Drawing in Pygame
RGB Color

- Tristimulus theory:
  - Human retina uses 3 kinds of color receptors (cones)
  - So, colors can be created by adding red, green, blue light (primaries)
Pygame colors

• express colors as tuples (r,g,b)
• primary intensities are integers from 0 through 255

Table 4-1. Color Table

<table>
<thead>
<tr>
<th>Color</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Tuple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>Blue</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>(0, 0, 255)</td>
</tr>
<tr>
<td>Green</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>(0, 255, 0)</td>
</tr>
<tr>
<td>Cyan</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td>(0, 255, 255)</td>
</tr>
<tr>
<td>Red</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>(255, 0, 0)</td>
</tr>
<tr>
<td>Magenta</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td>(255, 0, 255)</td>
</tr>
<tr>
<td>Yellow</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>(255, 255, 0)</td>
</tr>
<tr>
<td>White</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>(255, 255, 255)</td>
</tr>
</tbody>
</table>
colortest.py
Scaling colors

- scale brightness (intensity) by scaling each primary separately

```python
def scale_color(color, scale):
    red, green, blue = color
    red = int(red*scale)
    green = int(green*scale)
    blue = int(blue*scale)
    return red, green, blue

fireball_orange = (221, 99, 20)
print fireball_orange
print scale_color(fireball_orange, .5)
```
Saturating colors

- if the value of a primary is greater than 255, python unhappy :-(
  - ex) scale_color ( (200, 100,55), 5.)
- saturate == clamp each primary at 255

```python
def saturate_color(color):
    red, green, blue = color
    red = min(red, 255)
    green = min(green, 255)
    blue = min(blue, 255)
    return red, green, blue
```
Blending colors

- make one color change gradually to another
- linear interpolation == "lerp"
- use a parameter ('factor') between 0.0 and 1.0

```python
def lerp(value1, value2, factor):
    return value1+(value2-value1)*factor

print lerp(100, 200, 0.)
print lerp(100, 200, 1.)
print lerp(100, 200, .5)
print lerp(100, 200, .25)
```
Blending colors

def blend_color(color1, color2, blend_factor):
    red1, green1, blue1 = color1
    red2, green2, blue2 = color2
    red = red1+(red2-red1)*blend_factor
    green = green1+(green2-green1)*blend_factor
    blue = blue1+(blue2-blue1)*blend_factor
    return int(red), int(green), int(blue)
Using Images

• JPEG (Joint Photographic Expert Group)
  • best for "real" images

• PNG (Portable Network Graphics)
  • supports transparency
  • best for "synthetic" images like fonts, logos

• also GIF(non-animated), BMP, PCX, TGA(uncompressed), LBM, PBM/PGM/PPM, XPM
Using Surfaces

- pygame uses surfaces to hold images:
  
  ```python
  surf = pygame.image.load(filename)
  ```

- create a blank surface:
  
  ```python
  surf = pygame.Surface((640,480))
  ```

- convert a surface to use the same pixel format as the screen (faster blitting):
  
  ```python
  surf = pygame.image.load(filename).convert()
  ```
Rectangles

- defined by top left corner, width and height
  - Rect (100, 200, 200, 150)
  - Rect ((100,200), (120,80))
Clipping

- set the "clip_rect" to limit drawing to a specific region of the screen
- all subsequent drawing commands only affect the clip_rect
Setting and Getting Pixels

• Setting pixel color
  • surf.set_at ( position, color )
  • random.py (in text)

• Getting pixel color
  • surf.get_at ( position )
Locking Surfaces

• all surfaces must be locked before access
• all drawing routines lock surfaces implicitly
• explicit locking in your code can speed it up

```python
screen.lock()
drawALotOfStuff()
screen.unlock()
```
Blitting

• `blit()` method copies from supplied surface (into self)

```python
screen.blit(background, (0,0))
```

• can specify that a only a portion be copied

```python
screen.blit(ogre, (320,240), (100*frame_no, 0, 100, 100) )
```
If we have an image containing several frames of an ogre walking, we could use something like this to blit it to the screen. By changing the value of `frame_no`, we can blit from a different area of the source surface.

**Drawing with Pygame**

We have used a few functions from the `pygame.draw` module in the preceding examples. The purpose of this module is to draw lines, circles, and other geometric shapes to the screen. You could use it to create an entire game without loading any images. The classic Atari game Asteroids is an example of a great game that just uses shapes drawn with lines. Even if you don't use the `pygame.draw` module to create a complete game, you will find it useful for experimenting when you don't want to go to the trouble of creating images. You could also use it to draw a debug overlay on top of your game when you need to visualize what is happening in your code.

The first two parameters for functions in `pygame.draw` are the surface you want to render to—which could be the screen (display surface) or a plain surface—followed by the color you want to draw in. Each draw function will also take at least one point, and possibly a list of points. A point should be given as a tuple containing the x and y coordinates, where (0, 0) is the top left of the screen.

The return value for these draw functions is a `Rect` object that gives the area of the screen that has been drawn to, which can be useful if we only want to refresh parts of the screen that have been changed. Table 4-2 lists the functions in the `pygame.draw` module, which we will cover in this chapter.

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>rect</td>
<td>Draws a rectangle</td>
</tr>
<tr>
<td>polygon</td>
<td>Draws a polygon (shape with three or more sides)</td>
</tr>
<tr>
<td>circle</td>
<td>Draws a circle</td>
</tr>
<tr>
<td>ellipse</td>
<td>Draws an ellipse</td>
</tr>
<tr>
<td>arc</td>
<td>Draws an arc</td>
</tr>
<tr>
<td>line</td>
<td>Draws a line</td>
</tr>
<tr>
<td>lines</td>
<td>Draws several lines</td>
</tr>
<tr>
<td>aaline</td>
<td>Draws an antialiased (smooth) line</td>
</tr>
<tr>
<td>aalines</td>
<td>Draws several antialiased lines</td>
</tr>
</tbody>
</table>
pygame.draw.rect()

• give a color, a Rect, and optional line width (else filled)
• colrects.py
**pygame.draw.polygon()**

- supply a list of points, optional line width
- polytest.py
pygame.draw.circle()

• supply radius and center

• circletest.py
pygame.draw.ellipse()

- give a Rect to hold the ellipse
- ellipsetest.py
pygame.draw.arc()

- give Rect, like ellipse, plus start and end angle in radians.
- arctest.py
pygame.draw.line()

- give start point and end point
- drawinglines.py
**pygame.draw.lines()**

- draw multiple, connected lines
  - supply a list of points, optional flag to close loop

**pygame.draw.aaline()**

- draws anti-aliased lines