Lecture 10

The Game Loop
2110-Level Apps are Event Driven

Generates event e and then calls method(e) on listener

Registers itself as a listener

JFrame

Application

Listener

@105dc

method(Event)
Limitations of the Event Model

- Program only reacts to user input
  - Nothing changes if user does nothing
  - Desired behavior for productivity apps

- Games continue without input
  - Character animation
  - Clock timers
  - Enemy AI
  - Physics Simulations

Game Loop
The Game Loop

- Update
- Draw

Game Loop
The Game Loop

Update

Draw

Cull non-visible objects
Transform visible objects
Draw to backing buffer
Display backing buffer
The Game Loop

Update

Receive player input
Process player actions
Process NPC actions
Interactions (e.g. physics)

Draw

Cull non-visible objects
Transform visible objects
Draw to backing buffer
Display backing buffer

Game Loop
The Game Loop

60 times/s = 16.7 ms

Update
- Receive player input
- Process player actions
- Process NPC actions
- Interactions (e.g. physics)

Draw
- Cull non-visible objects
- Transform visible objects
- Draw to backing buffer
- Display backing buffer
Few Words on Drawing

• Drawing needs to be **fast**!
  • Do as little computation as possible
  • But draw as few objects as possible

• Is this a contradiction?
  • Need to compute what to draw
  • So drawing *less* has extra overhead

• **Rule**: do *not* modify game state in draw
  • Any extra computation is local-only
The Game Loop

Update

Receive player input
- Process player actions
- Process NPC actions
- Interactions (e.g. physics)

Draw

Cull non-visible objects
- Transform visible objects
- Draw to backing buffer
- Display backing buffer
Player Input

• Traditional input is event-driven
  • Events capture state of controller
  • OS/VM generates events for you
  • Listeners react to events

• Game loop uses **polling** for input
  • Ask for controller state at start of loop
  • **Example**: What is joystick position?
  • If no change, do no actions that loop
Problem with Polling

- Only one event per update loop
  - Multiple events are lost
  - Example: Fast typing

- Captures state at beginning
  - Short events are lost
  - Example: Fast clicks

- Event-driven does not have these problems
  - Captures all events as they happen
  - But capture still has a frame-rate resolution
Combining Input Approaches

- LibGDX input is extremely flexible
  - Every input type supports events OR polling

- **Polling**: `Input` interface
  - Access it through the static class GDX.Input
  - Allows you to read the input state right now

- **Events**: `InputProcessor` interface
  - Register it with the appropriate input device
  - Works exactly like Swing listeners
Problem: Timing

public class MyProcessor implements InputProcessor {
    public void keyTyped(char c) {
        // Do something with input
    }
}

Game Loop
public class MyProcessor implements InputProcessor {
    public void keyTyped(char c) {
        // Do something with input
    }
}
Classic Producer-Consumer Problem

Consumer

Update

Draw

Producer

Input Handler

Game Loop
Classic Producer-Consumer Problem

Consumer

Producer

Update

Draw

Buffer

Input Handler

Game Loop
Classic Producer-Consumer Problem

Consumer

Producer

Game Loop

Update

Draw

Buffer

Input Handler

Answer
Classic Producer-Consumer Problem

Consumer

Producer

Game Loop

Update

Draw

Buffer

Polling!

Answer

Input Handler
Classic Producer-Consumer Problem

Producer

Input Handler

Buffer

Overwriting?

Check

Polling!

Update

Draw

Consumer

Game Loop

Game Loop
Buffering Input

- If overwriting an issue, need an **event queue**
  - Input processor writes at end of the queue
  - Game loop reads from the front of queue

```
  event₁ → event₂ → event₃ → event₄ → event₅
```

- Generally requires multiple **threads**
  - Event handler is (usually) OS/VM provided thread
  - Game loop itself is an additional thread
Event Handlers: Really Necessary?

- Most of the time: No
  - Frame rate is short: 16.7 ms
  - Most events are > 16.7 ms
  - Event loss not catastrophic

- Buffering is sometimes undesirable
  - Remembers every action ever done
  - But may take a longer time to process
  - If takes too long, just want to abort
Picking the Right Input

Polling
- When game loop is explicit
  - Actively animating screen
  - Must time input correctly
- **Example**: playing the game

Event Driven
- When game loop is implicit
  - Art assets are largely static
  - Nothing to do if no input
- **Example**: a menu screen
The Game Loop

The Game Loop consists of two main steps: Update and Draw.

**Update**
- Receive player input
- Process player actions
- Process NPC actions
- Interactions (e.g., physics)

**Draw**
- Cull non-visible objects
- Transform visible objects
- Draw to backing buffer
- Display backing buffer
Player Actions

- Actions alter the game state
  - Can alter player state: movement
  - Can alter opponent state: damage

- Player actions correspond to user input
  - Choice is determined by input controller
  - Else action is performed by computer

- These are your game verbs!
Abstract Actions from Input

• **Actions**: functions that modify game state
  - move(dx,dy) modifies x, y by dx, dy
  - attack(o) attacks opponent o

• Input controller **maps** input to actions
  - Read input state from controller
  - Pick an action and call that function

• Input handler should never alter state directly!
Abstract Actions from Input

- **Actions**: functions that modify game state
  - `move(dx,dy)` modifies x, y by dx, dy
  - `attack(o)` attacks opponent o

- Input controller **maps** input to actions
  - Read input state from controller
  - Pick an action and call that function

- Input handler should never alter state directly!
  - Input handler only identifies the action
The Game Loop

- **Update**
  - Receive player input
  - Process player actions
  - **Process NPC actions**
  - Interactions (e.g. physics)

- **Draw**
  - Cull non-visible objects
  - Transform visible objects
  - Draw to backing buffer
  - Display backing buffer
NPC: Non-Player Character

- NPC is an intelligent computer-controlled entity
  - Unlike a physics object, it can act, not just interact
  - Sometimes called an *agent*

- NPCs have their own actions/verbs
  - But no input controller to choose

- Work on *sense-think-act* cycle
  - **Sense:** perceive the world around it
  - **Think:** choose an action to perform
  - **Act:** update the game state

Game Loop
Act versus Sense-Think

- Act should be *very* fast!
  - Function to update state
  - **Example**: apply velocity
  - Exactly like the player
- Sense-think unique to NPC
  - The *hard* computation
  - Focus of AI lectures
- **Multiplayer**: Replace sense-think with human decision
Problem with Sensing

- Sensing may be slow!
  - Consider *all* objects

- Example: morale
  - *n* knights, *n* skeletons
  - Knights fear skeletons
  - Proportional to # seen

- Count skeletons in view
  - $O(n)$ to count skeletons
  - $O(n^2)$ for all units

Game Loop

<table>
<thead>
<tr>
<th>Time per tick</th>
<th>3 units</th>
<th>2 units</th>
<th>1 unit</th>
</tr>
</thead>
</table>

Sensing may be slow! Consider *all* objects.

Example: morale
- *n* knights, *n* skeletons
- Knights fear skeletons
- Proportional to # seen

Count skeletons in view
- $O(n)$ to count skeletons
- $O(n^2)$ for all units
Processing NPCs

- Naïve solution: sequentially

- **Problem**: NPCs react too fast!
  - Each reads the actions of previous
  - Even before drawn on screen!
Processing NPCs

- **Naïve solution:** sequentially

- **Problem:** NPCs react too fast!
  - Each reads the actions of previous
  - Even before drawn on screen!

- **Idea:** only react to what can see
  - \textbf{Choose} actions, but don’t perform
  - Once all chosen, then perform
  - Another reason to abstract actions
Processing Actions in Lab 3

- Decides whether to shoot
- Stores intent in the object
- But **DOES NOT** shoot
- Waits until objects commit
- Checks intent in Ship object
- Performs action for intent
**Problem: Pathfinding**

- **Focus of Game Lab 2**
  - Crucial if top view
  - Major area of research
- **Potentially very slow**
  - \( n \) NPCs, \( g \) grid squares
  - Dijkstra: \( O(g^2) \)
  - For each NPC: \( O(ng^2) \)
- **Moving obstacles?**
Problem: Pathfinding

- Focus of Game Lab 2
  - Crucial if top view
  - Major area of research

- Potentially very slow
  - Dijkstra: $O(g^2)$
  - For each NPC: $O(ng^2)$

- Moving obstacles?

Often more than 16.7ms
Asynchronous Pathfinding

Thread 1

Game Loop

Update

Check for request
Compute answer
Store in buffer

Buffer

Answer

Request path

Thread 2

Pathing Engine

Looks like input buffering!

Game Loop
Asynchronous Pathfinding

- NPCs do not get answer right away
  - Check every loop until answered
  - Remember request; do not ask again

- What to do until then?
  - Act, but don’t think!
  - If nothing, **fake** something
  - “Stomping Feet” in RTSs
The Game Loop

- Update
  - Receive player input
  - Process player actions
  - Process NPC actions
  - Interactions (e.g. physics)

- Draw
  - Cull non-visible objects
  - Transform visible objects
  - Draw to backing buffer
  - Display backing buffer
Purpose of a Physics Engine

• Moving objects about the screen
  • **Kinematics**: Without regard to external forces
  • **Dynamics**: The effect of forces on the screen

• Collisions between objects
  • **Collision detection**: Did a collision occur?
  • **Collision resolution**: What do we do?

• More on this issue later (~Spring Break)
Physics Engines: Two Levels

• **White Box**: Engine corrects movement errors
  - Update object state ignoring physics
  - Physics engine nudges object until okay

• **Black Box**: Engine handles everything
  - Do not move objects or update state
  - Give forces, mass, velocities, etc. to engine
  - Engine updates to state that is *close enough*
Almost everything is in loop
- Except asynchronous actions
- Is enough for simple games

How do we organize this loop?
- Do not want spaghetti code
- Distribute over programmers
Architecture: Organizing Your Code

Game Engine
- Input Devices
- Discrete Simulation Engine
- Compiler
- Data Management Layer

Game Content
- Character Scripts
- Character Data
- UI Elements
- Models and Textures
- Sounds

Player
- GUI
- Rendering Engine
- Audio Engine

Programmer
- Physics Engine
- AI Engine (e.g. Pathfinding)

Designer or Modder

Game Loop
Architecture: Organizing Your Code

Game Engine
- Implementation
- Interface
- Input Devices
- GUI
- Rendering Engine
- Audio Engine

Game Content
- Character Scripts
- Character Data
- UI Elements
- Models and Textures
- Sounds

Game Loop
Where Did This Come From?

Next Time!