td>
<td></td>
</tr>
<tr>
<td>cmp</td>
<td></td>
</tr>
<tr>
<td>main</td>
<td></td>
</tr>
</tbody>
</table>

main

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>win</td>
<td></td>
</tr>
<tr>
<td>ball</td>
<td></td>
</tr>
<tr>
<td>beach</td>
<td></td>
</tr>
</tbody>
</table>
```
And we redraw the memory in a more compact way:
The next line of code in `main` is

```python
scene_list = [beach, ball]
```

which makes a list of lists. Below, we draw the arrows associated with `scene_list` in blue, to make it easier for the eye to follow the different arrows.
Next in `main` is the loop

```python
for part in scene_list:
    cmp.draw(part, win)
```

The first time through the loop, `part` refers to the ball list (we use a red arrow for this one):

![Diagram showing the `main` function call graph with `part` referring to `ball` list]

The code in the loop executes the `draw` function in `component.py`. This function will in turn loop through each object in the list, calling that object’s `draw` method.
The second time through the loop, `part` refers to the beach list (we use a red arrow for this one):

The code in the loop executes the `draw` function in `component.py`. This function will in turn loop through each object in the list, calling that object’s `draw` method.

Then we move on to the code

```python
    win.getMouse()
    win.close()
```

which waits for the mouse to be clicked, and then closes the window. Then the main function is done, and all of the code in the module has been executed.

Bye!