

Lecture 23

GUI Applications

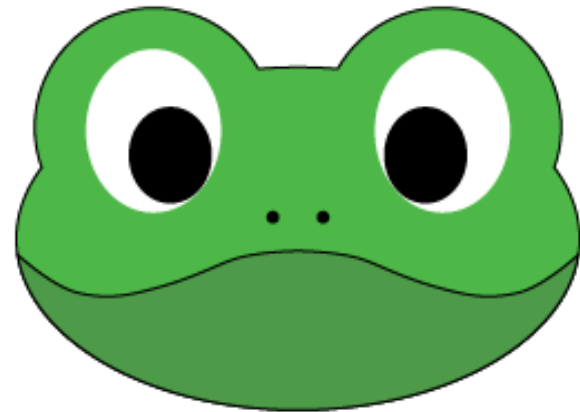
Announcements for This Lecture

Assignments

- A5 currently being graded
 - Will have results this Sat
- A6 is due **TOMORROW**
 - Worth 8% of your grade
 - Remember to fill in Survey
- A7 posted **TOMORROW**
 - Based on today's lecture!
 - Due **December 8th** (last day)
 - Minor extensions possible

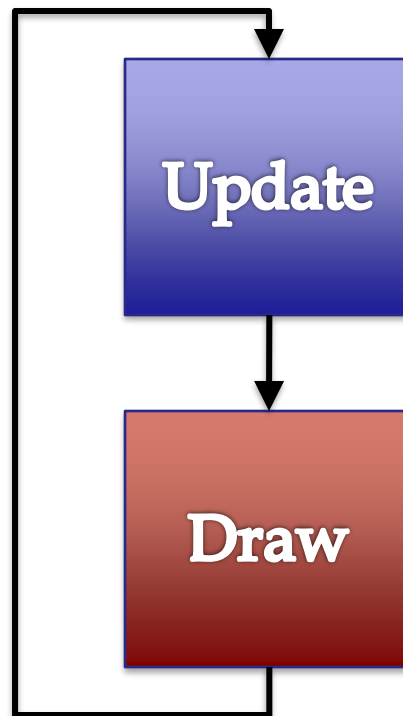
Video Lessons

- **Lesson 24** for **today**
- **Lessons 26, 27** for Tues
- Last material on 2nd exam



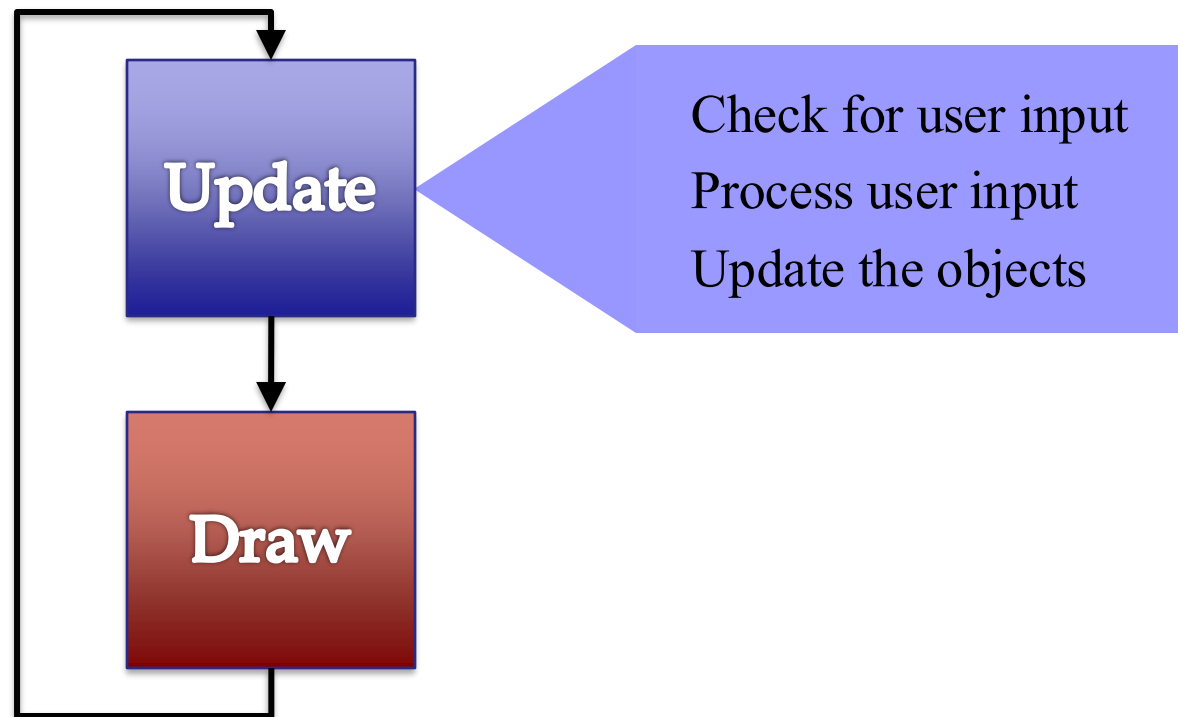
A Standard GUI Application

Animates the
application,
like a movie



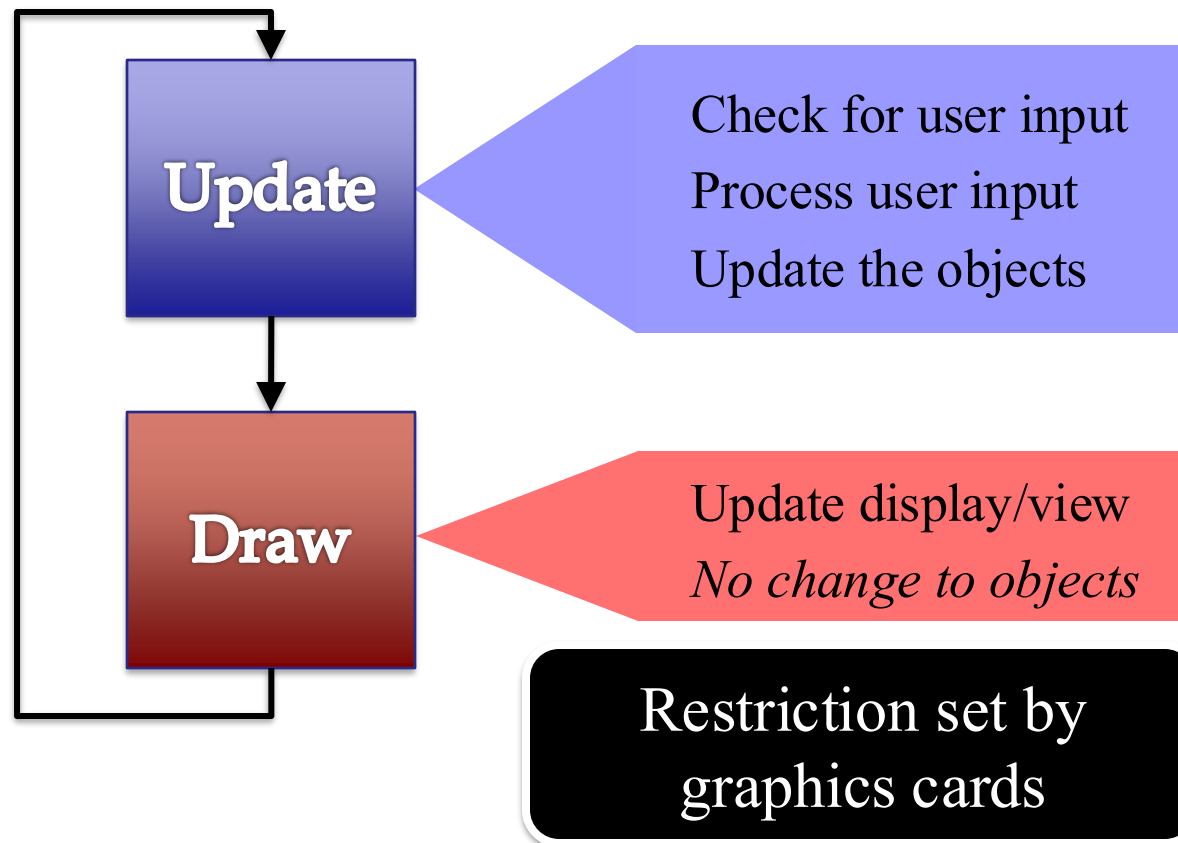
A Standard GUI Application

Animates the
application,
like a movie

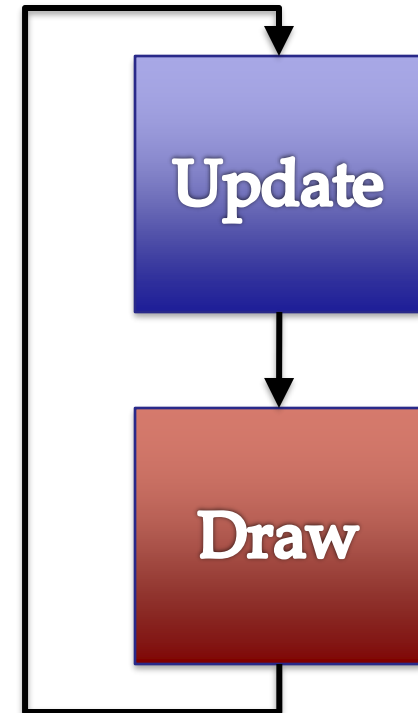
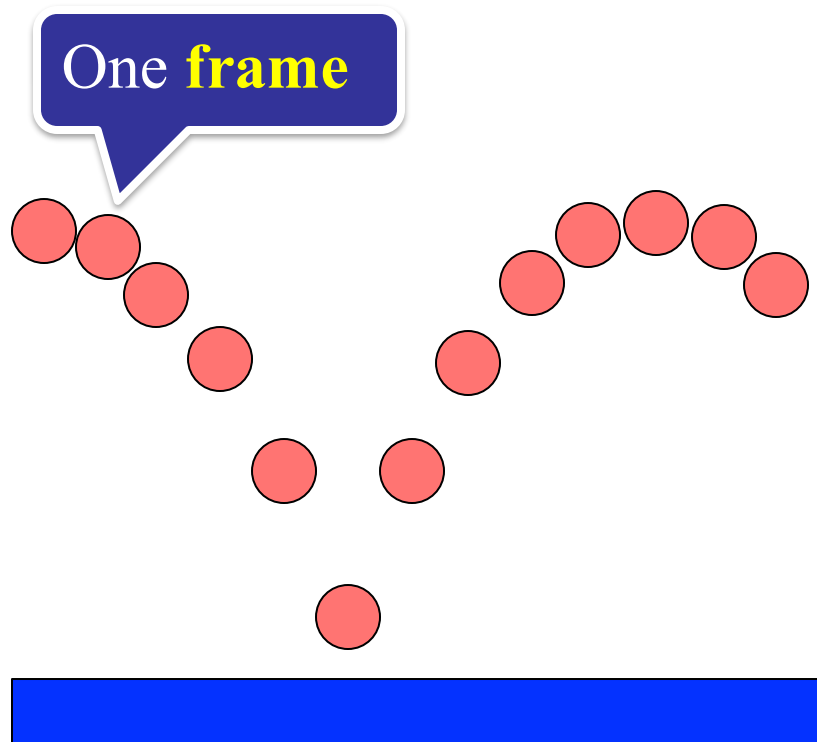


A Standard GUI Application

Animates the
application,
like a movie



The Animation Frame



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  
    # Handled by OS/GUI libraries
```

Would like to
“plug in” code

```
# Your code goes h
```

Why do we need to
write this each time?

```
# 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
    application.update_screen()
    # Handled by OS/GUI libraries
```

Method call
(for loop body)

- Put loop **BODY** in an app class.
- OS/GUI handles everything else.

Custom Application class
with its own **attributes**

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

This creates
a **call frame**

All its variables are
erased when done

Attributes = Loop Variables

Normal Loops

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

Variables “external”
to the loop body

Application

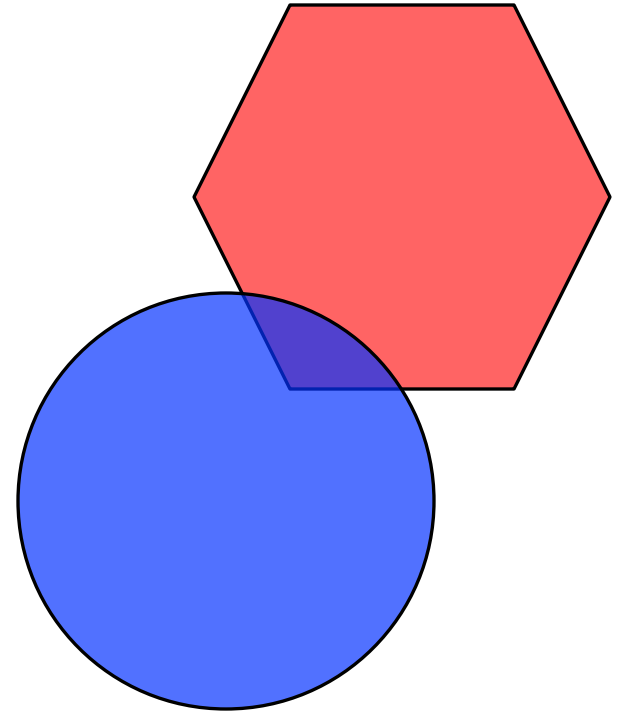
Attributes are the
“external” variables

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

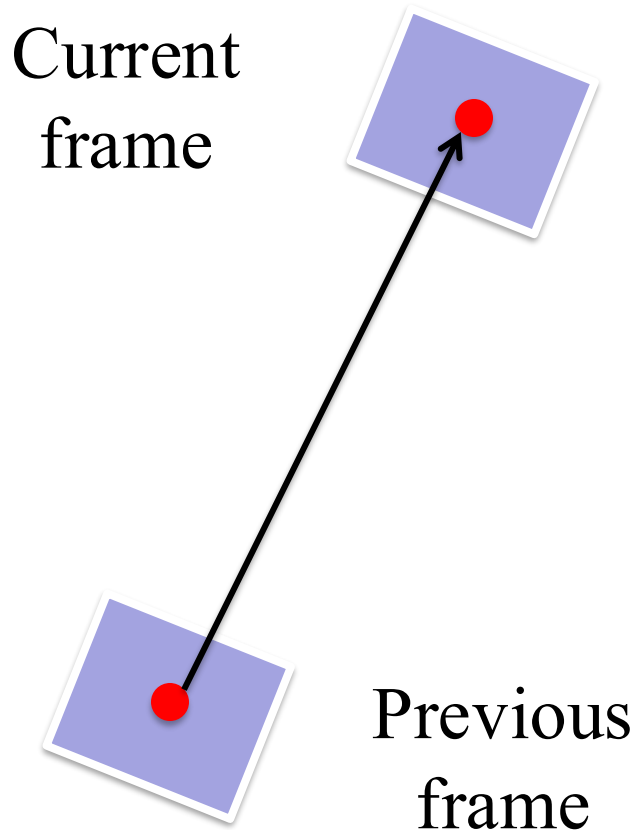
Programming Animation

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 *last* frame
- **Example:** Movement
 - Need old position/velocity
 - Compute next position
- Requires **attributes**
 - Attributes never deleted
 - Remain after `update()` ends

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

Inter-Frame

- 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

Variables and the Loop

```
while program_is_running:
    # Get information from
mouse/keyboard
    # Handled by OS/GUI libraries

    # Your code
application.

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

Local variables erased.
But **attributes** persist.

The Actual Game Loop

```
# Constructor
```

```
game = GameApp()
```

Too *early* to initialize
everything

```
...
```

```
game.start() #Loop init
```

Actual loop
initialization

```
while program_running:
```

```
    # Get input
```

```
    # Your code goes here
```

```
    game.update(time_
```

```
    game.draw())
```

Separate `update()`
and `draw()`
methods

Designing a Game Class: Animation

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

"""App to animate an ellipse
circle."""

See
animation.py

```
    def start(self):
        """Initializes the game loop."""
        ...
```

```
    def update(self, dt):
        """Changes the ellipse position."""
        ...
```

```
    def draw(self):
        """Draws the ellipse"""
        ...
```

Designing a Game Class: Animation

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

Designing a Game Class: Animation

```
class Animation(game2d.GameApp):  
    """App to animate a circle."""  
  
    def start(self):  
        """Initializes the loop"""  
        ...  
  
    def update(self, dt):  
        """Changes the position."""  
        ...  
  
    def draw(self):  
        """Draws the circle"""  
        ...
```

See animation.py

Parent class that does hard stuff

Loop initialization
Do NOT use init

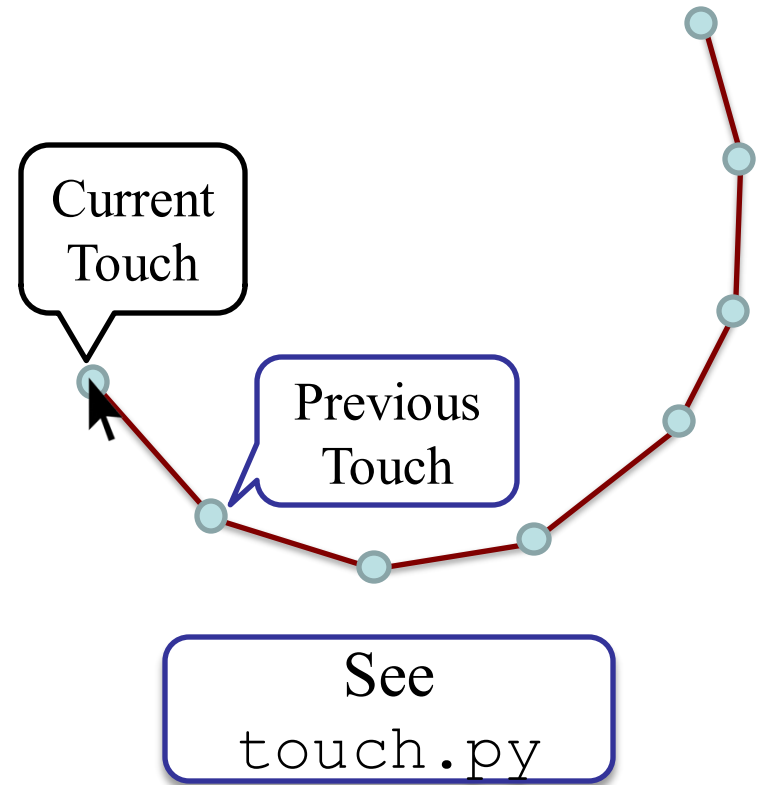
Loop body

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`

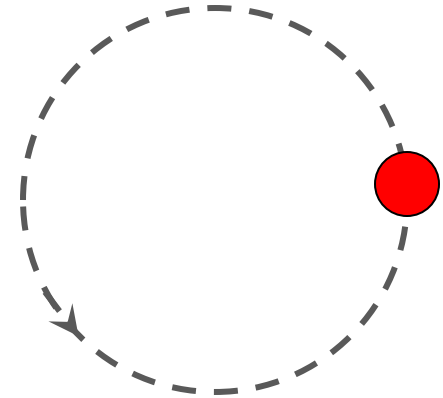
Line segment = 2 points



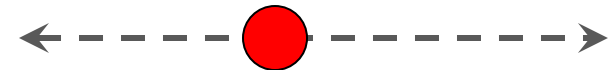
State: Changing What the Loop Does

- **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`



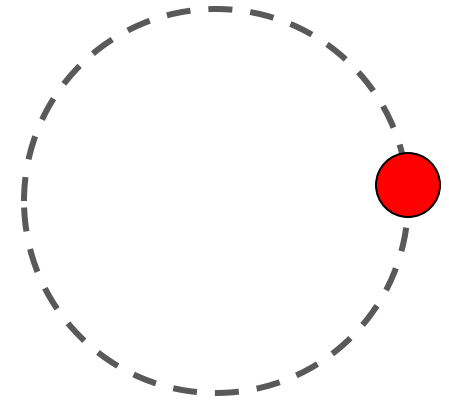
State `ANIMATE_HORIZONTAL`



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)



See
`state.py`

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 no key pressed
- `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

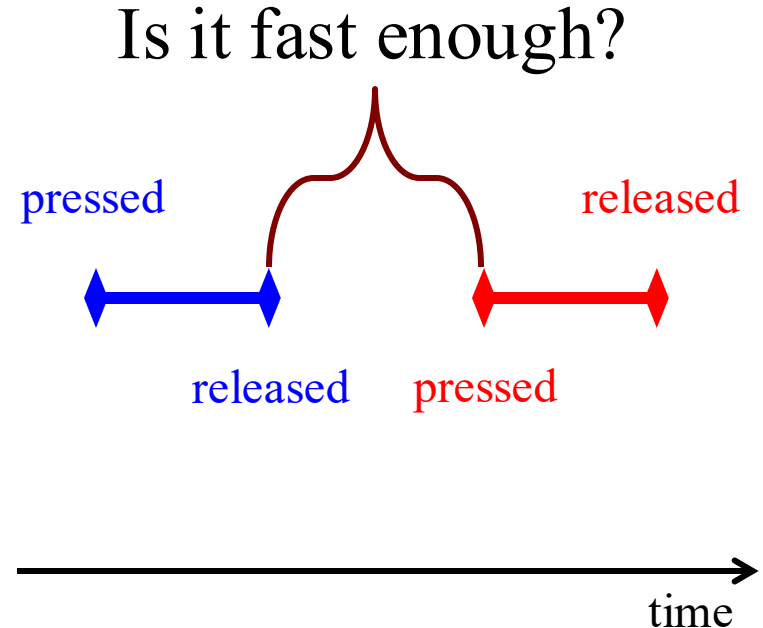
Mouse/Touch

- `touch`
 - **Attribute** giving a position
 - Stored as a Point2 object
 - None if no touch
- `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

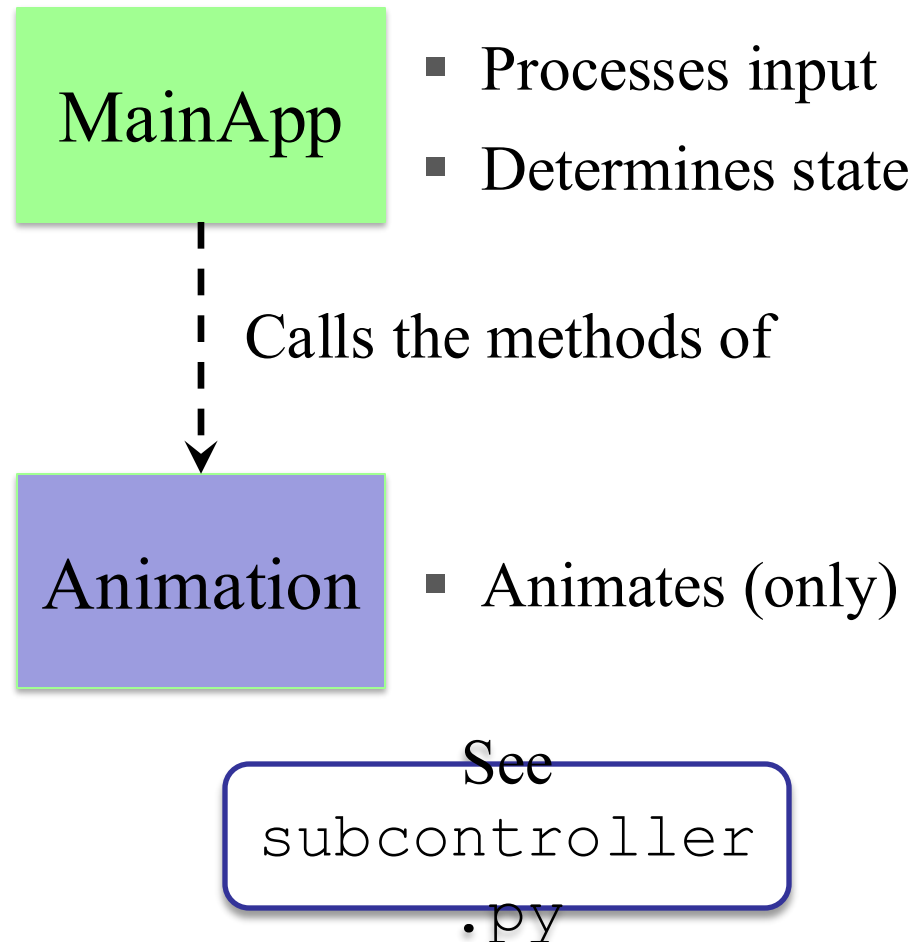
- Double click = 2 fast clicks
- Count number of fast clicks
 - Add an attribute `clicks`
 - Reset to 0 if not fast enough
- Time click speed
 - Add an attribute `time`
 - Set to 0 when mouse released
 - Increment when not pressed (e.g. in loop method `update()`)
 - Check time when next pressed



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

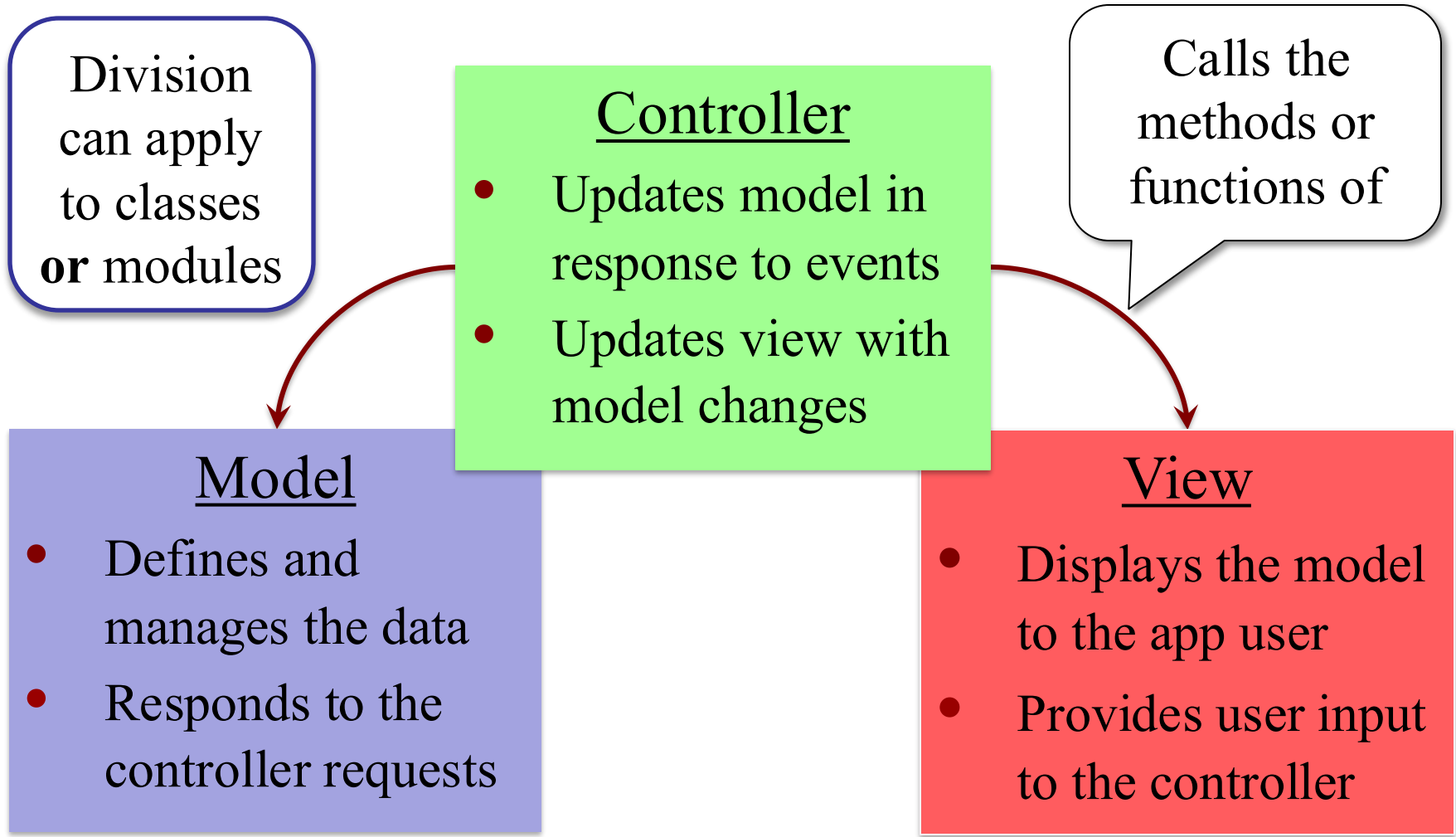


How to Break Up: Software Patterns

- **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**: Pixels, Image
 - Data is always in model
- **A7**: Frog, Log, etc..
 - All shapes/geometry

Controller

- **A3**: a3app.py
 - Hidden classes
- **A4**: Functions in a4.py
 - No need for classes
- **A6**: Filter
 - Processes the data/image
- **A7**: Froggit, Level
 - Main part of assignment!

MVC in this Course

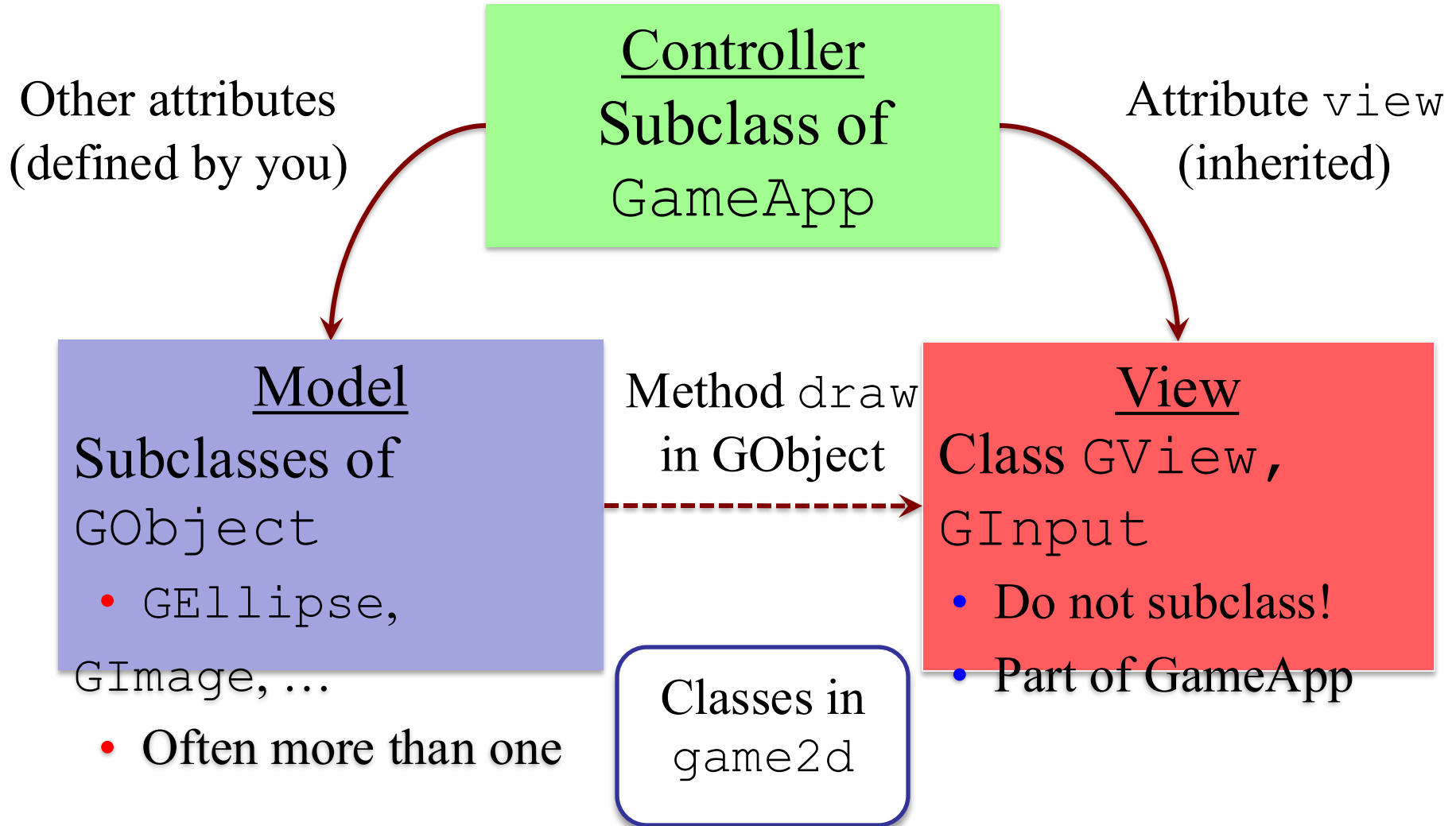
Model

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

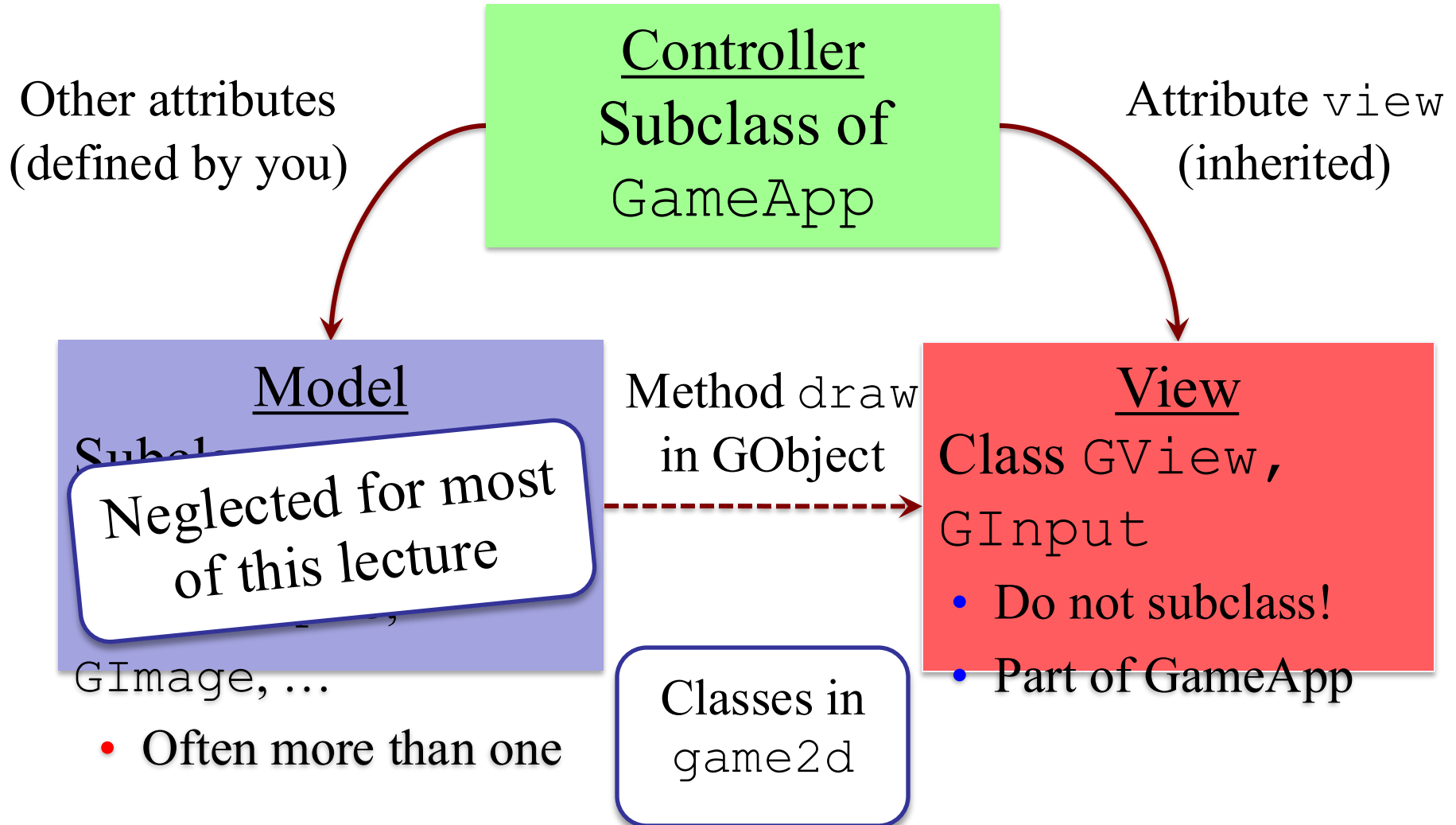
Controller

- **A3**: `a3app.py`
 - Hidden classes
- **A4**: Functions in `a4.py`
 - No need for classes
- **A6**: `Filter`
 - Processes the data/image
- **A7**: `Froggit`, `Level`
 - Main part of assignment!

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
- 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

