A Standard GUI Application

Check for user input
Process user input
Update the objects

No change to objects

Update display/view

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

Write loop body in an app class.
OS/GUI handles everything else.

Loop Invariants Revisited

Normal Loops

\[
\begin{align*}
    x &= 0 \\
    i &= 2 \\
    \# \ x &= \text{sum of squares of } 2..i \\
\end{align*}
\]

Application

What are the “external” vars?

while program_running:

# Get input
# Your code called here
application.update()

# Draw

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

Example: Animation

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

What Attributes 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)

Current Touch
Previous Touch
Line segment = 2 points

See animation.py

Parent class that does hard stuff
Loop initialization
Do NOT use __init__
Loop body
Use method draw() defined in GObject

See touch.py
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

- Division can apply to classes or modules
- **Model**
  - Defines and manages the data
  - Responds to the controller requests
- **View**
  - Displays the model to the app user
  - Provides user input to the controller
- **Controller**
  - Updates model in response to events
  - Updates view with model changes
  - Calls the methods or functions of

Model-View-Controller in CS 1110

- Other attributes (defined by you)
- **Model**
  - Subclasses of `GObject`
    - `GEllipse`, `GImage`, ...
    - Often more than one
- **Controller Subclass of GameApp**
  - Method `draw in GObject`
  - Attribute `view` (inherited)
- **View**
  - Class `GView, GInput`
  - Do not subclass!
  - Part of GameApp
  - Classes in `game2d`