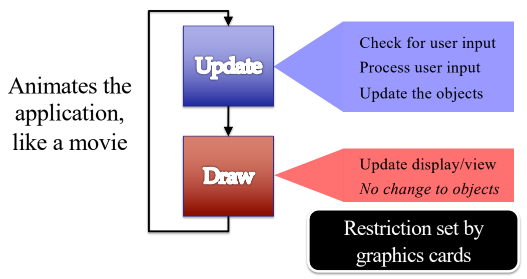
A Standard GUI Application



1

Must We Write this Loop Each Time?

```
while program_is_running:
    # Get information from mouse/keyboard
    # Handled b

    # Your code
    application.update()

    #
    #                                  es
```

Method call (for loop body)

Custom Application class with its own **attributes**

- Write loop body in an app class.
- OS/GUI handles everything else.

2

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

3

Designing a Game Class: Animation

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

    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

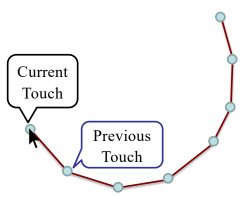
Use method draw() defined in GObject

4

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



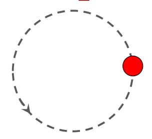
See touch.py

5

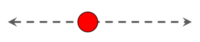
State: Changing What the Loop Does

- **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

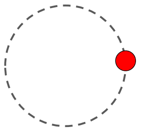


See state.py

6

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

7

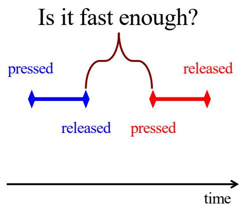
Checking Input

Keyboard	Mouse/Touch
• `is_key_down`(`key`)	• `touch`
▪ Returns True if key is down	▪ **Attribute** giving a position
▪ `key` is a string (`'a'` or `'space'`)	▪ Stored as a Point2 object
▪ Empty string means *any* key	▪ But None if no touch
• `is_key_pressed`(`key`)	• `is_touch_pressed`()
▪ Returns True if `key` pressed	▪ True if `touch` pressed
▪ `key` **not** down prev. frame	▪ `touch` was None prev. frame
• `is_key_released`(`key`)	• `is_touch_released`()
▪ Returns True if `key` released	▪ True if `touch` released
▪ `key` was down prev. frame	▪ `touch` **not** None prev. frame

8

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

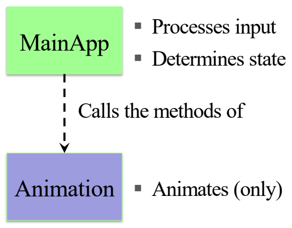


See touch.py

9

Designing Complex Applications

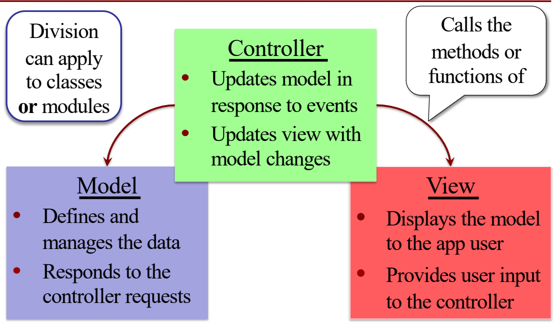
- Applications can become extremely complex
  - Large classes doing a lot
  - Many states & invariants
  - Specification unreadable
- **Idea**: Break application up into several classes
  - Start with a "main" class
  - Other classes have roles
  - Main class delegates work



See subcontroller.py

10

Model-View-Controller Pattern


Division can apply to classes or modules
Calls the methods or functions of

Controller
- Updates model in response to events
- Updates view with model changes

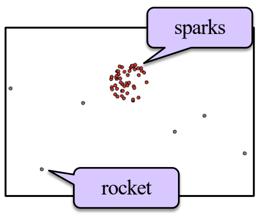
Model
- Defines and manages the data
- Responds to the controller requests

View
- Displays the model to the app user
- Provides user input to the controller

11

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
  - `Ship` to animate the player
  - `Alien` to represent an alien



See pyro.py

12