Lecture 24

GUI Applications
Announcements for This Lecture

Prelim 2

- Prelim, Nov 21st at 7:30
  - See webpage for rooms
  - Review Sun Nov. 19 at 6pm
- Material up to Nov. 9
  - Recursion + Loops + Classes
  - Study guide is now posted
- Conflict with Prelim?
  - Submit them NOW
  - Decisions this week

Assignments

- A5 has now been graded
  - Mean: 47.4  Median: 49
  - A: 47 (79%), B: 40 (15%)
  - Scores are very typical
  - Study soln for the exam
- A6 due TONIGHT
  - Last minute extension
  - Also, fill out survey
  - Worth 8% of your grade

11/14/23
Announcements for This Lecture

Labs

• There is a lab today
  ▪ Jump starts you on A7
  ▪ But no manual grade
• Lab Thurs/Fri is special
  ▪ MUST show up to get credit
  ▪ Or show far enough in A7
• NO LAB on Tuesday
• Nov 28/29 is LAST LAB
  ▪ Will have manual grade
  ▪ Finish it by end of week

Assignment 7

• Due December 4th
  ▪ 2.5 weeks excluding T-Day
  ▪ 3 weeks including the break
• Focus of today’s lecture

Video Lessons

• Lesson 27 (all) for today
• Lesson 30 for next time
• Will return to Lesson 28 later
A Standard GUI Application

Animates the application, like a movie
A Standard GUI Application

Animations 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

Restriction set by graphics cards
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
    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?
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

Custom Application class with its own attributes
```

- Write loop body in an app class.
- OS/GUI handles everything else.
But There is a Catch

```
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
```

This creates a **call frame**

All its variables are **erased** when done
Programming Animation

Intra-Frame

- Computation within frame
  - Only need current frame
- **Example:** Collisions
  - Need current position
  - Use to check for overlap
- Can use **local variables**
  - All lost at `update()` end
  - But no longer need them
Programming Animation

Inter-Frame

- Computation across frames
  - Use values from \textit{last} frame
- \textbf{Example}: Movement
  - Need old position/velocity
  - Compute next position
- Requires \textit{attributes}
  - Attributes never deleted
  - Remain after \texttt{update()} ends
Variables and the Loop

```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
```

Local variables erased. But attributes persist.
# Programming Animation

<table>
<thead>
<tr>
<th>Intra-Frame</th>
<th>Inter-Frame</th>
</tr>
</thead>
</table>
| • Computation within frame  
  - Only need current frame  
  - **Example:** Collisions  
    - Need current position  
    - Use to check for overlap  
  - Can use **local variables**  
    - All lost at `update()` end  
    - But no longer need them  |  
| • Computation across frames  
  - Use values from last frame  
  - **Example:** Movement  
    - Need old position/velocity  
    - Compute next position  |  
| • Requires **attributes**  
  - Attributes never deleted  
  - Remain after `update()` ends |  

11/14/23
Attributes = Loop Variables

Normal Loops

Variables “external” to the loop body

\[
x = 0 \\
i = 2 \\
# x = \text{sum of squares of } 2..i-1 \\
while \ i \leq 5: \\
\quad x = x + i \times i \\
\quad i = i + 1 \\
# x = \text{sum of squares of } 2..5
\]

Application

Attributes are the “external” variables

\[
while \ \text{program\_running}:
\begin{align*}
&\quad \# \ \text{Get input} \\
&\quad \# \ \text{Your code called here} \\
&\quad \text{application.update()} \\
&\quad \# \ \text{Draw}
\end{align*}
\]
The Actual Game Loop

# Constructor
game = GameApp(...)  

...  
game.start() # Loop initialization

while program_running:
    # Get input
    # Your code goes here
    game.update(time_elapsed)
    game.draw()
Designing a Game Class: Animation

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"""
        ...

See animation.py
class Animation(game2d.GameApp):
    """App to animate an ellipse."""

    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
Designing a Game Class: Animation

class Animation(game2d.GameApp):
    """App to animate an ellipse."""

    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
Interframe Computation: Touch

- Works like an Etch-a-Sketch
  - User draws by touching
  - Checks position each frame
  - Draws lines between touches
- Uses attribute `touch` in `GInput`
  - The mouse press position
  - Or `None` if not pressed
  - Access with `self.input.touch`
- But we also need last touch!
  - Forgot if we do not store it
  - Purpose of attribute `last`

Line segment = 2 points

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

State `ANIMATE_CIRCLE`

State `ANIMATE_HORIZONTAL`

See `state.py`
States and the Class Invariant

• Think of each state as a mini-program
  ▪ Has its own update functionality/logic
  ▪ Usually separated out as helper to update
  ▪ update uses ifs to send to correct helper

• Need to include in the class invariant
  ▪ Some attributes only used in certain states
  ▪ What values must they have in other states?

• Also need rules for when we switch states
  ▪ Could be the result of an event (e.g. game over)
  ▪ Could be the result of an input (e.g. a key press)
# Checking Input

## Keyboard

- **is_key_down(key)**
  - Returns True if key is down
  - key is a string ('a' or 'space')
  - Empty string means *any* key
- **is_key_pressed(key)**
  - Returns True if key pressed
  - key **not** down prev. frame
- **is_key_released(key)**
  - Returns True if key released
  - key was down prev. frame

## Mouse/Touch

- **touch**
  - *Attribute* giving a position
  - Stored as a Point2 object
  - But None if no touch
- **is_touch_pressed()**
  - True if touch pressed
  - touch was None prev. frame
- **is_touch_released()**
  - True if touch released
  - touch **not** None prev. frame
# Checking Input

## Keyboard

- **is_key_down**(key)
  - Returns True if key is down
  - key is a string ('a' or 'space')
  - Empty string means any key
- **is_key_pressed**(key)
  - Returns True if key pressed
  - key not down prev. frame
- **is_keyReleased**(key)
  - Returns True if key released
  - key was down prev. frame

## Mouse/Touch

- **touch**
  - Attribute giving a position
  - Stored as a Point2 object
  - None if no touch
- **is_touch_pressed()**
  - True if touch pressed
  - touch was None prev. frame
- **is_touch_released()**
  - True if touch released
  - touch not None prev. frame

All accessed from `self.input` in App
Complex Input: 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

- Animation
  - Animates (only)

See subcontroller.py
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

11/14/23
GUI Applications
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**: Player, Board
  - Data is always in model
- **A7**: Ship, Alien, etc..
  - All shapes/geometry

Controller

- **A3**: a3app.py
  - Hidden classes
- **A4**: Functions in a4.py
  - No need for classes
- **A6**: Game
  - Drives program forward
- **A7**: Invaders, Wave
  - Main part of assignment!
### MVC in this Course

#### Model
- **A3**: Color classes
  - RGB, CMYK & HSV
- **A4**: Turtle, Pen
  - Window is View
- **A6**: Player, Board
  - Data is always in model
- **A7**: Ship, Alien, etc..
  - All shapes/geometry

#### Controller
- **A3**: a3app.py
  - Hidden classes
- **A4**: Functions in a4.py
  - No need for classes
- **A6**: Game
  - Drives program forward
- **A7**: Invaders, Wave
  - Main part of assignment!

---

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

Controller
Subclass of
GameApp

Attribute view
(inherited)

Method draw
in GObject

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

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

Classes in
game2d

Other attributes
(defined by you)

GUI Applications
11/14/23
Subclasses of GObject

• GEllipse, GImage, ...

• Often more than one View Class
  • GView, GInput
  • Do not subclass!
  • Part of GameApp

• Other attributes (defined by you)

Neglected for most of this lecture

Classes in game2d

Method draw in GObject

Attribute view (inherited)

Controller Subclass of GameApp

Model Subclass of GameApp

Model-View-Controller in CS 1110
Models in Assignment 7

- Often subclass of GObject
  - Has built-in draw method
- 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
  - Ship to animate the player
  - Alien to represent an alien

See pyro.py