Lecture 24

Callbacks & Stateful Controllers
Model-View-Controller Pattern

**Model**
- Defines and manages the data
- Responds to the controller requests

**Controller**
- Updates model in response to events
- Updates view with model changes

**View**
- Displays model to the user
- Provides interface for the controller

Division can apply to classes or modules

Calls the methods or functions of

11/20/12
Callbacks & Controllers
# MVC in this Course

## Model
- A3: Color classes
  - RGB, CMYK & HSV
- A4: Turtle, Pen
  - Window does the drawing
- A5: Matrix, Vector
- A6: ImageArray
- A7: Ball, Paddle, Bricks

## Controller
- A3: Functions in a3.py
  - No need for classes
- A4: Functions in a4.py
  - No need for classes
- A5: **Nothing you wrote**
- A6: ImageProcessor
- A7: Breakout
MVC in this Course

<table>
<thead>
<tr>
<th>Model</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A3: Color classes</td>
<td>• A3: Functions in a3.py</td>
</tr>
<tr>
<td>▪ RGB, CMYK &amp; HSV</td>
<td>▪ No need for classes</td>
</tr>
<tr>
<td>• A4: Turtle, Pen</td>
<td>• A4: Functions in a4.py</td>
</tr>
<tr>
<td>▪ Window does the drawing</td>
<td>▪ No need for classes</td>
</tr>
<tr>
<td>• A5: Matrix, Vector</td>
<td>• A5: Nothing you wrote</td>
</tr>
<tr>
<td>• A7: Ball, Paddle, Bricks</td>
<td>• A6: ImageProcessor</td>
</tr>
</tbody>
</table>

When need functions and when need classes?
A Standard GUI Application

- Animates the application, like a movie

**Update**
- Check for user input
- Process that user input
- Update the models

**Draw**
- Update the display
- *No major computation*
A Standard GUI Application

- **while-loop**
- **Event Loop**
- **Update**
  - Check for user input
  - Process that user input
  - Update the models
- **Draw**
  - Update the display
  - *No major computation*
- **Controller**
- **View**

Callbacks & Controllers
Must We Write this Loop Each Time?

while program_is_running:

    # Get information from mouse/keyboard
    # Handled by OS/GUI libraries

    # Your code goes here

    # Draw stuff on the screen
    # Handled by OS/GUI libraries
Must We Write this Loop Each Time?

```
while program_is_running:
    # Get information from mouse/keyboard
    # Handled by OS/GUI libraries
    Would like to “plug in” code
    # Your code goes here
    # Draw stuff on the screen
    # Handled by OS/GUI libraries
```

Why do we need to write this each time?
Function Names are Variables

• Calling a function
  § Provide arguments in ()
  § Executes the body

• Passing a function
  § Assign another variable
  § Use the name without ()

• Example:
  >>> x = greet
  >>> x('Walker')
  Hello Walker!

```python
def greet(n):
    print 'Hello '+n+'!'
```

Body stored in heap space
Callback Functions

- **Given**: predefined code that calls some function
  - But function not defined
  - You want to replace it with your function
- Assign that function to the name of yours
  - When called, it *calls back* to your function definition
  - *Sort of* like overriding
  - But can’t get old version

```python
callback = <your function>
...

while program_running:
    # Get input
    # Your code goes here
    callback()
    # Draw

See callback.py
```
### Application: Buttons

- Buttons in Kivy all have a special attribute
  - Named **on_press**
  - Stores a function
- Called on button press
  - Assign it what you want
- Standard for GUI apps
  - Libraries do hard work
  - Customize behavior w/ callback functions

```python
class ButtonMain(Widget):
    """Kivy window with a single button""
    
    def __init__(self, **kw):
        """Constructor: make panel w/ button""
        super(ButtonMain, self).__init__(**kw)
        button = Button(text='Click Me!',
                        size_hint=(1,1))
        self.add_widget(button)

        # Set the callback function
        button.on_press = self.my_callback

    def my_callback(self):
        """Function to call on button press.""
        print 'Hello World!'
```

11/20/12
Loop Invariants Revisited

Normal Loops

\[ \begin{align*}
  x &= 0 \\
  i &= 2 \\
  \# x &= \text{sum of squares of } 2..i \\
  \textbf{while} \ i \leq 5: \\
  &\quad x = x + i \times i \\
  &\quad i = i + 1 \\
  \# x &= \text{sum of squares of } 2..5
\end{align*} \]

Properties of “external” vars

Loops & Callbacks

What are the “external” vars?

\[ \begin{align*}
  \textbf{while} \ \text{program\_running}: \\
  &\quad \# \text{Get input} \\
  &\quad \# \text{Your code goes here} \\
  &\quad \text{callback()} \\
  &\quad \# \text{Draw}
\end{align*} \]
## Loop Invariants Revisited

### Normal Loops

- **x** = 0
- **i** = 2
- # x = sum of squares of 2..i

```
while i <= 5:
    x = x + i*i
    i = i + 1
# x = sum of squares of 2..5
```

**Properties of “external” vars**

### Loops & Callbacks

- What are the “external” vars?

```
while program_running:
    # Get input
    # Your code goes here
    callback()
    # Draw
```

If callback a method, then it has attributes
**Attribute Invariants = Loop Invariants**

- Fields are only way to store value between calls
  - Not part of call frame
  - Variables outside loop
- So all callback functions should be methods
  - Variable stores function definition **and** the object
  - Knows to call method on that particular object
  - Uses its fields for **state**

```python
callback = obj.method
...
# inv: obj attributes are ...
while program_running:
    # Get input
    # Your code goes here
    callback()
    # Draw
    # post: obj attributes are ...
```
Example: Animation

- **Callback**: `animate(...)`
  - Called 60x a second
  - Moves back and forth
- **Animate** is a method
  - Associated with an object
  - Object has changing **state**
- **Examples** of state
  - Ellipse position
  - Current velocity
  - Current animation step

```python
def animate(self, dt):
    """Animate the ellipse back & forth""
    if self._steps == 0:
        # Initialize
        ...
    elif self._steps > ANIMATION_STEPS/2:
        # Move away
        x = self._ellipse.pos[0]
        y = self._ellipse.pos[1]
        self._ellipse.pos = (x + self._vx, y + self._vy)
        self._steps = self._steps - 1
    else:
        # Move back
        x = self._ellipse.pos[0]
        y = self._ellipse.pos[1]
        self._ellipse.pos = (x - self._vx, y - self._vy)
        self._steps = self._steps - 1
```

11/20/12  Callbacks & Controllers  15
Example: Animation

- **Callback**: `animate(...)`
  - Called 60x a second
  - Moves back and forth
- **Animate** is a method
  - Associated with an object
  - Object has changing state
- **Examples** of state
  - Ellipse position
  - Current velocity
  - Current animation step

```python
def animate(self, dt):
    """Animate the ellipse back & forth""
    if self._steps == 0:
        # Initialize
        ...
    elif self._steps > ANIMATION_STEPS/2:
        # Move away
        x = self._ellipse.pos[0]
        y = self._ellipse.pos[1]
        self._ellipse.pos = (x + self._vx, y + self._vy)
        self._steps = self._steps - 1
    else:
        # Move back
        x = self._ellipse.pos[0]
        y = self._ellipse.pos[1]
        self._ellipse.pos = (x - self._vx, y - self._vy)
        self._steps = self._steps - 1
```

Kivy requires argument in animation callbacks

See `animate.py`
State Across Multiple Callbacks

- Sometimes have more than one callback function
- Example: touch events
  - `on_touch_down`: User presses mouse (or a finger); does not release
  - `on_touch_up`: Releases mouse (or finger)
  - `on_touch_move`: Moves mouse (or finger)
- State needed to track change in touch over time

See touch.py
State Across Multiple Callbacks

# None or previous touch
_anchor = None

def on_touch_down(self, touch):
    # Track touch state
    self._anchor = (touch.x, touch.y)

def on_touch_up(self, touch):
    # Nothing to track
    self._anchor = None

def on_touch_move(self, touch):
    if not self._anchor is None:
        self.drawLine(self._anchor[0], self._anchor[1],
                      touch.x, touch.y, LINE_COLOR)
    self._anchor = (touch.x, touch.y)