Lecture 8

Component Design
Recall: 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
Recall: 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:

- NPC
  - Warrior
    - Human Warrior
    - Orc Warrior
  - Archer
    - Human Archer
    - Orc Archer

No Help

Redundant Behavior
Alternative: Decorator Pattern

Request → Decorator Object → Original Functionality → Original Object

New Functionality

Alternative: Decorator Pattern
Alternate: Delegation Pattern

Inversion of the Decorator Pattern
Issues with Static Typing

Original object class

Original Object

obj.request(arg1, ..., argn)

Delegate Object

Method in original class

Request

Reference to delegate

Forward Request
The Subclass Problem Revisited

Delegates?

NPC

Human

Orc

Human Warrior

Human Archer

Orc Warrior

Orc Archer

Redundant Behavior
Component-Based Programming

- **Role**: Set of capabilities
  - Class with very little data
  - A collection of methods
- **Add it to object as delegate**
  - Object gains those methods
  - Acts as a “function pointer”
- **Can-it**: search object roles
  - Check class of each role
  - Better than duck typing
  - Possible at compile time?

Field storing a single delegate or a set of delegates
Aside: Objective-C

- Delegates are emphasized in **Objective-C**
  - ObjC **protocols** function like Java interfaces
  - But implement with a delegate, not a subclass

- Often leverages **message passing**
  - Method call is considered a message
  - Can call *any* method on *any* object
  - Will get runtime error if method not supported
  - Heavily optimized in latest Apple compilers
Aside: Objective-C

- Delegates are emphasized in **Objective-C**
  - ObjC *protocols* function like Java interfaces
  - But implement with a delegate, not a subclass
- Often leverages message passing
  - Method call is considered a message
  - Can call *any* method on *any* object
  - Will get runtime error if method not supported
- Heavily optimized in latest Apple compilers
Working With Static Typing

**Uniform Method**

\[o.\text{doSomething}(\text{kitchenSink})\]

**Direct Access**

\[
\text{if (o has role) } \{
  \text{extract role from o}
  \text{role.\text{doSomething}(info)}
\}
\]
## Working With Static Typing

### Uniform Method

```
0.doSomething(kitchenSink)
```

### Direct Access

```
if (o has role) {
    extract role from o
    role.doSomething(info)
}
```

---

**Problem:**

Multiple capabilities
Working With Static Typing

**Uniform Method**

```
o.doSomething(kitchenSink)
```

**Direct Access**

```
if (o has role) {
    extract role from o
    role.doSomething(info)
}
```

**Problem:**

- Multiple capabilities

**Problem:**

- Big switch statements
Entities Need Both *Is-a* and *Can-it*

<table>
<thead>
<tr>
<th>Table</th>
<th>Chair</th>
</tr>
</thead>
</table>

Objects share same capabilities *in theory*. But certain actions are *preferred* on each.
# Solving the Problem

<table>
<thead>
<tr>
<th>Can-It</th>
<th>Is-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Indicated by object <strong>roles</strong></td>
<td>• Indicated by object <strong>class</strong></td>
</tr>
<tr>
<td>• Component indicates role</td>
<td>• But could do more than that</td>
</tr>
<tr>
<td>• Check type of each role</td>
<td>• Keep a <strong>bag of types</strong></td>
</tr>
<tr>
<td>• So again a typing solution</td>
<td>• Contains of the descriptions that apply to this object</td>
</tr>
<tr>
<td>• <strong>“Formal duck typing”</strong></td>
<td>• <strong>Example</strong>: table, heirloom</td>
</tr>
<tr>
<td>• Not limited to method name</td>
<td>• Why would we do this?</td>
</tr>
<tr>
<td>• Role type indicates action name, not a noun</td>
<td></td>
</tr>
</tbody>
</table>
Fast Can-It Testing: Bit Vectors

• Give each object a bit-vector to cover types
  • Each role is a position in the bit vector
  • Example: 0100101
Fast Can-It Testing: Bit Vectors

• Give each object a bit-vector to cover types
  • Each role is a position in the bit vector
  • **Example:** 0100101
    - Can Fight
    - Can Heal
    - Can Build

• Use array if more than 32/64 roles
Fast Can-It Testing: Bit Vectors

- Give each object a bit-vector to cover types
  - Each role is a position in the bit vector
  - Example: 0100101
    - Can Build
    - Can Fight
    - Can Heal
- Use array if more than 32/64 roles
- Test is quick logical operation.
  - Example: `vector & CAN_FIGHT`
  - Faster than dynamic typing (why?)
# Model-Controller Separation Revisited

## Model
- Store/retrieve **object data**
  - Preserve any invariants
  - Data may include delegates
  - Determines **is-a** properties

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

## Components
- Process **game actions**
  - Attached to a entity (model)
  - Uses the model as context
  - Determines **can-it** properties
Model-Controller Separation Revisited

**Model**
- Store/retrieve **object data**
- Preserve any invariants
- Data may include delegates
- Determines is-a properties

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

**Components**
- Process **game actions**
  - Attached to a entity (model)
  - Uses the model as context
  - Determines can-it properties

**Is this always the best idea?**
Recall: Lab 3 in CS 3152

Current shells: 20
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
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

Ship.cs

GameplayController.cs

Attach components to controller?
Related: Handling the View

- Way too much to draw
  - Backgrounds
  - UI elements
  - Individual NPCs
  - Other moveable objects

- Cannot cram all in Draw

- Put it in game object?
  - But objects are models
  - Violates MVC again
First Attempt: A Drawing Canvas

- Treat display as a **container**
  - Often called a canvas
  - Cleared at start of frame
  - Objects added to container
  - Draw contents at frame end

- Canvas abstracts **rendering**
  - Hides animation details
  - Like working with widget

- Implement `draw(c)` in model
  - Classic heavyweight model
  - No problems with extension

```java
void draw(Canvas c) {
    // Specify perspective
    // Add to canvas
}
```
Problem: Canvas Methods

Model1
draw(canvas)

Model2
draw(canvas)

Canvas
drawShape(…)

Ideal
Problem: Canvas Methods

Model1
   draw(canvas)

In Practice

Canvas
   drawShape1(...)
   drawShape2(...)

Model2
   draw(canvas)
Views and Components

Model1

Component1
drawShape1(...)  

Model2

Component1
drawShape1(...)  

Canvas
drawPrimitive(...)  

Views and Components

Complex, Shape-specific rendering

Component1
drawShape1(…)

Canvas
drawPrimitive(…)

Model1
Slot
Slot
Slot

Model2
Slot
Slot
Slot

Primitive effects
Components and AI

• NPCs use **sense-think-act** cycle
  - **Sense**: perceive the world
  - **Think**: choose an action
  - **Act**: update the game state

• **Act** is stored in components
  - Capabilities given by roles

• Where are **sense** and **think**?
  - In the component/role? **Yes**
  - How do roles interact? **This is hard**

Alert!
Why Do They Need to Interact?

- Because you are trying to pick an action
  - NPC can only do one action (why?)
  - *Think* needs all roles to pick best one

- Why not let **each** component do an action?
  - Some things can be simultaneously (run + shoot)
  - Only concern is *conflicting* actions (left vs. right)
  - Can we guarantee components never conflict?

- Do we have to solve this in thinking-phase?
Recall: Rule-Based AI

- Have a list of **rules**
  - All if-then statements
  - Rules fires if satisfied
  - Need to pick one (conflict resolution)

- Often resolve by **order**
  - Each rule has a **priority**
  - Higher priorities go first
  - “Flattening” conditionals

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$: if event(_1) then act(_1)</td>
<td></td>
</tr>
<tr>
<td>$R_2$: if event(_2) then act(_2)</td>
<td></td>
</tr>
<tr>
<td>$R_3$: if event(_3) then act(_3)</td>
<td></td>
</tr>
<tr>
<td>$R_4$: if event(_4) then act(_4)</td>
<td></td>
</tr>
<tr>
<td>$R_5$: if event(_5) then act(_5)</td>
<td></td>
</tr>
<tr>
<td>$R_6$: if event(_6) then act(_6)</td>
<td></td>
</tr>
<tr>
<td>$R_7$: if event(_7) then act(_7)</td>
<td></td>
</tr>
</tbody>
</table>
Treat Components the Same

- Each component a rule
  - Chooses an action
  - Or possibly none
  - More complex than simple if-then rules

- Conflict resolution
  - Priorities to components
  - Or priorities to actions
  - Resolved *globally*

\[ C_1: \text{warrior component} \]
\[ C_2: \text{archer component} \]
\[ C_3: \text{human component} \]
\[ C_4: \text{orc component} \]

Implies a global AI subsystem
Non-Conflict Resolution?

- What if they do not conflict?
  - Pick just one?
  - Apply them both?
- Only if both **commutative**
  - Treat action as \( f: \mathbb{S} \rightarrow \mathbb{S} \)
  - Require \( f(g(s)) = g(f(s)) \)
  - Easy if state is disjoint
- **Animation** is a big problem
  - Each action has animation
  - Same solution as state?
- Commutative, **disjoint**:
  - \( \text{move}(dx,dy): \)
    \[
    x = x + dx \\
    y = y + dy
    \]
  - \( \text{damage}(d): \)
    \[
    \text{hp} = \text{hp} - d
    \]
- Commutative, **not disjoint**:
  - 2 move actions
  - Addition commutes
  - **Example**: walk, push
Scalable Games Language

<table>
<thead>
<tr>
<th>States</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>123</td>
</tr>
</tbody>
</table>

class Unit {
  public state int id;

  // State with update rules
  public state float x : (x+vx);
  public state float y : (y+vy);
  public state int hp: (hp-dmg);

  // effects
  public effect float vx : avg;
  public effect float vy : avg;
  public effect int dmg : sum;

...
public void run() {
    // Compute # skeletons, group center
    effect int c : sum;
    effect float sx : sum, float sy : sum;
    foreach (Unit u : Extent<Unit>) {
        if (isEnemySkeleton(u) &&
            dist(this, u) < range) {
            c <- 1; sx <- u.x; sy <- u.y;
        }
    }

    // If too many skeletons
    if (c > morale) {
        const float norm = (x-sx/c)*(x-sx/c) +
                           (y-sy/c)*(y-sy/c);
        // Run in opposite direction
        vx <- (x-sx/c)/norm;
        vy <- (y-sy/c)/norm;
    } ...
}
public void run() {
    // Compute # skeletons, group center

    int c : sum;
    float sx : sum;
    float sy : sum;

    foreach (Unit u : Extent<Unit>) {
        if (isEnemySkeleton(u) &&
            dist(this, u) < range) {
            c <- 1;
            sx <- u.x;
            sy <- u.y;
        }
    }

    // If too many skeletons
    if (c > morale) {
        const float norm = (x-sx/c) * (x-sx/c) +
                           (y-sy/c) * (y-sy/c);
        // Run in opposite direction
        vx <- (x-sx/c) / norm;
        vy <- (y-sy/c) / norm;
    } ...

Classic DB optimizations
• Query rewrites
• Join selection
• Automatic indexing
Animation: Blend Trees
Conflicts and Non-Conflicts?

- Create a **DAG structure**
  - Directed acyclic graph
  - $a > b$ if $a$ reachable from $b$
  - $a \mid b$ if neither $a < b$, $a > b$ (e.g. $a$, $b$ are incomparable)

- Use for conflict resolution
  - If comparable, pick greatest
  - If incomparable commute

- Looks good, hard in practice
  - This is a **global** structure
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
- Proper design leads to unusual OO patterns
  - Subclass hierarchies are unmanageable
  - Want *component-based design* to model actions
  - Interesting challenges when incorporating AI