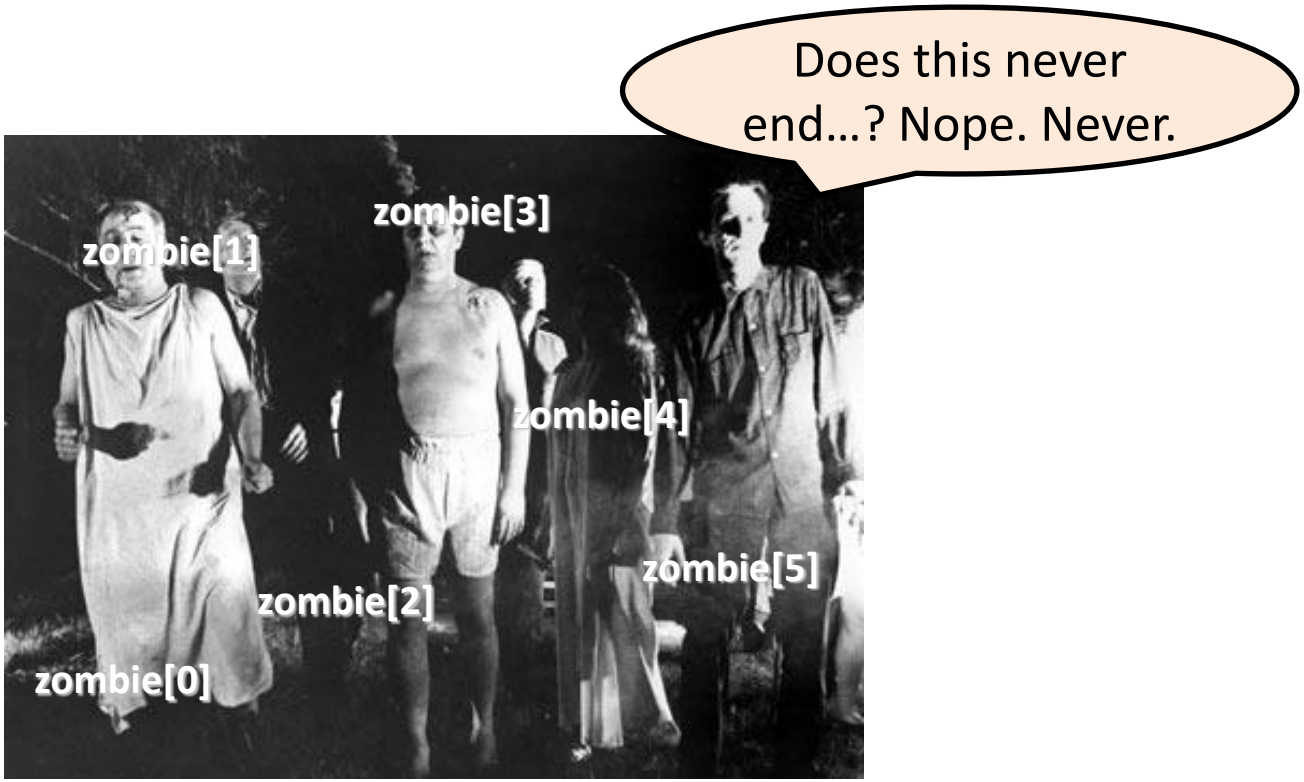
Problem Decomposition Revisited (Again and Again): Object Oriented Design Part IV



Overview

- Designing the API:
  - Find the nouns – the potential objects
  - Define the attributes
  - Define the behavior / methods

Finding the Objects

- Can end up with more or less objects than we really need
  - The process of thinking each class through helps you decide if you need to add or remove classes from the design
  - e.g. Wumpus World
    - You may not need separate classes for the wumpus, the pits and the gold
      - These might simply be attributes of a location
    - You may need to add a class that describes “rooms” in the cave

Finding the Objects (continued)

- e.g. Wumpus World
  - You may not need separate classes for the wumpus, the pits and the gold
    - These might simply be attributes of a location
  - You may need to add a class that describes “rooms” in the cave
- NOTE: This is subjective, and a lot of it is about design decisions
  - If you plan on extending your game you might want to keep the wumpus, etc. as classes so that you can add behaviors
  - If you have an elegant way of implementing rooms without a separate class, you may not want to add that

Defining the Attributes

- These will become the instance variables
- State
  - Instance variables and their values
  - e.g. class: room
    - boolean smelly
    - boolean breezy
    - boolean glittery
    - boolean wumpus
    - boolean pit
    - boolean gold
    - boolean visited

The value of these makes up the state of the room

Defining the Behavior

- These will become the methods
  - e.g. Wumpus World
    - The “room” class does very little except change its state (probably only needs getters and setters)
    - Let’s look at “player” behavior instead:
      - move
      - shoot
      - grab

Defining the Behavior

- What information do these methods need in order to run, what data type will they return and what do they do, exactly?

```
boolean move(int direction)
// moves the player in the specified direction if not at
// boundary, returns whether move was successful or
// not
boolean shoot(int direction)
// shoots the arrow, if the user has one, in the specified
// direction and returns whether the wumpus was hit or
// not; results in the player no longer having an arrow
boolean grab()
// returns true if player is in same room as gold, false
// otherwise
```

Defining the Behavior

```
boolean move(int direction)
// moves the player in the specified direction if not at
// boundary, returns whether move was successful or
// not
boolean shoot(int direction)
// shoots the arrow, if the user has one, in the specified
// direction and returns whether the wumpus was hit or
// not; results in the player no longer having an arrow
boolean grab()
// returns true if player is in same room as gold, false
// otherwise
```

This is technically the API to the class “player”