Lecture 24

GUI Applications
Announcements for This Lecture

Prelim 2

- Difficulty was just right
  - **Mean**: 74, **Median**: 76
  - Actually expected lower
- What do grades mean?
  - **A**: 80s+
  - **B**: 60s+
  - **C**: 25+
- Final will be about same
  - Some easier, some harder

Assignments

- A6 due **TOMORROW**
  - You are welcome
  - Also, fill out survey
- A7 due **December 4**
  - Instructions posted today
  - Focus of today’s lecture
  - 2.5 weeks including T-Day
  - 2 weeks without the break
- Both are **very important**
  - Each worth 8% of grade

11/15/16
The Experience Factor

• Will consider experience in final grades
  ▪ 39% No experience (G1)
  ▪ 38% Some experience (G2)
  ▪ 23% AP or equivalent (G3)

• Prelim 1: Mean 78, Median 83
  ▪ Group Means: G1 75, G2 78, G3 83

• Prelim 2: Mean 74, Median 76
  ▪ Group Means: G1 71, G2 74, G3 78

• Difference is noticeable, but adjustable
A Standard GUI Application

Animates the application, like a movie
A Standard GUI Application

Animates the application, like a movie

Update

Draw

Check for user input
Process user input
Update the objects
A Standard GUI Application

Animates the application, like a movie

- Update:
  - Check for user input
  - Process user input
  - Update the objects

- Draw:
  - Update display/view
  - No change to objects
Must We Write this Loop Each Time?

```python
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
    # Your code goes here

    # Draw stuff on the screen
    # Handled by OS/GUI libraries
```

Why do we need to write this each time?

Would like to “plug in” code
Must We Write this Loop Each Time?

```python
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
```

- Write loop body in an app class.
- OS/GUI handles everything else.
Loop Invariants Revisited

Normal Loops

\[ x = 0 \]
\[ i = 2 \]
\[ \# \ x = \text{sum of squares of } 2..i-1 \]

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

Application

What are the “external” vars?

\[ \textbf{while} \ \text{program\_running}: \]
\[ \quad \# \ \text{Get input} \]
\[ \quad \# \ \text{Your code called here} \]
\[ \quad \text{application.update()} \]
\[ \quad \# \ \text{Draw} \]
Loop Invariants Revisited

Normal Loops

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

Properties of "external" vars

# x = sum of squares of 2..i

Application

What are the "external" vars?

\[
\begin{align*}
\text{while program_running:} \\
&\quad \text{# Get input} \\
&\quad \text{# Your code called here} \\
&\quad \text{application.update()} \\
&\quad \text{# Draw}
\end{align*}
\]

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

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

# inv: game attributes are ...

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

# post: game attributes are ...
```
Example: Animation

class Animation(game2d.GameApp):
    """Application to 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"""
        ...

See animation.py
class Animation(game2d.GameApp):
    """Application to 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"""
    ...

See animation.py

Parent class that does hard stuff
Example: Animation

class Animation(game2d.GameApp):

    """Application to 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"""

        ...

See animation.py

Parent class that does hard stuff

Loop initialization
Do NOT use __init__

Loop body

Use method draw() defined in GObject
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)

See `touch.py`
Input and Invariants

• Attribute `input` is…
  ▪ A `GInput` object

• Attribute `input.touch` is…
  ▪ Either a `GPoint` or None
  ▪ Location of mouse cursor (if it is pressed)

• Attribute `last` is…
  ▪ Either a `GPoint` or None
  ▪ `input.touch` in prev. frame

Line segment = 2 points

Relationship between two variables.

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

State `ANIMATE_CIRCLE`

State `ANIMATE_HORIZONTAL`

See `state.py`
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

Is it fast enough?

See `touch.py`
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

`MainApp`
- Processes input
- Determines state
- Calls the methods of
  - Animates (only)

See `subcontroller.py`

11/15/16
GUI Applications
How to Break Up: Software Patterns

- **Pattern**: reusable solution to a common problem
  - Template, not a single program
  - Tells you how to design your code
  - Made by someone who ran into problem first

- In many cases, a pattern gives you the interface
  - List of headers for non-hidden methods
  - Specification for non-hidden methods
  - Only thing missing is the implementation
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

Division can apply to classes or modules.

Calls the methods or functions of...
MVC in this Course

Model

- **A3**: Color classes
  - RGB, CMYK & HSV
- **A4**: Turtle, Pen
  - Window is View
- **A6**: Database, Cluster
  - Data is always in model
- **A7**: Ball, Brick, etc..
  - All shapes/geometry

Controller

- **A3**: a3app.py
  - Hidden classes
- **A4**: Functions in a4.py
  - No need for classes
- **A6**: ClusterGroup
  - Also visualizer
- **A7**: Breakout
  - Controller class for you!
### MVC in this Course

#### Model
- **A3**: Color classes
  - RGB, CMYK & HSV
- **A4**: Turtle, Pen
  - Window is View
- **A6**: Database, Cluster
  - Data is always in model
- **A7**: Ball, Brick, etc..
  - All shapes/geometry

#### Controller
- **A3**: a3app.py
  - Hidden classes
- **A4**: Functions in a4.py
  - No need for classes
- **A6**: ClusterGroup
  - Also visualizer
- **A7**: Breakout
  - Controller class for you!

---

**Why classes sometimes and functions others?**
Model-View-Controller in CS 1110

Controller
Subclass of GameApp

Model
Subclasses of GObject
- GEllipse, GImage, …
- Often more than one

Method `draw` in GObject

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

Classes in `game2d.py`

Other attributes (defined by you)

Attribute view (inherited)

GUI Applications 26
Model-View-Controller in CS 1110

- **Model**
  - Subclass of GameApp
  - Neglected for most of this lecture

- **Controller**
  - Subclass of GObject

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

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

Classes in `game2d.py`

**GUI Applications**

11/15/16
Models in Assignment 7

- 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
- **A7**: Several model classes
  - Ball to animate the ball
  - `BrickWall` to manage bricks

11/15/16

GUI Applications 28