A Standard GUI Application

Animates the application, like a movie

- Update display/view
- No change to objects

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
    application.update()
    # Draw stuff on the screen
    # Handled by OS/GUI libraries
```

Custom Application class

- Method call (for loop body)
  - Write loop body in an app class.
  - OS/GUI handles everything else.

Loop Invariants Revisited

<table>
<thead>
<tr>
<th>Normal Loops</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 0</td>
<td>Properties of &quot;external&quot; vars</td>
</tr>
<tr>
<td>i = 2</td>
<td>What are the &quot;external&quot; vars?</td>
</tr>
<tr>
<td># x = sum of squares of 2..i</td>
<td>while program_running:</td>
</tr>
<tr>
<td>while i &lt;= 5:</td>
<td># Get input</td>
</tr>
<tr>
<td>x = x + i*i</td>
<td></td>
</tr>
<tr>
<td>i = i + 1</td>
<td></td>
</tr>
<tr>
<td># x = sum of squares of 2..6</td>
<td>application.update()</td>
</tr>
<tr>
<td># Draw</td>
<td></td>
</tr>
</tbody>
</table>

- Application is an object. It will have attributes!

Attribute Invariants = Loop Invariants

- Attributes are a way to store value between calls
  - Not part of call frame
  - Variables outside loop
- An application needs
  - Loop attributes
  - Initialization method (for loop, not __init__) 
  - Method for body of loop
- Attribute descriptions, invariants are important

Loop initialization

```
# Constructor
game = GameApp(...) ... game.start() #Loop initialization
```

```
while program_running:
    # Get input
    # Your code goes here 
    game.update(time_elapsed)
    game.draw()
    # post: game attributes are ... 
```

Attribute to Keep: Touch

- Attribute `touch` in GInput
  - The mouse press position
  - Or `None` if not pressed
  - Use `self.input.touch` inside your subclass definition
- Compare touch, `last` position
  - `last` None, `touch` not None: Mouse button `pressed`
  - `last` not None, `touch` None: Mouse button `released`
  - `last` and `touch` both not None: Mouse dragged (button down)

Example: Animation

```
class Animation(game2d.GameApp):
    """Application to an ellipse in a circle."""
    game = game2d.GameApp() ...
    def start(self):
        """Initializes the game loop."""
        ... 
    def update(self, dt):
        """Changes the ellipse position.""
        ... 
    def draw(self):
        """Draws the ellipse""
        ... 
```

Loop initialization

```
Parent class that does hard stuff
```

```
Loop body
```

```
Loop body
```

```
Loop body
```

What Attributes to Keep: Touch

```
Current Touch
```

```
Previous Touch
```

```
See touch.py
```

```
Line segment = 2 points
```

```
See animation.py
```

```
Use method draw() defined in GObject
```

```
Use method draw() defined in GObject
```

```
Use method draw() defined in GObject
```

```
Use method draw() defined in GObject
```
State: Changing What the Loop Does

- **State**: Current loop activity
  - Playing game vs. pausing
  - Ball countdown vs. serve

- Add an attribute `state`
  - Method `update()` checks state
  - Executes correct helper

- How do we store state?
  - State is an *enumeration*; one of several fixed values
  - Implemented as an `int`
  - Global constants are values

Designing States

- Each state has its *own set* of invariants.
  - **Drawing?** Then `touch` and `last` are not `None`
  - **Erasing?** Then `touch` is `None`, but `last` is not

- Need rules for when we switch states
  - Could just be “check which invariants are true”
  - Or could be a *triggering event* (e.g. key press)

- Need to make clear in class specification
  - What are the invariants *for each state*?
  - What are the rules to switch to a new state?

Triggers: Checking Click Types

- Double click = 2 fast clicks
- Count number of fast clicks
  - Add an attribute `clicks`
  - Reset to 0 if not fast enough
- Time click speed
  - Add an attribute `time`
  - Set to 0 when mouse released
  - Increment when not pressed (e.g. in loop method `update()`)
  - Check `time` when next pressed

Designing Complex Applications

- Applications can become extremely complex
  - Large classes doing a lot
  - Many states & invariants
  - Specification unreadable

- **Idea**: Break application up into several classes
  - Start with a “main” class
  - Other classes have roles
  - Main class delegates work

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 the model to the app user
  - Provides user input to the controller

Calls the methods or functions of

Division can apply to classes or modules

Model-View-Controller in CS 1110

- **Model**
  - Subclasses of `GObject`
  - Often more than one

- **View**
  - Class `GView`, `GInput`
  - Do not subclass!
  - Part of `GameApp`

- **Controller**
  - Subclass of `GameApp`
  - Method `draw` in `GObject`

- Other attributes (defined by you)
  - (inherited)