A Standard GUI Application

- Update display/view
- No change to objects
- Restriction set by graphics cards

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
    application.update()
    # Draw stuff on the screen
    # Handled by OS/GUI libraries
```

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

Designing a Game Class: Animation

```python
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"""
        ...
```

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!
  - Forget if we do not store it
  - Purpose of attribute `last`

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 `animation.py`
See `state.py`
See `touch.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)

**Checking Input**

**Keyboard**
- `is_key_down(key)`
  - Returns True if key is down
  - key is a string ('x' 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

**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

**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

**Model-View-Controller Pattern**

- Division can apply to classes or modules

**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 the model to the app user
- Provides user input to the controller

**Calls the methods or functions of**

**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