Lecture 6

Game Architecture Revisited
Recall: 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
The Game Loop

- 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
Model-View-Controller Pattern

**Model**
- Defines/manages the program data
- Responds to the controller requests

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

**View**
- Displays model to the user/player
- Provides interface for the controller

Calls the methods of...
The Game Loop and MVC

- **Model**: The game state
  - Value of game resources
  - Location of game objects

- **View**: The draw phase
  - Rendering commands only
  - Major computation in update

- **Controller**: The update phase
  - Alters the game state
  - Vast majority of your code
Structure of a CUGL Application

Main → Application

Scene

Models → Root Node

Scene

Models → Root Node
Structure of a CUGL Application

Main

Application

App Configuration

Scene

Models

Root Node

Scene

Models

Root Node

Memory policy (future lecture)
Structure of a CUGL Application

Main → Application

- Active
  - Scene
    - Models
    - Root Node
  - Dormant
  - Scene
    - Models
    - Root Node

Architecture Revisited
Structure of a CUGL Application

Main

Application

Controller(s)

Scene

Models

Root Node

View

Scene

Models

Root Node

Architecture Revisited
# The Application Class

<table>
<thead>
<tr>
<th><strong>onStartup()</strong></th>
<th><strong>update()</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Handles the game assets</td>
<td></td>
</tr>
<tr>
<td>• Attaches the asset loaders</td>
<td></td>
</tr>
<tr>
<td>• Loads immediate assets</td>
<td></td>
</tr>
<tr>
<td>• Starts any global singletons</td>
<td></td>
</tr>
<tr>
<td>• <strong>Example</strong>: AudioChannels</td>
<td></td>
</tr>
<tr>
<td>• Creates any player modes</td>
<td></td>
</tr>
<tr>
<td>• But does not launch <em>yet</em></td>
<td></td>
</tr>
<tr>
<td>• Wait for assets to load</td>
<td></td>
</tr>
<tr>
<td>• Like <code>GDXRoot</code> in 3152</td>
<td></td>
</tr>
<tr>
<td>• Called each animation frame</td>
<td></td>
</tr>
<tr>
<td>• Manages gameplay</td>
<td></td>
</tr>
<tr>
<td>• Converts input to actions</td>
<td></td>
</tr>
<tr>
<td>• Processes NPC behavior</td>
<td></td>
</tr>
<tr>
<td>• Resolves physics</td>
<td></td>
</tr>
<tr>
<td>• Resolves other interactions</td>
<td></td>
</tr>
<tr>
<td>• Updates the scene graph</td>
<td></td>
</tr>
<tr>
<td>• Transforms nodes</td>
<td></td>
</tr>
<tr>
<td>• Enables/disables nodes</td>
<td></td>
</tr>
</tbody>
</table>
# The Application Class

<table>
<thead>
<tr>
<th><strong>onStartup()</strong></th>
<th><strong>update()</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Handles the game assets</td>
<td>- Called each animation frame</td>
</tr>
<tr>
<td>- Attaches the asset loaders</td>
<td>- Manages gameplay</td>
</tr>
<tr>
<td>- Loads immediate assets</td>
<td>- Converts input actions into game actions</td>
</tr>
<tr>
<td>- Starts any scripts or player modes</td>
<td>- Resolves other interactions</td>
</tr>
<tr>
<td>- But does not launch yet</td>
<td>- Updates the scene graph</td>
</tr>
<tr>
<td>- Waits for assets to load</td>
<td>- Transforms nodes</td>
</tr>
<tr>
<td>- Like <code>GDXRoot</code> in 3152</td>
<td>- Enables/disables nodes</td>
</tr>
</tbody>
</table>

`onShutdowm()` cleans this up

Does not draw! Handled separately
Application Structure

Ownership

Scene Controller

Subcontroller

Model

Subcontroller

Model

View

Collaboration

Model

Architecture Revisited
Application Structure

Ownership

Collaboration
- Must import class/interface
- Instantiates an object **OR**
- Calls the objects methods

Ownership
- Instantiated the object
- Responsible for disposal
- Superset of collaboration

Collaboration
Avoid Cyclic Collaboration

collaborates with

Controller

collaborates with

The image illustrates the concept of avoiding cyclic collaboration in design, showing a before and after scenario. Initially, entities Y and X collaborate cyclically, which is represented by the arrows. After applying the solution, the collaboration is redirected through a controller Z, ensuring a linear flow of communication.
Scene Structure

Scene Controller

Subcontroller

Subcontroller

View

Model

Model

Model

Architecture Revisited
CUGL Views: Scene Graphs
CUGL Views: Scene Graphs

Architecture Revisited
CUGL Views: Scene Graphs
## Model-Controller Separation (Standard)

### Model
- Store/retrieve **object data**
  - Limit access (getter/setter)
  - Preserve any invariants
  - Only affects this object
- Implements **object logic**
  - Complex actions on model
  - May affect multiple models
  - **Example**: attack, collide

### Controller
- Process **user input**
  - Determine action for input
  - **Example**: mouse, gamepad
  - Call action in the model

---

**Traditional controllers are “lightweight”**
Classic Software Problem: Extensibility

- **Given**: Class with some base functionality
  - Might be provided in the language API
  - Might be provided in 3rd party software

- **Goal**: Object with *additional* functionality
  - Classic solution is to subclass original class first
  - **Example**: Extending GUI widgets (e.g. Swing)

- But subclassing does not always work…
  - How do you extend a *Singleton* object?
Problem with Subclassing

- Games have *lots* of classes
  - Each game entity is different
  - Needs its own functionality (e.g. object methods)

- Want to avoid **redundancies**
  - Makes code hard to change
  - Common source of bugs

- Might be tempted to **subclass**
  - Common behavior in parents
  - Specific behavior in children

![Subclassing Diagram](image-url)
Problem with Subclassing

- Games have *lots* of classes
  - Each game entity is different
  - Needs its own functionality (e.g. object methods)
- Want to avoid **redundancies**
  - Makes code hard to change
  - Common source of bugs
- Might be tempted to **subclass**
  - Common behavior in parents
  - Specific behavior in children

![Diagram](image)
Model-Controller Separation (Standard)

**Model**

- Store/retrieve **object data**
  - Limit access (getter/setter)
  - Preserve any invariants
  - Only affects this object

- Implements **object logic**
  - Complex actions on model
  - May affect multiple models
  - **Example**: attack, collide

---

Redundant Behavior
Model-Controller Separation (Alternate)

**Model**
- Store/retrieve **object data**
  - Limit access (getter/setter)
  - Preserve any invariants
  - Only affects this object

**Controller**
- Process **game actions**
  - Determine from input or AI
  - Find *all* objects effected
  - Apply action to objects

**Process interactions**
- Look at current game state
- Look for “triggering” event
- Apply interaction outcome

In this case, models are lightweight
Does Not Completely Solve Problem

- Code **correctness** a concern
  - Methods have specifications
  - Must use according to spec
- Check correctness via **typing**
  - Find methods in object class
  - **Example**: orc.flee()
  - Check type of parameters
  - **Example**: force_to_flee(orc)
- **Logical** association with type
  - Even if not part of class

Can I **flee**?
Issues with the OO Paradigm

• Object-oriented programming is very **noun-centric**
  • All code must be organized into classes
  • Polymorphism determines capability via type

• OO became popular with **traditional MVC pattern**
  • Widget libraries are nouns implementing view
  • Data structures (e.g. CS 2110) are all nouns
  • Controllers are not necessarily nouns, but lightweight

• Games, interactive media break this paradigm
  • View is animation (process) oriented, not widget oriented
  • Actions/capabilities only loosely connected to entities
### Programming and Parts of Speech

#### Classes/Types are Nouns
- Methods have verb names
- Method calls are sentences
  - `subject.verb(object)`
  - `subject.verb()`
- Classes related by *is-a*
  - Indicates class a subclass of
  - **Example**: String is-a Object
- Objects are class *instances*

#### Actions are Verbs
- Capability of a game object
- Often just a simple function
  - `damage(object)`
  - `collide(object1,object1)`
- Relates to objects via *can-it*
  - **Example**: Orc can-it attack
  - Not necessarily tied to class
  - **Example**: swapping items
Duck Typing: Reaction to This Issue

- “Type” determined by its
  - Names of its methods
  - Names of its properties
  - If it “quacks like a duck”

- Python has this capability
  - `hasattr(<object>,<string>)`

- This has many problems
  - Correctness is a nightmare

Java:
```java
public boolean equals(Object h) {
    if (!(h instanceof Person)) {
        return false;
    }
    Person ob = (Person)h;
    return name.equals(ob.name);
}
```

Python:
```python
def __eq__(self,ob):
    if (not (hasattr(ob,'name')))
        return False
    return (self.name == ob.name)
```
Duck Typing: Reaction to This Issue

- "Type" determined by its
  - Names of its methods
  - Names of its properties
  - If it "quacks like a duck"
- Python has this capability
  - `hasattr(object, string)`
    - True if object has attribute or method of that name
- This has many problems
  - Correctness is a nightmare

Java:

```java
public boolean equals(Object h) {
    if (!(h instanceof Person)) {
        return false;
    }
    Person ob = (Person) h;
    return name.equals(ob.name);
}
```

What do we really want?

- Capabilities over properties
- Extend capabilities without necessarily changing type
- Without using new languages

Again, use software patterns

Python:

```python
def __eq__(self, ob):
    if (not (hasattr(ob, 'name'))):
        return False
    return (self.name == ob.name)
```

Architecture Revisited
Possible Solution: Decorator Pattern

Request → Decorator Object

Original Functionality

New Functionality

Original Object
Java I/O Example

```
InputStream input = System.in;

Reader reader = new InputStreamReader(input);

BufferedReader buffer = new BufferedReader(reader);
```

- **Built-in console input**
- **Make characters easy to read**
- **Read whole line at a time**

Most of java.io works this way
Alternate Solution: Delegation Pattern

Inversion of the Decorator Pattern
Alternate Solution: Delegation Pattern

Inversion of the Decorator Pattern
public class SortableArray extends ArrayList {
    private Sorter sorter = new MergeSorter();  
    new QuickSorter();
    public void setSorter(Sorter s) { sorter = s; }
    public void sort() {
        Object[] list = toArray();
        sorter.sort(list);
        clear();
        for (o:list) { add(o); }
    }
}

public interface Sorter {
    public void sort(Object[] list);
}

Example: Sort Algorithms
## Comparison of Approaches

<table>
<thead>
<tr>
<th>Decoration</th>
<th>Delegation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Pattern applies to <em>decorator</em></td>
<td>• Applies to <em>original object</em></td>
</tr>
<tr>
<td>• Given the original object</td>
<td>• You designed object class</td>
</tr>
<tr>
<td>• Requests through decorator</td>
<td>• All requests through object</td>
</tr>
<tr>
<td>• <strong>Monolithic</strong> solution</td>
<td>• <strong>Modular</strong> solution</td>
</tr>
<tr>
<td>• Decorator has all methods</td>
<td>• Each method can have own</td>
</tr>
<tr>
<td>• “Layer” for more methods</td>
<td>delegate implementation</td>
</tr>
<tr>
<td>(e.g. Java I/O classes)</td>
<td>• Like higher-order functions</td>
</tr>
<tr>
<td>• Works on <em>any</em> object/class</td>
<td>• Limited to classes you make</td>
</tr>
</tbody>
</table>
The Subclass Problem Revisited

Redundant Behavior
Summary

- Games naturally fit a **specialized MVC** pattern
- Want *lightweight* models (mainly for serialization)
- Want *heavyweight* controllers for the game loop
- View is specialized rendering with few widgets
- CUGL view is handled in scene graphs
- Proper design leads to unusual OO patterns
  - Subclass hierarchies are unmanageable
  - **Component-based design** better models actions