Game Loops + pygame

Part II

Reference: **www.pygame.org**

The Game Loop

- Animation / Game loop
  1. [Get input from the user] (GameLoop only)
     a. Usually update variables
  2. Update non-input-related variables
  3. Draw (using variables)
     a. Erase the screen
     b. Actual drawing code (using variables)
     c. Flip
  4. Repeat [Brain absorbs info while new screen is being drawn in steps 1-3b]

Window input

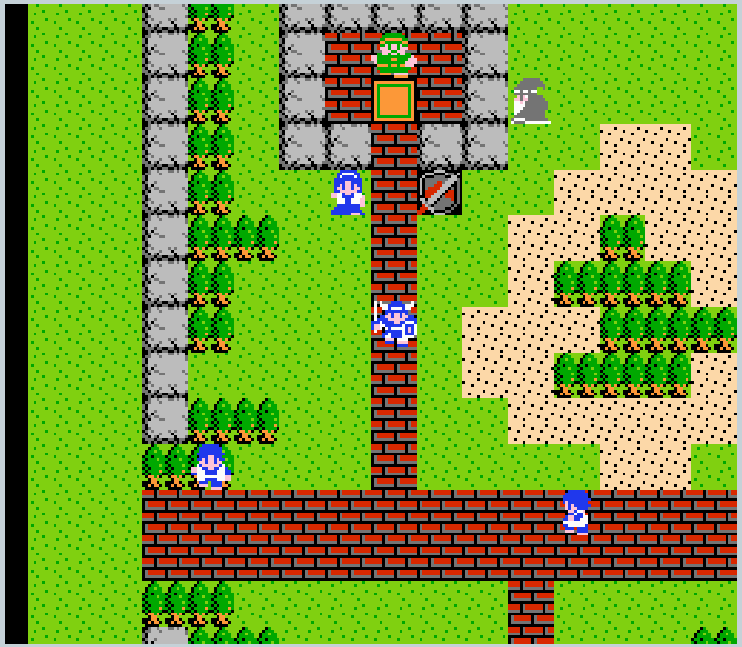
- (common to nearly all OS's)
- Application == Window (to the OS)
  - Application might have sub-windows
    - Buttons, Menus, …
- OS sends messages (events) to the App
  - Upon creation
  - Minimize / Maximize / Resize / Close
  - Mouse / Keyboard / Gamepad events

Events and pygame

- Pygame gives us two choices (not mutually exculsive):
  - **Event Handling**:
    - Get a *single* waiting Event **object** with pygame.event.poll()
      - The name's unfortunate…
    - If this event is important, handle it, else ignore
    - Only one notification per event
  - **Device-polling**:
    - You must call event.poll, but ignore result
    - Pygame saves the input-state to variables
    - We poll those variables every frame.
    - Call pygame.event.pump()
      - Process all waiting events
      - Saves result in status variables
    - Access the status variables

Event-Handling

- Code snippet [on board]
- https://www.pygame.org/docs/ref/event.html
- When to use?
- Example:
  - Print common event codes
  - Dragon-Warrior type movement

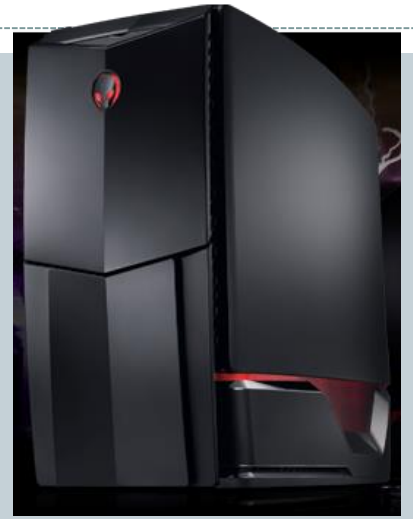


Device-Polling

- Code snippet [on board]
- Common functions:
  - pygame.key.get_pressed()
  - pygame.mouse.get_pressed()
  - pygame.mouse.get_pos()
- When to use?
- Example:
  - Smooth character movement (v 1.0)

A Game Loop "problem"

Do these run at the same speed?



- So…a game loop run on these would be faster / slower.
- Character moving at 1 pixels / update.
  - Suppose we write it on the IIgs
  - What would it look like on the Alienware?

A Game Loop "problem", cont.

- Solution #1: Insert pauses
  - Set a desired frame rate (60fps == 1/60s per frame)
  - Calculate time since last frame
  - Insert a time.sleep if necessary (or use the clock object)
  - Advantage: simple
  - Disadvantages:
    - If the computer can't reach 60 fps, there will be differences.
    - time.sleep pauses *everything*. We could do useful work during that pause
      - AI calculations
      - physics
      - etc.
  - When to use:
    - Fighting games (e.g. Street Fighter)
    - Physics-heavy games (e.g. Kerbal Space Program)
      - Although physics is often done a separate thread...

A Game Loop "problem", cont.

- Solution #2: Adjust movements based on dt
  - Let the computer run as fast as possible
  - Calculate dT (the time since last frame)
  - Express desired movments speeds as a **velocity** (e.g. 100 pixels / **s**)
  - Each frame, velocity * dT is the distance to move.
    - On a slow computer…
    - On a fast computer…
  - Disadvantages: we have to do an extra multiply every frame.
  - When to use:
    - PC games
    - When not doing the games on previous slide☺

Pygame clock object

- [Example: smooth character movement v 2.0]
  - Discuss how to implement both approaches.
- [Example: smooth character movement v 3.0 – add sprite sheet]
  - Reiner's tilesets: http://www.reinerstilesets.de/