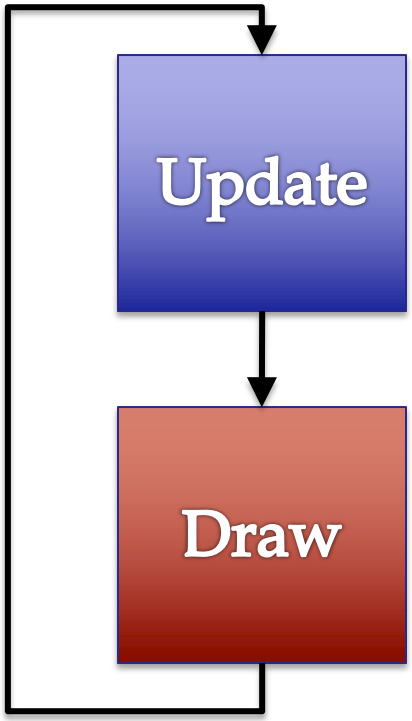
Module 27

GUI Applications

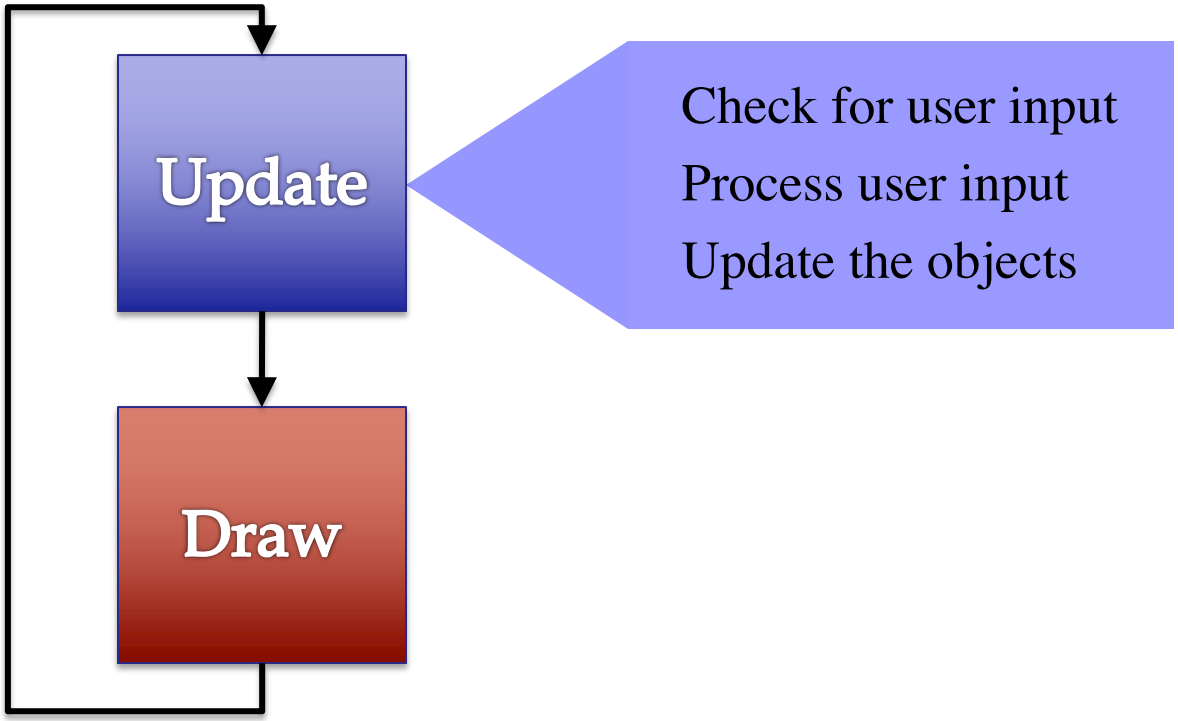
A Standard GUI Application

Animates the application, like a movie



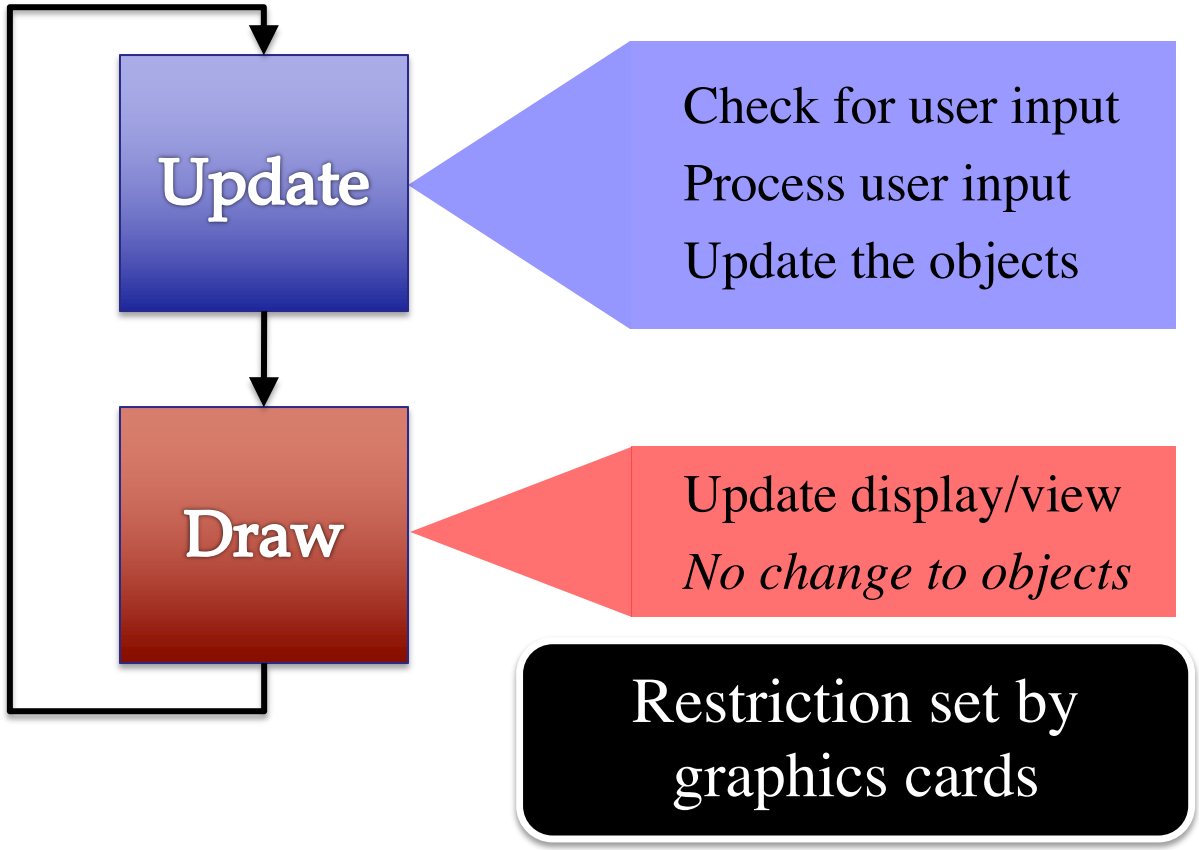
A Standard GUI Application

Animates the application, like a movie



A Standard GUI Application

Animates the application, like a movie



Basic Application Loop

```
while program_is_running:
    # Get user input



    # Custom Application Code



    # Draw stuff on the screen
```

Do We Need to Write All This?

```
while program_is_running:
    # Get user input
    # Custom Application Code
    # Draw stuff on the screen
```

Can we get this handled for us?

Code you **must** write yourself.

Can we get this handled for us?

Idea: Use a Class/Object

```
application = AppClass()
while application.isRunning():
    application.getInput()

    application.update()

    application.drawToScreen()
```

Leverage Subclassing

```
application = AppClass()
while application.isRunning():
    application.getInput()

    application.update()

    application.drawToScreen()
```

Overridden

Inherited

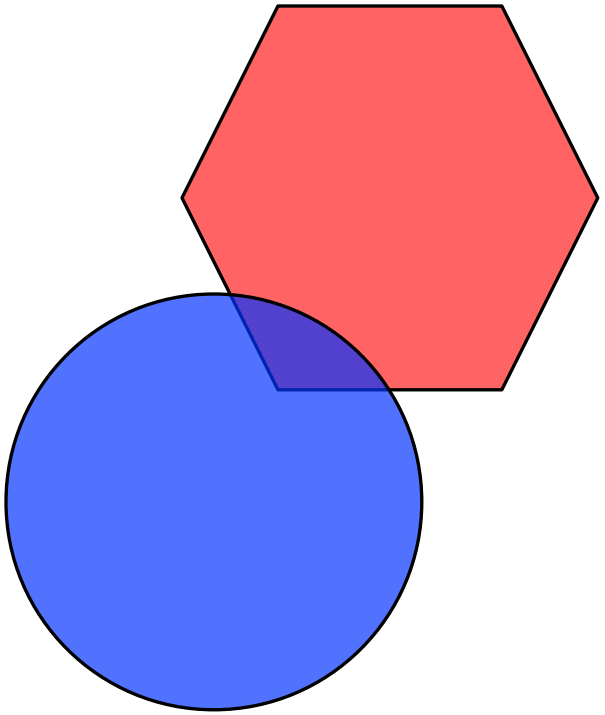
Overridden

Inherited

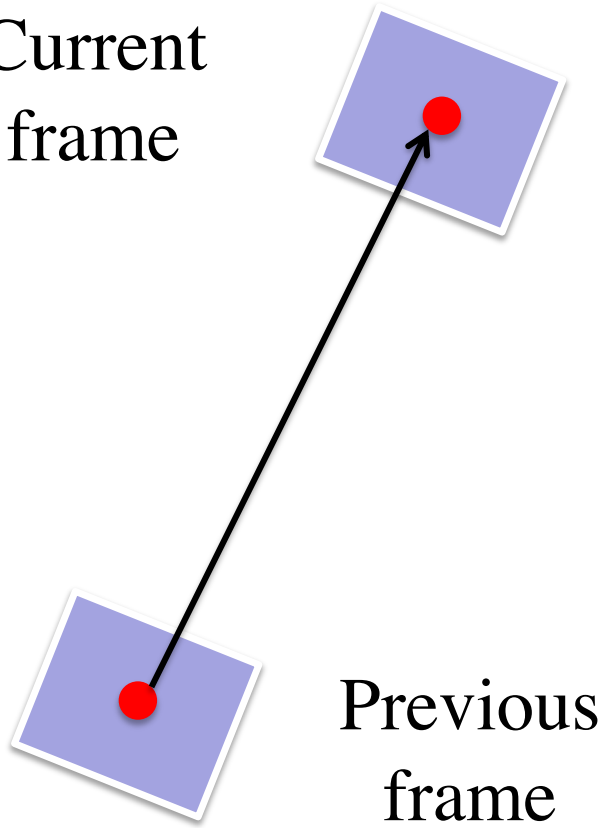
Programming Animation

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

Idea: Use a Class/Object

```
application = AppClass()
while application.isRunning():
    application.getInput()

    application.update()

    application.drawToScreen()
```

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

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

Attributes = Loop Variables

Normal Loops

Variables "external" to the loop body

```
x = 0
i = 2

while i <= 5:
    x = x + i*i
    i = i +1
```

Application

Attributes are the "external" variables

```
while app.isRunning():
    app.getInput()
    # Your code called here
    application.update()
    app.drawToScreen()
```

The Actual Game Loop

```
# Constructor
game = GameApp(...)
...
game.start() #Loop initialization
while game.isRunning():
    # Get input
    # Your code goes here
    game.update(time_elapsed)
    game.draw()
```

To *early* to initialize everything

Actual loop initialization

Inherited

Separate update() and draw() methods

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

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"""
        ...
```

Parent class that does hard stuff

See animation.py

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"""
        ...
```

Parent class that does hard stuff

See animation.py

Loop initialization
Do NOT use __init__

Loop body

Use method draw()
defined in GObject

Drawing to The Screen

- All `GameApp` objects have a `view` attribute
  - Instance of `GView` (similar to Turtle `Window`)
  - Represents the window to draw to
- Create objects to draw
  - Subclasses of `GObject`
  - Special cases, `GLabel`, `GImage`, `GSprite`
  - All inherit a method `draw(view)`
- Just like our lessons on subclasses!

The GInput Class

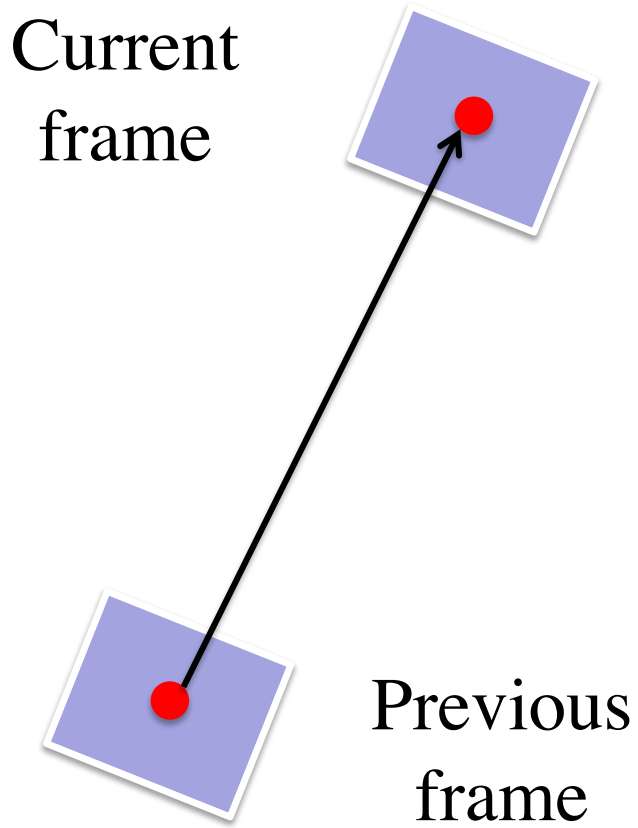
- All `GameApp` objects have an `input` attribute
  - Contains input for current animation frame
  - Support for Keyboard and Mouse (Touch)
- Class `GInput` defines attributes, methods
  - `is_key_down(key)`: Returns True if key held
  - `is_touch_down()`: Returns True if mouse pressed
  - `keys`: List of all keys currently pressed
  - `touch`: `Point2` of (pressed) mouse screen location

The GInput Class

- All GameApp objects have an input attribute
  - Contains input for current animation frame
  - Support for Keyboard and Mouse (Touch)
- Class GInput defines attributes, methods
  - is_key_down(key): Re
  - is_touch_down(): Re sed
  - keys: List of all keys
  - touch: Point2 of (pressed) mouse screen location

Simple Example:
Pausing animation

Recall: Programming Animation

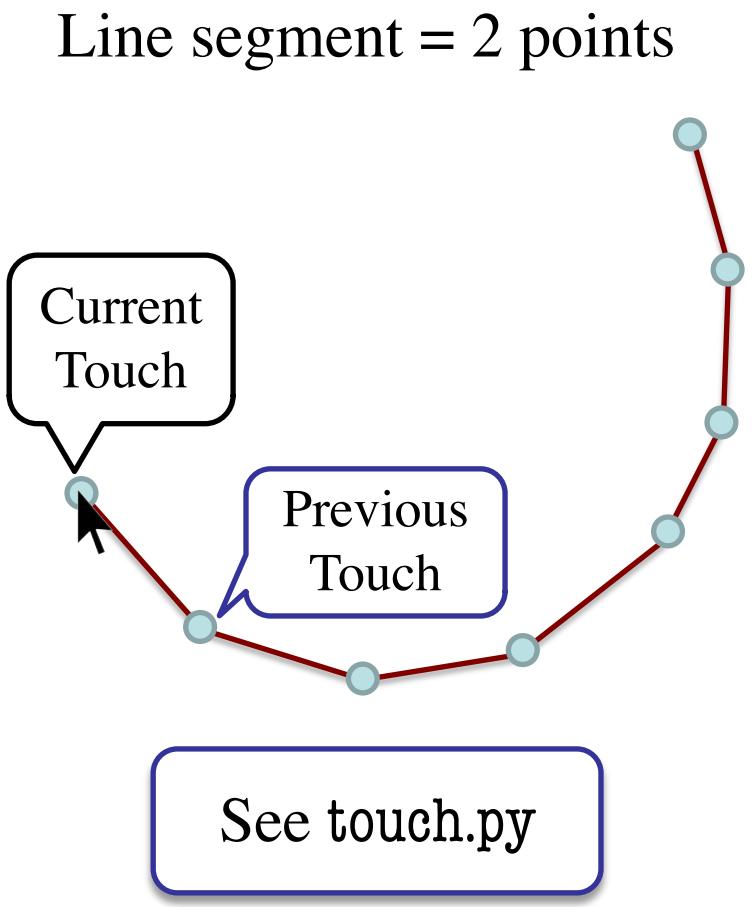


Inter-Frame

- 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

Inter-Frame Comparisons

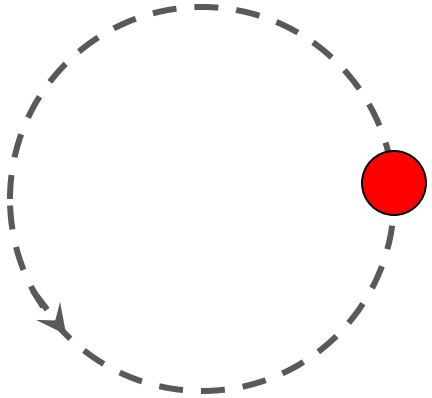
- Attribute touch in GInput
  - The mouse press position
  - Or **None** if not pressed
  - Access with self.input.touch
- Compare touch, last position
  - Mouse button **pressed:**
    last None, touch not None
  - Mouse button **released:**
    last not None, touch None
  - Mouse **dragged**:
    last and touch not None



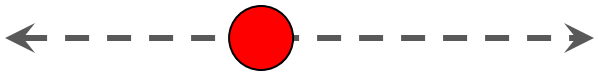
State: Changing What the Loop Does

- **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

Designing States

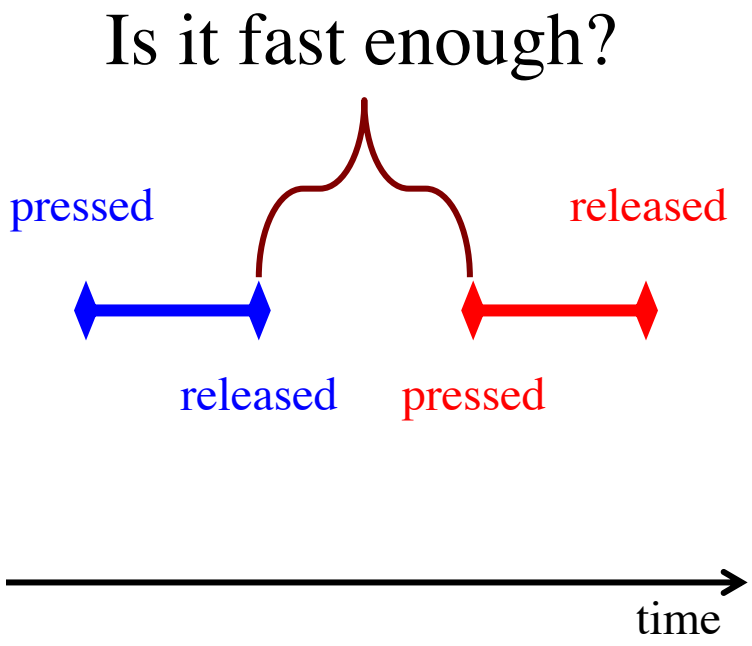
- Each state has its ***own set*** of invariants.
  - **Drawing?** Then `touch` and `last` are not None
  - **Erasing?** Then `touch` is None, but `last` is not
  - **Erasing?** Then `touch` and `last` are both None
- Need to make clear in class specification
  - What are the **application states**?
  - What are the invariants ***for each state***?
  - What are the rules to switch to a new state?

State Triggers

- Need a rule for switching between states
  - Look for some event to happen, and change state
  - **Example:** press space to change state in `state.py`
  - **Example:** double clicking to erase in `touch.py`
- Complex apps also limit state transitions
  - `ANIMATE_CIRCLE` => `ANIMATE_HORIZONTAL` **OK!**
  - `ANIMATE_HORIZONTAL` => `ANIMATE_CIRCLE` **BAD!**
- Again, make clear in specification

Example: Checking Click Types

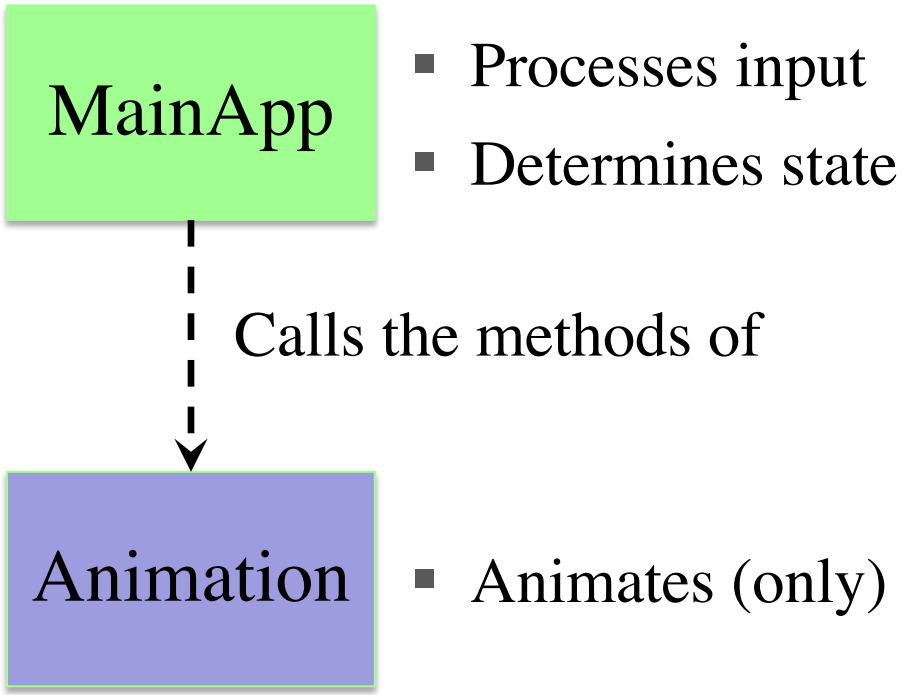
- Double click = 2 fast clicks
- Count number of fast clicks
  - Add an attribute `clicks`
  - Reset to 0 if not fast enough
- Time click speed
  - Add an attribute `time`
  - Set to 0 when mouse released
  - Increment when not pressed (e.g. in loop method `update()`)
  - Check `time` when next pressed



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



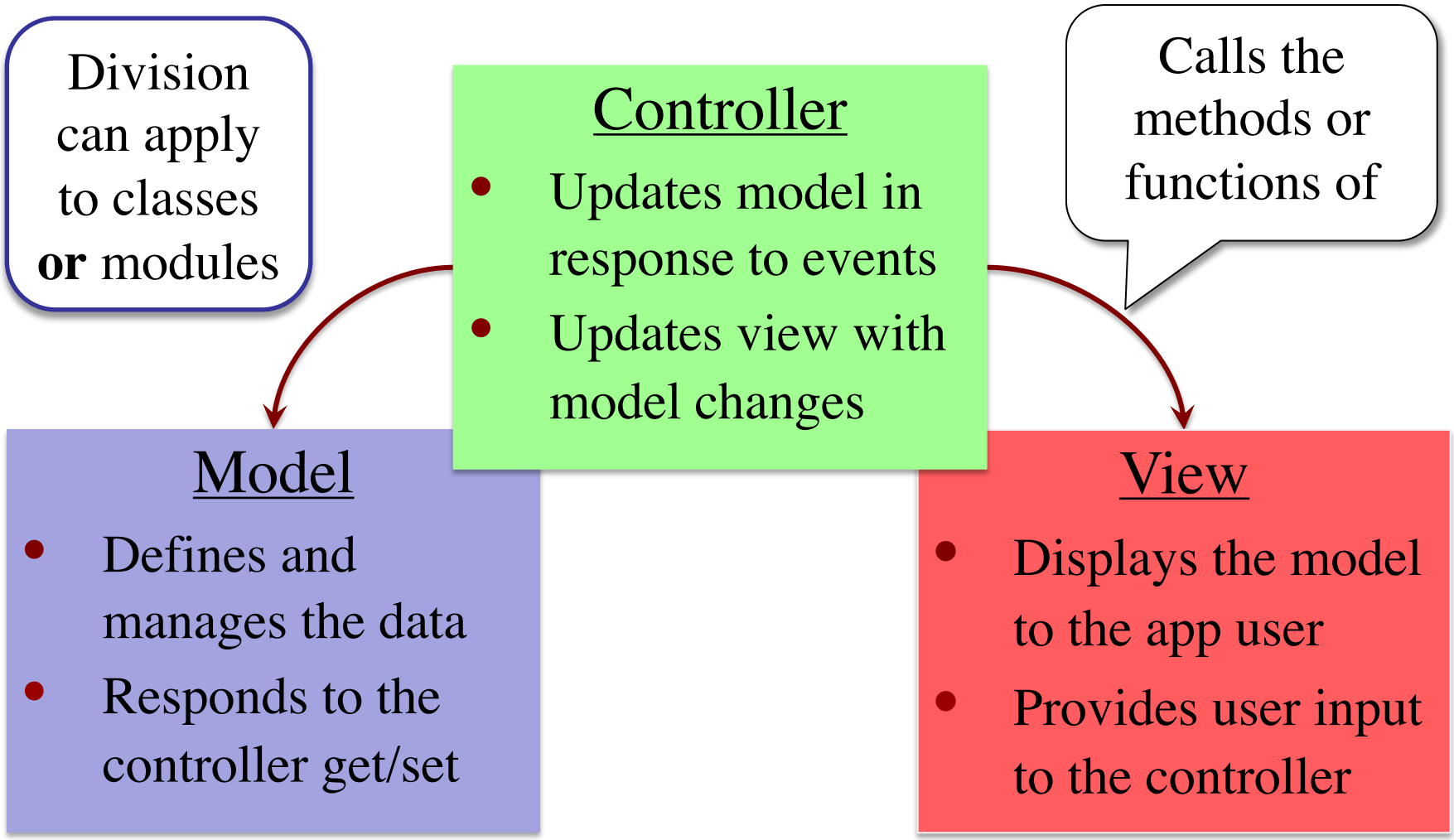
See subcontroller.py

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**
- **A7**: `Frog`, `Car`, etc..
  - All shapes/geometry

Controller

- **A3**: `a3app.py`
  - Hidden classes
- **A4**: Funcs in `a4.py`
  - No need for classes
- **A7**: `Froggit`, `Level`
  - The actual assignment!

MVC in this Course

Model

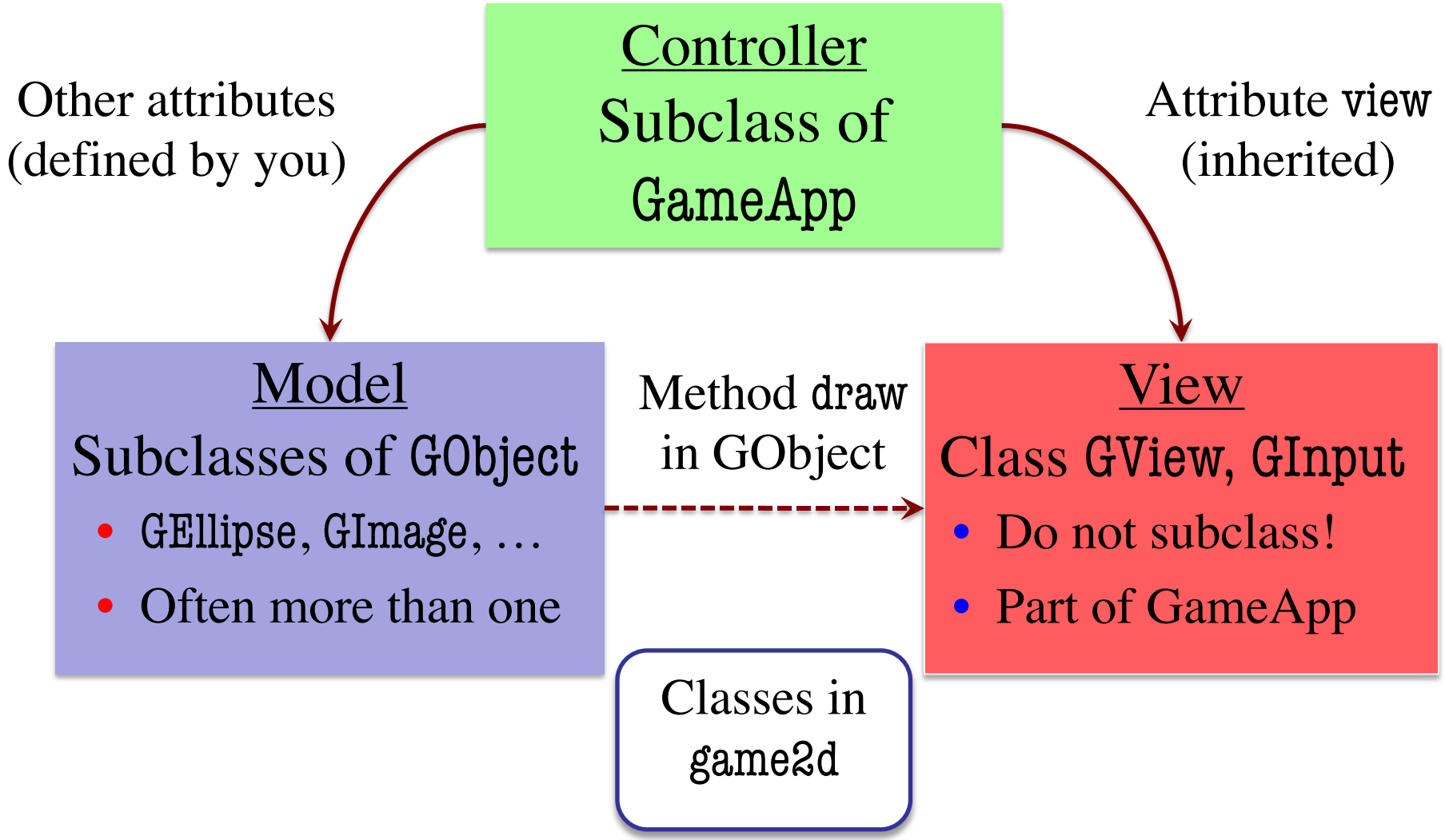
- **A3**: Color classes
  - RGB, CMYK & HSV
- **A4**: Turtle, Pen
-
  - All shapes/geometry

Why **classes** sometimes and **functions** others?

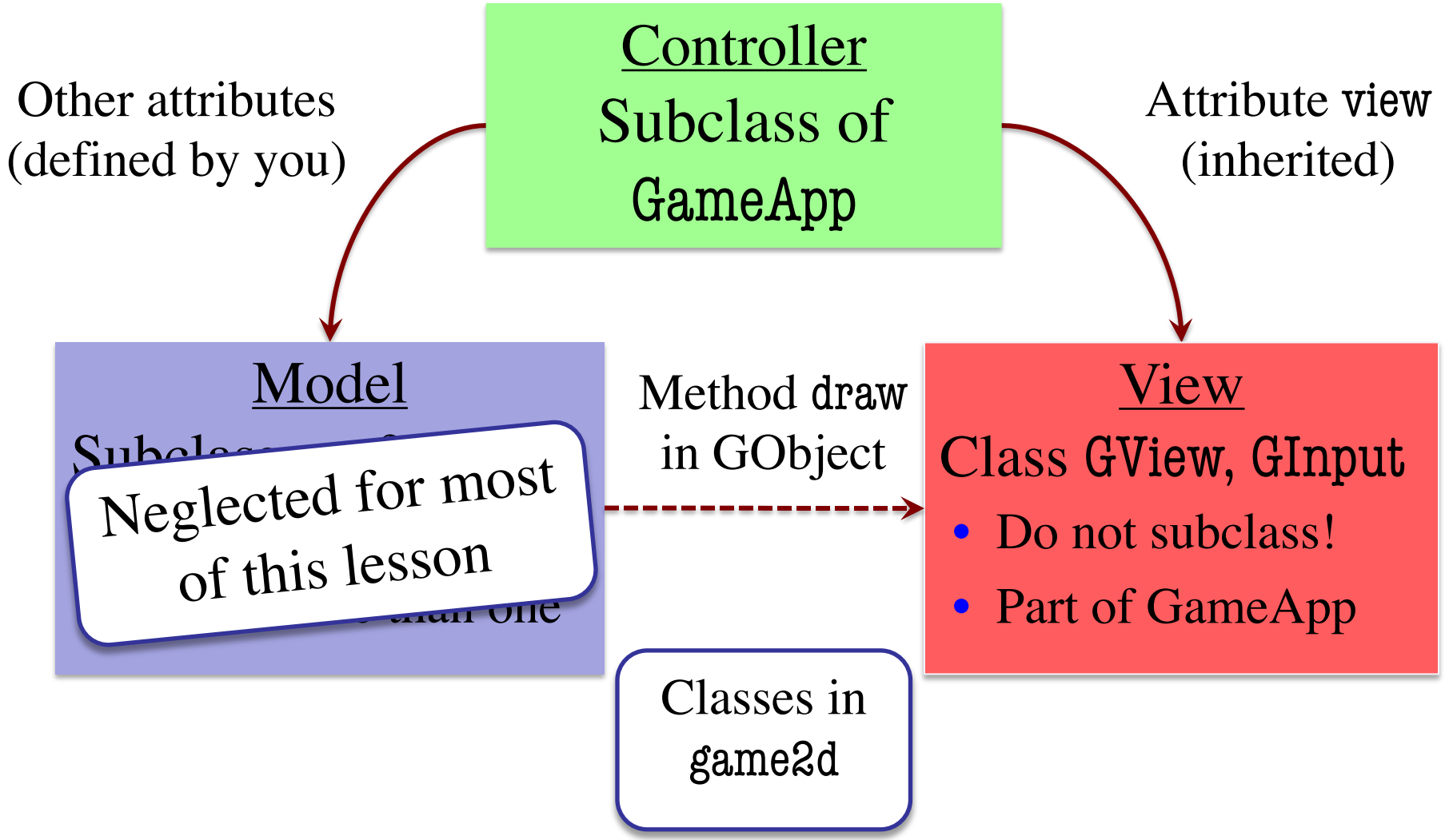
Controller

- **A3**: a3app.py
  - Hidden classes
- **A4**: Funcs in a4.py
  - No need for classes
- **A7**: Froggit, Level
  - The actual assignment!

Model-View-Controller in CS 1110

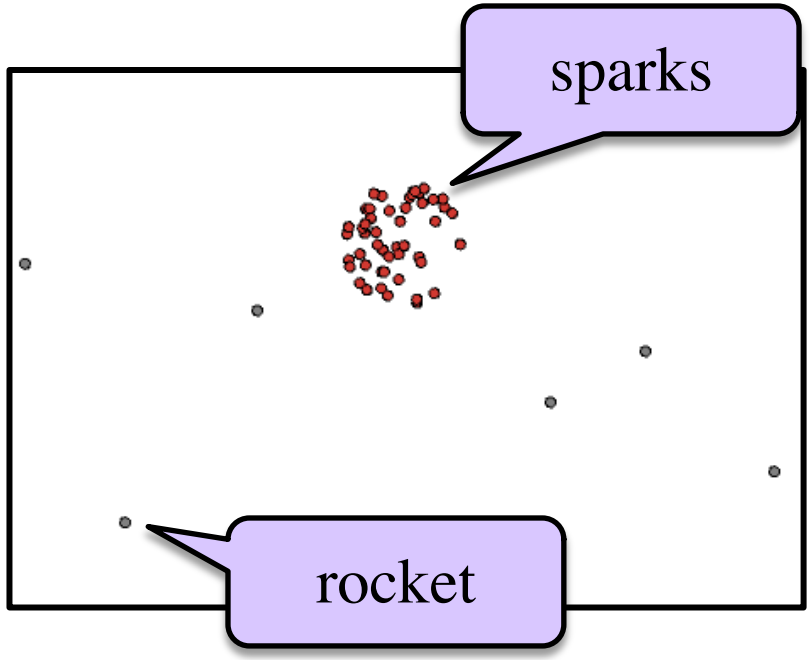


Model-View-Controller in CS 1110



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
  - `Frog` to animate the player
  - `Car` to represent a vehicle



See `pyro.py`