A Standard Game Application

Event Loop

while-loop

Controller

Update

Check for user input
Process user input
Update the models

Draw

Update display/view
No change to models

View

Must We Write this Loop Each Time?

while program_is_running:

# Get information from mouse/keyboard
# Handled by OS/GUI libraries
controller.update()
controller.draw()

# 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
controller.update()
controller.draw()

# Code to respond to user input
# Code to draw stuff in the window
# Code to respond to keyboard/mouse input
# Code to draw stuff in the window

Why do we need to write this each time?

Method calls for loop body

• No: make this loop part of the library!
• Then we only have to provide the controller class.

Model-View-Controller Pattern

Division can apply to classes or models

Controller

• Updates model in response to events
• Updates view with model changes

Model

• Defines and manages the data
• Responds to the controller requests

View

• Displays model to the user
• Provides interface for the controller

Model-View-Controller in A5

Controller

Subclass of Game

Purple because it’s also part of a View

Attribute view (inherited)

Other attributes (defined by you)

Model

Subclasses of GObject
• Ellipse, GImage, …
• Comprises many objects
• Knows how to draw

View

Class GObject
• Handles details of drawing
• Provided in Game

Example: Animation

class Animation(Game):
    """Application to an ellipse in a circle."""

    def __init__(self):
        """Special loop initialization method."""
        ...

    def update(self, dt):
        """Change the ellipse position."""
        ...

    def draw(self):
        """Draw the ellipse."""
        ...

See animation.py
Example: Animation

```python
class Animation(Game):
    """Application to an ellipse in a circle."""
    def __init__(self):
        """Special loop initialization method."""
        ...  
def update(self, dt):
        """Change the ellipse position."""
        ...  
def draw(self):
        """Draw the ellipse"""
        ...
```

What Attributes to Keep: Touch

- Attribute `touch` in `GView`
  - The mouse press position
  - Or None if not pressed
  - Use `self.view.touch` inside controller (`Game`) methods
- Compare `touch`, `last` position
  - `last` None, `touch` not None: Mouse button `pressed`
  - `last` not None, `touch` None: Mouse button `released`
  - `last` and `touch` not None: Mouse dragged (button down)

More Attributes: 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

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

Types of Models for Assignment 6

- Often subclass of `GObject`
  - Has built-in draw method
  - See documentation in `A6`
- Includes groups of models
  - Example: rockets in `pyro.py`
  - Each rocket is a model
  - But so is the entire list!
  - `update()` will change both
- A5: Model class
  - Container, like A4 `Dataset`
  - Holds bricks, ball, paddle

Importance of class invariants

- State `ANIMATE_CIRCLE`
- State `ANIMATE_HORIZONTAL`
- Line segment = 2 points
- Time `pressed` when next pressed
- Is it fast enough?

See `animation.py`
See `touch.py`
See `state.py`
See `pyro.py`