Lecture 23

Controllers and Object Oriented Design
# Announcements for This Lecture

## Exams

- Not so top heavy this time
  - Mean: 74, **Median**: 77
  - For-loop, not recursion hard
- Good grade distribution
  - **A**: Mid 80s up
  - **B**: Mid-low 60s to mid 80s
  - **C**: 35 to mid-low 60s
- Final should be similar
  - More time, more questions

## Assignment & Lab

- A5 is due tonight!
  - Grading will be curved
  - Parts A-F get at least A-
- Finish Survey by Friday
  - New questions this time!
- Today’s lab is on invariants
  - Due before Thanksgiving
  - No official lab next week
  - But will be there on Tues
Computer Game Development

Credits: Planetfall (1983)

Steve Meretzky
Steve Meretzky
Challenge: Breaking Up Software
Challenge: Breaking Up Software

[Diagram showing a breakdown of software components into modules, with labels for Game Engine, Physics Engine, AI Engine (e.g., Pathfinding), Compiler, Data Management Layer, Game Content, Character Scripts, Character Data, UI Elements, Models and Textures, Sounds, GUI, Rendering Engine, Audio Engine, Input Devices, and a note on importing modules.]
Challenge: Breaking Up Software

Written by one person/group

By another person/group

MEETINGS!
Challenge: Breaking Up Software

Coordinating groups must agree on what the headers of the functions look like.
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 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
- **A4**: Turtle, Pen
  - Window is View
- **A5**: Database, Cluster
  - Data is always in model
- **A6**: Ball, Brick, etc..
  - All shapes/geometry

Controller

- **A3**: `a3app.py`
  - Hidden classes
- **A4**: Functions in `a4.py`
  - No need for classes
- **A5**: `best_cluster`
  - But visualizer is a class
- **A6**: Breakout
  - Controller class for you!
MVC in this Course

Model

- **A3**: Color classes
  - RGB, CMYK & HSV
- **A4**: Turtle, Pen
  - Window is View
- **A5**: Database, Cluster
  - Data is always in model
- **A6**: Ball, Brick, etc..
  - All shapes/geometry

Controller

- **A3**: a3app.py
  - Hidden classes
- **A4**: Functions in a4.py
  - No need for classes
- **A5**: best_cluster
  - But visualizer is a class
- **A6**: Breakout
  - Controller class for you!

Why *classes* sometimes and *functions* others?
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

- **Event Loop**
- **while-loop**
- **Update**
  - Check for user input
  - Process user input
  - Update the models
- **Draw**
  - Update display/view
  - *No change to models*
- **Controller**

OO Design
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

Would like to “plug in” code

Why do we need to write this each time?
```
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
    controller.doloop()
    # Draw stuff on the screen
    # Handled by OS/GUI libraries

- Write loop body in a controller.
- OS/GUI handles everything else.
## Loop Invariants Revisited

### Normal Loops

- **x = 0**
- **i = 2**
- **# x = sum of squares of 2..i**

```
while i <= 5:
    x = x + i*i
    i = i + 1
# x = sum of squares of 2..5
```

### Properties of “external” vars

- x: sum of squares of 2..i
- i: counter

### Controller

- **What are the “external” vars?**

```
while program_running:
    # Get input
    # Your code called here
    controller.doloop()
    # Draw
```
Loop Invariants Revisited

Normal Loops

- x = 0
- i = 2
- # x = sum of squares of 2..i
- while i <= 5:
  - x = x + i*i
  - i = i + 1
- # x = sum of squares of 2..5

Properties of “external” vars

Controller

- What are the “external” vars?
- while program_running:
  - # Get input
  - # Your code called here
  - controller.doloop()
  - # Draw

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

Controller Subclass of Game

Other attributes (defined by you)

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

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

Attribute view (inherited)

Method draw in GObject

Classes in game2d.py

OO Design 22
11/19/13
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
game = Game(...) #constructor
...

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(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

Loop initialization
Do NOT use __init__

Loop body

Use method draw() defined in GObject
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"""
        ...
```

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 `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)

Line segment = 2 points

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

See `state.py`

11/19/13
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

See `state.py`
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
- **A6**: Model class
  - Container, like Database
  - Holds bricks, ball, paddle

See `pyro.py`