

Lecture 24

GUI Applications

Announcements for This Lecture

Assignments

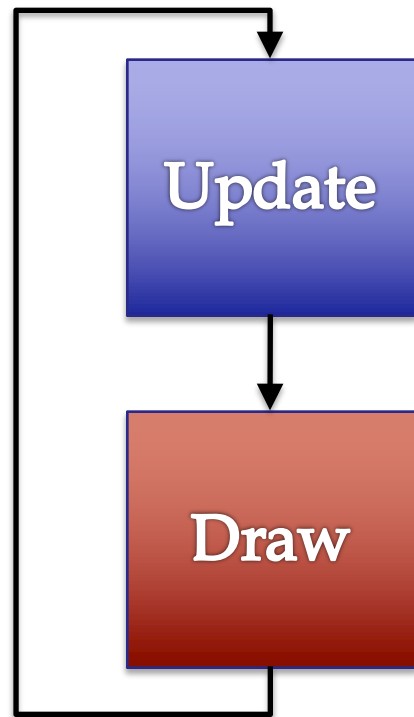
- A6 due midnight **TONIGHT**
 - Last day for consultants
 - Also, fill out survey
- A7 due **December 11**
 - Instructions posted today
 - Focus of today's lecture
 - 3.5 weeks including T-Day
 - 3 weeks without the break
- Both are **very important**
 - Each worth 8% of grade

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
 - Frees remaining time for A7

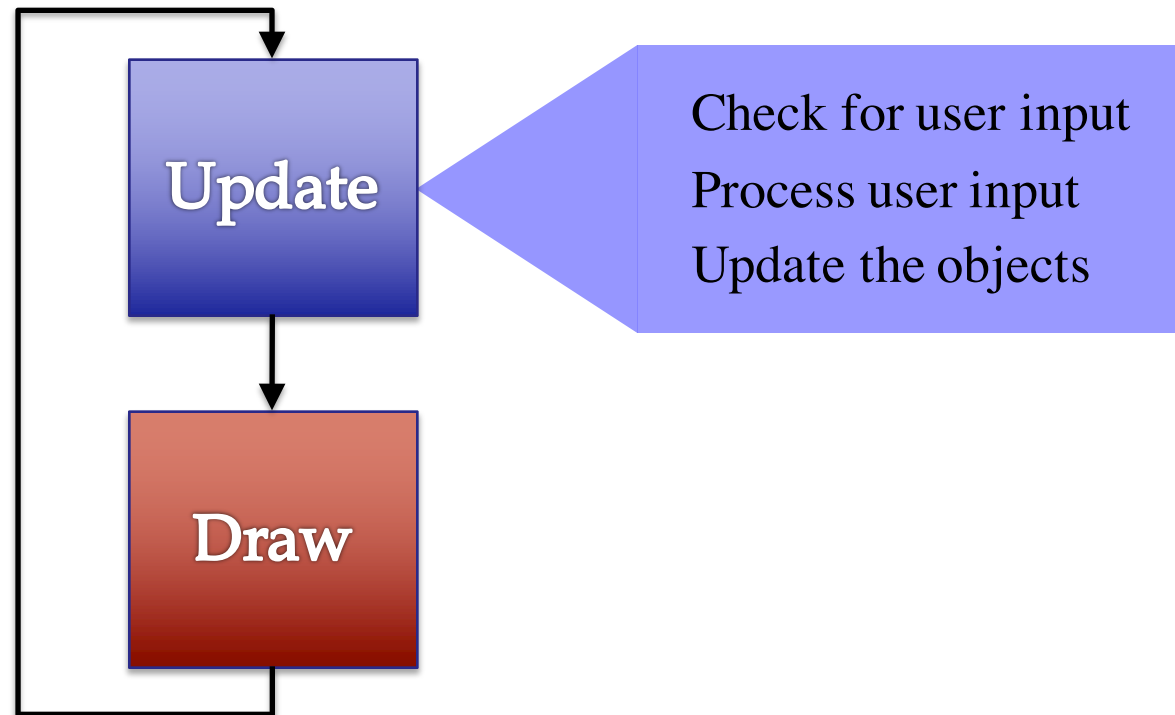
A Standard GUI Application

Animates the application,
like a movie



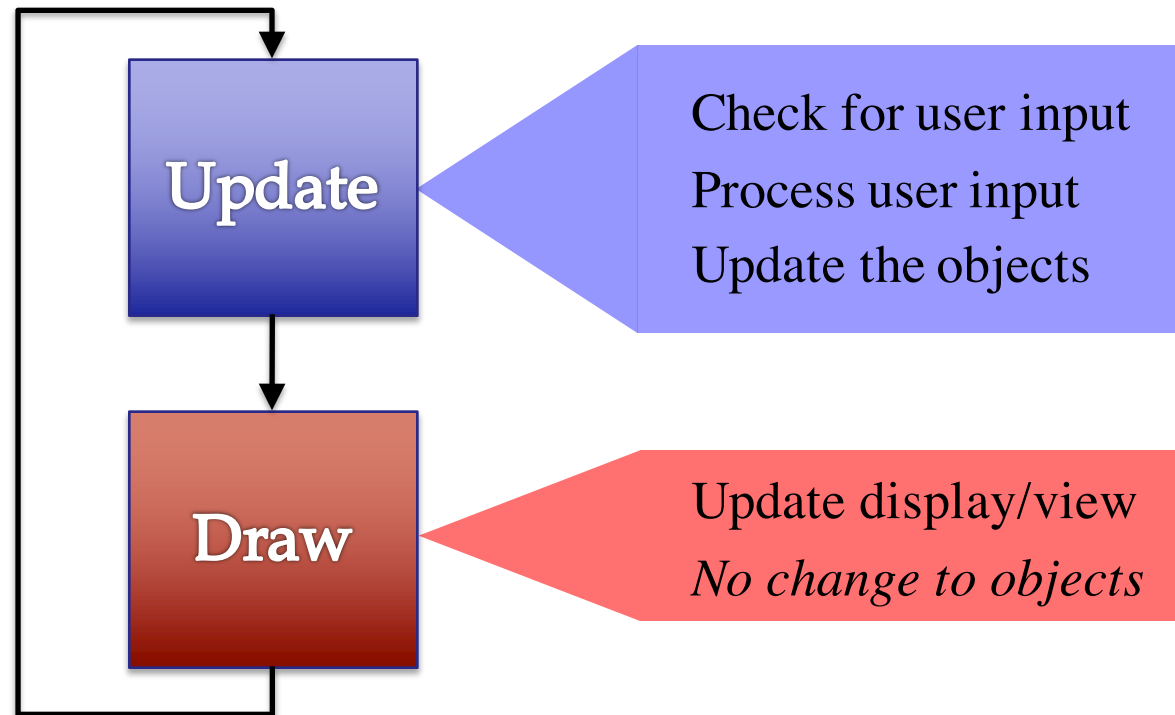
A Standard GUI Application

Animates the application,
like a movie



A Standard GUI Application

Animates the application,
like a movie



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
```

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
```

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`

Method call
(for loop body)

`application.update()`

Custom Application class

been

`# 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 = 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

Application

What are the
“external” vars?

```
while program_running:
    # Get input
    # Your code called here
    application.update()
    # 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

Application

What are the
“external” vars?

```
while program_running:
```

```
    # Get input
```

```
    # Your code called here
```

```
    application.update()
```

```
    # Draw
```

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

```
# 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](#)

Example: Animation

```
class Animation(game2d.GameApp):
```

```
    """Application to an ellipse"""
```

Parent class that
does hard stuff

See animation.py

```
    def start(self):
```

```
        """Initializes the game loop."""
```

```
        ...
```

```
    def update(self,dt):
```

```
        """Changes the ellipse position."""
```

```
        ...
```

```
    def draw(self):
```

```
        """Draws the ellipse"""
```

```
        ...
```

Example: Animation

```
class Animation(game2d.GameApp):
```

See animation.py

```
    """Application to an ellipse"""
```

Parent class that
does hard stuff

```
    def start(self):
```

```
        """Initializes the game loop."""
```

```
        ...
```

Loop initialization
Do NOT use `__init__`

```
    def update(self,dt):
```

```
        """Changes the ellipse position."""
```

```
        ...
```

Loop body

```
    def draw(self):
```

```
        """Draws the ellipse"""
```

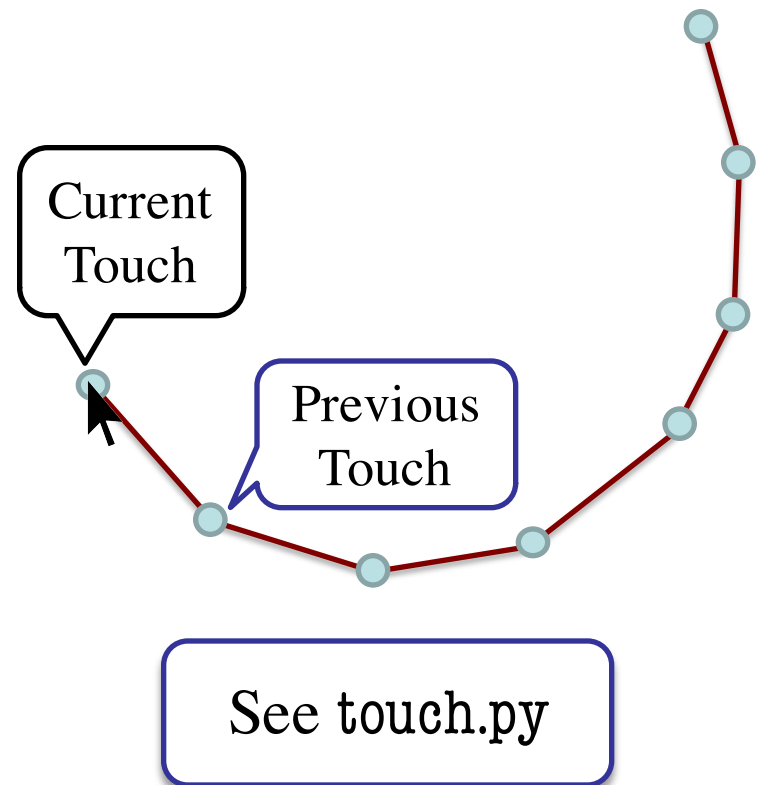
```
        ...
```

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)

Line segment = 2 points

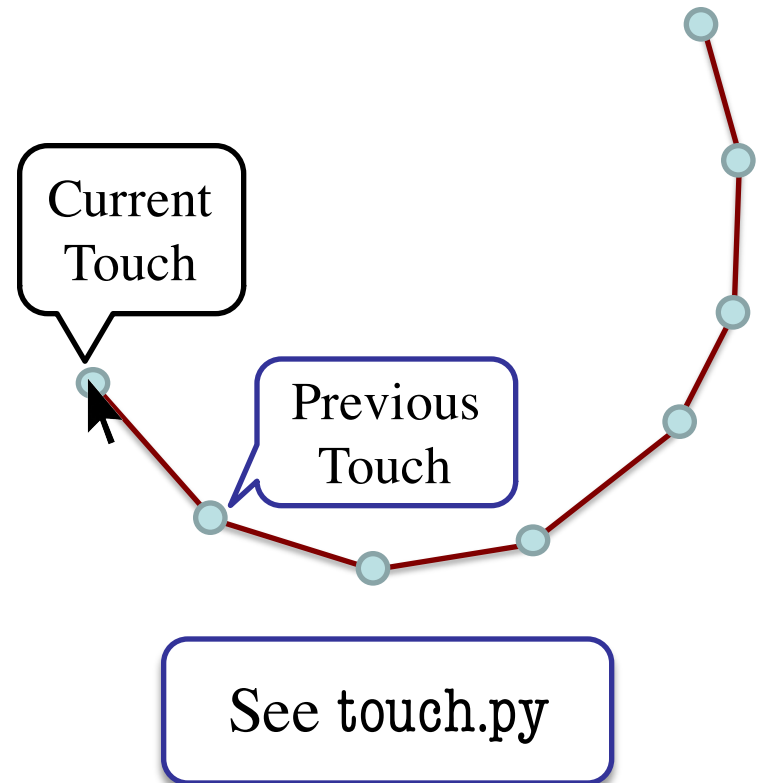


Input and Invariants

- 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**

Relationship between two variables.

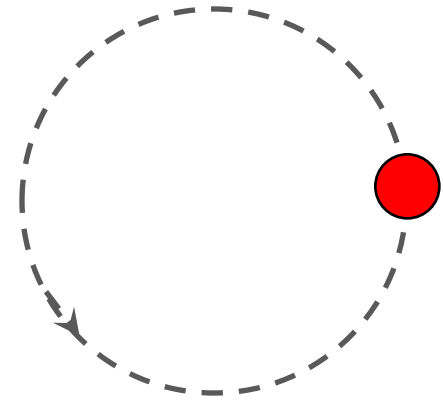
Line segment = 2 points



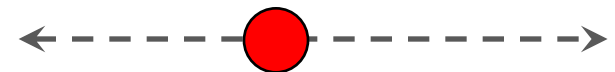
State: Changing What the Loop Does

- **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**



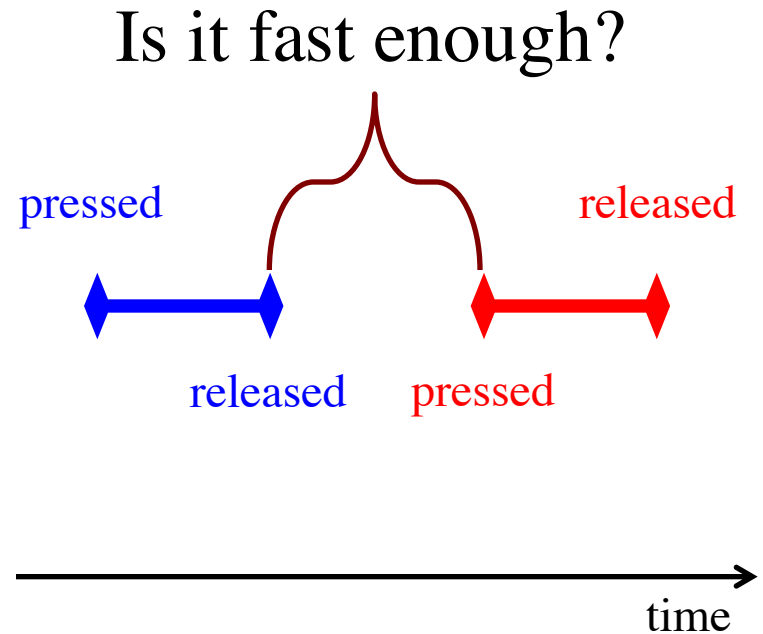
See `state.py`

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

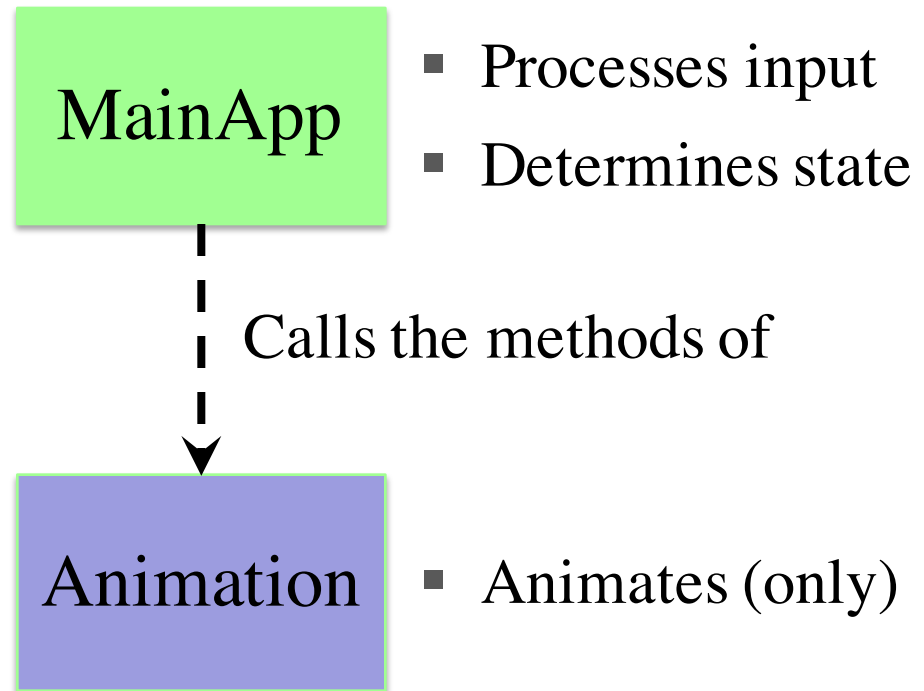
- Double click = 2 fast clicks
- Count number of fast clicks
 - Add an attribute `clicks`
 - Reset to 0 if not fast enough
- Time click speed
 - Add an attribute `time`
 - Set to 0 when mouse released
 - Increment when not pressed (e.g. in loop method `update()`)
 - Check time when next pressed



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



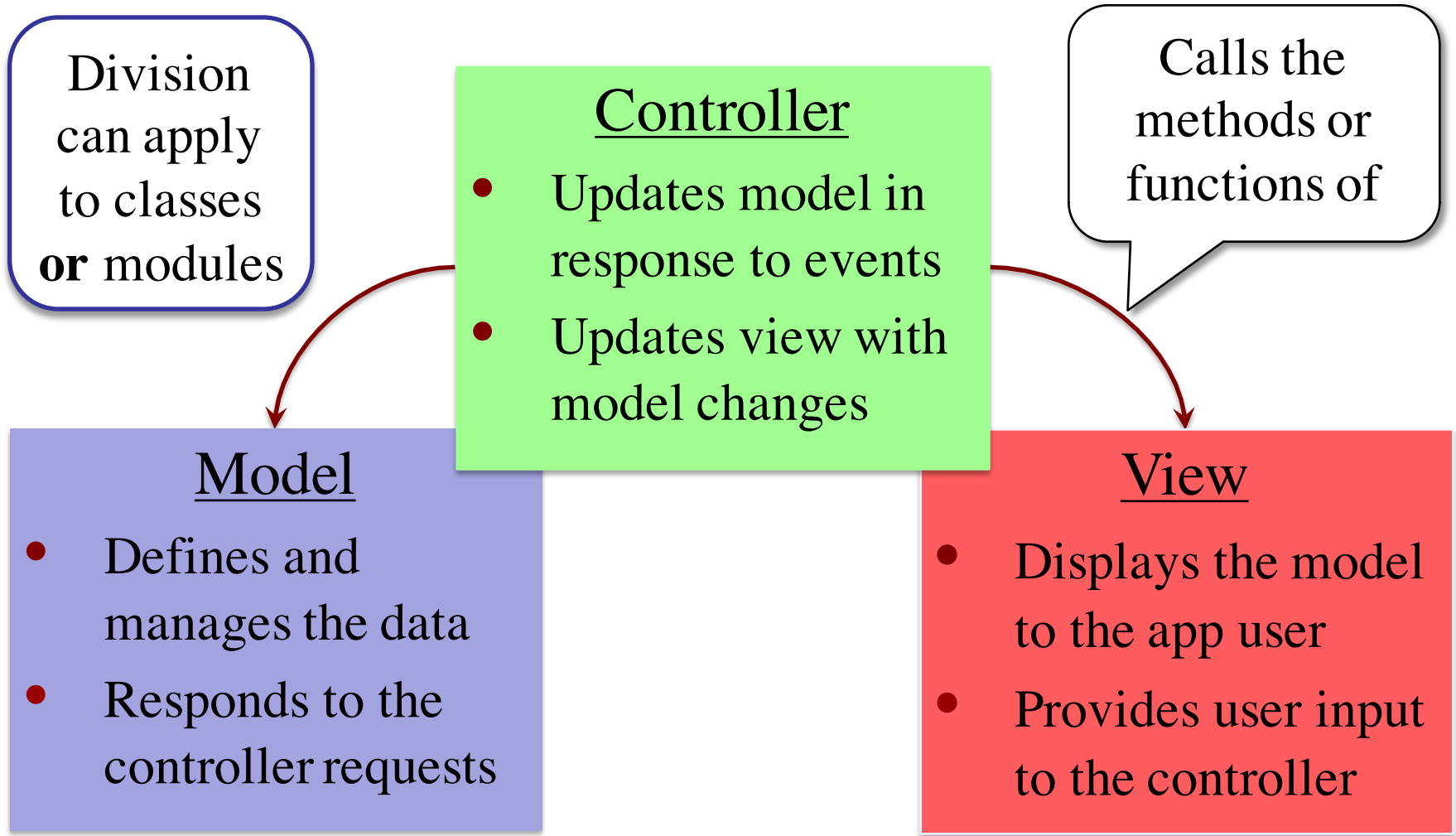
See subcontroller.py

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

Just like
this course!

Model-View-Controller Pattern



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

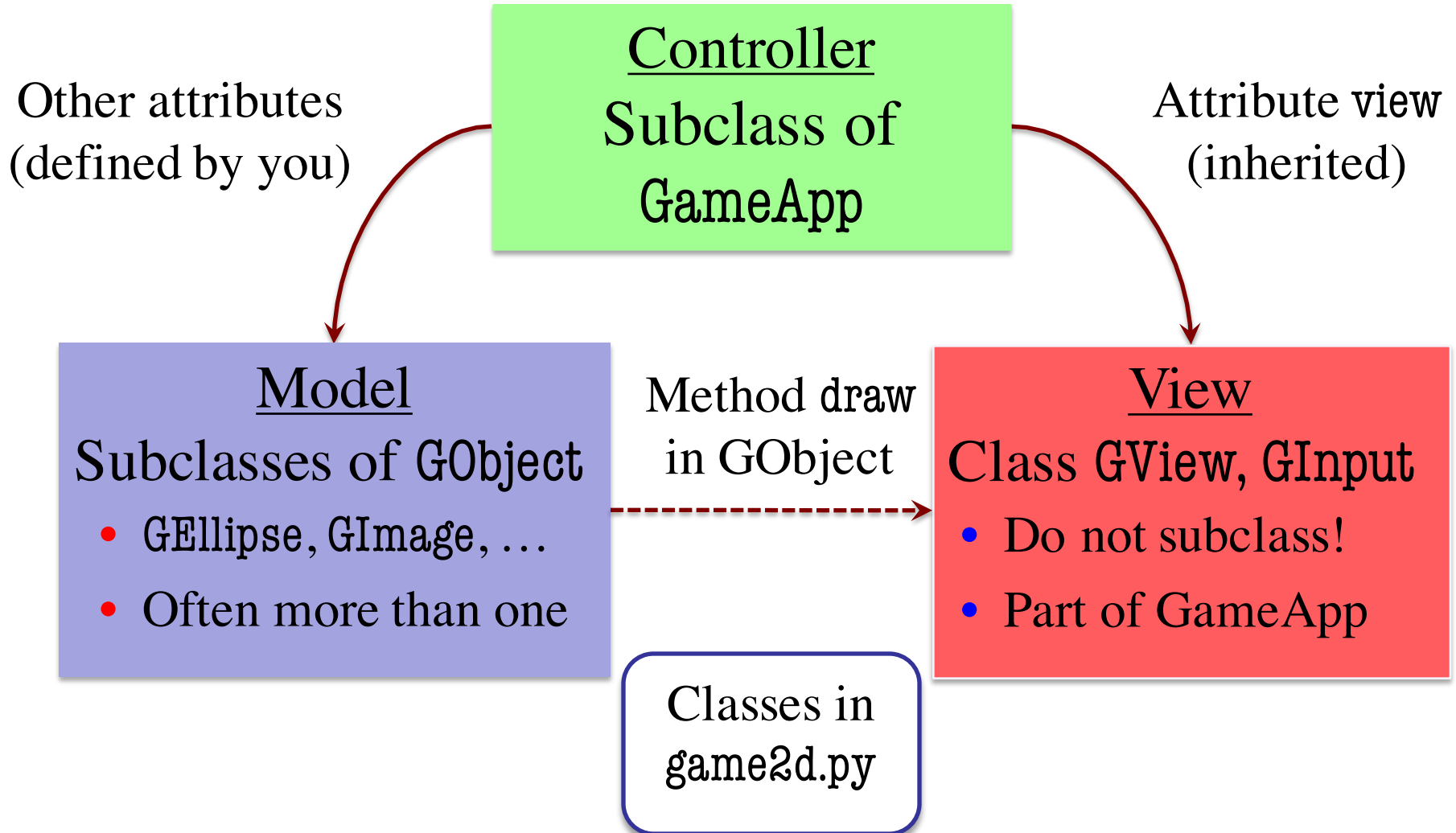
Model

- **A3**: Color classes
 - RGB, CMYK & HSV
- **A4**: Turtle, Pen
 - Window is **View**
- **A5**: Why **classes** sometimes and **functions** others?
- **A7**: Ball, Brick, etc..
 - All shapes/geometry

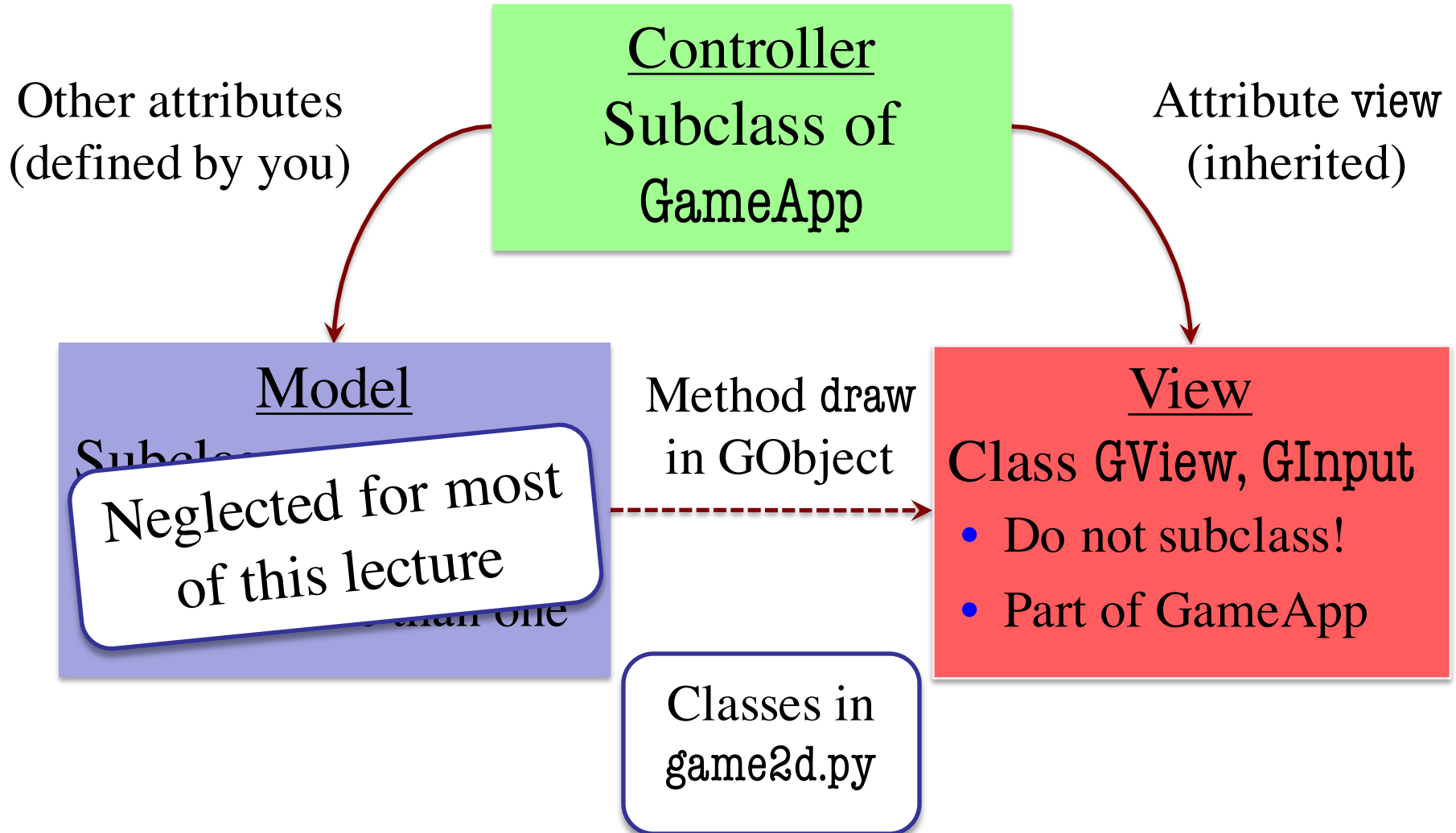
Controller

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

Model-View-Controller in CS 1110

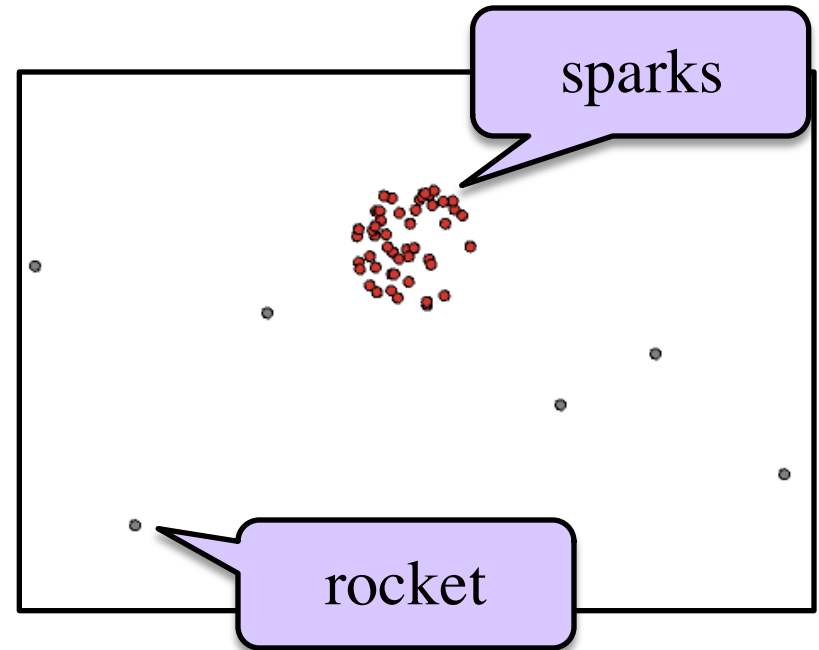


Model-View-Controller in CS 1110



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



See pyro.py