Lecture 24

Controllers and Object Oriented Design
## Announcements for This Lecture

### Assignments

- **A6 due midnight** **TONIGHT**
  - Last day for consultants
  - Also, fill out survey
- **A7 due Wednesday, Dec. 10**
  - Instructions posted today
  - Focus of today’s lecture
  - 3 weeks including T-Day
  - 2.5 weeks without the break
- **Both are very important**
  - Each worth 8% of grade

### Next Week

- There is no lab next week
  - But Tuesday hours are open
  - Open to **EVERYONE**
  - Go for help on lab or A7
- But lecture is **important**
  - Continue Tuesday’s topic
  - Setting us up for sorting
- Try to finish lab 12 first
  - Not due, but helps lecture
Computer Game Development

Credits: Planetfall (1983)

Steve Meretzky
Computer Game Development

Credits: Planetfall (1983)
Steve Meretzky

Credits: Portal (2007)
Challenge: Breaking Up Software

Game Engine
- Input Devices
- Discrete Simulation Engine

Programmer
- Physics Engine
- AI Engine (e.g. Pathfinding)

Game Content
- Character Scripts
- Character Data
- UI Elements
- Models and Textures
- Sounds

Player
- GUI
- Rendering Engine
- Audio Engine

11/20/14
Challenge: Breaking Up Software
Challenge: Breaking Up Software

Game Engine

- Input Devices
- GUI
- Rendering Engine
- Audio Engine

Game Content

- Character Scripts
- Character Data
- UI Elements
- Models and Textures
- Sounds

Programmer

- Physics Engine
- AI Engine (e.g. Pathfinding)

Designer or Modder

- Compiler
- Data Management Layer

Written by one person/group

By another person/group

MEETINGS!
Coordinating groups must agree on the specifications of functions in the module.
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

Just like this course!
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 model to the user
- Provides interface for 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
- **A6**: Database, Cluster
  - Data is always in model
- **A7**: Ball, Brick, etc..
  - All shapes/geometry

**Controller**
- **A3**: a3app.py
  - Hidden classes
- **A6**: ClusterGroup
  - Also visualizer
- **A7**: Breakout
  - Controller class for you!
MVC in Previous Semesters

**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 Previous Semesters

### 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?**

---

11/20/14
A Standard GUI Application

Animates the application, like a movie
A Standard GUI Application

Animates the application, like a movie

Update

Check for user input
Process user input
Update the models

Draw
A Standard GUI Application

Animates the application, like a movie

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

Draw:
- Update display/view
  - No change to models
A Standard GUI Application

while-loop

Event Loop

Controller

Update

Check for user input
Process user input
Update the models

Draw

Update display/view
No change to models

View

OO Design 17
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?

```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
```

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

- Write loop body in a controller.
- OS/GUI handles everything else.

Controller object

Method call (for loop body)
Loop Invariants Revisited

Normal Loops

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

Properties of "external" vars

Controller

What are the "external" vars?

\[
\textbf{while } \text{program\_running}: \\
\quad \text{# Get input} \\
\quad \text{# Your code called here} \\
\quad \text{controller.update()} \\
\quad \text{# Draw}
\]
Normal Loops

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

Properties of “external” vars

Controller

What are the “external” vars?

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

controller is an object. It will have attributes!
Model-View-Controller in CS 1110

Controller
Subclass of Game

Attribute view
(inherited)

Other attributes
(defined by you)

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

Method draw
in GObject

View
Class GView
- Do not subclass!
- Provided in Game

Classes in
game2d.py

11/20/14
OO Design
Attribute Invariants = Loop Invariants

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

```python
#constructor
game = Game(...)  
...

game.init()  
# 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(GameApp):
    """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(GameApp):
    """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"""
...
```

Parent class that does hard stuff

See animation.py

Loop initialization
Do NOT use `__init__`

Loop body

Use method `draw()` defined in GObject
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)

See `touch.py`
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

Is it fast enough?

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

[Diagram showing state transitions: ANIMATE_CIRCLE -> ANIMATE_HORIZONTAL]

See `state.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

Importance of class invariants
Types of Models for 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