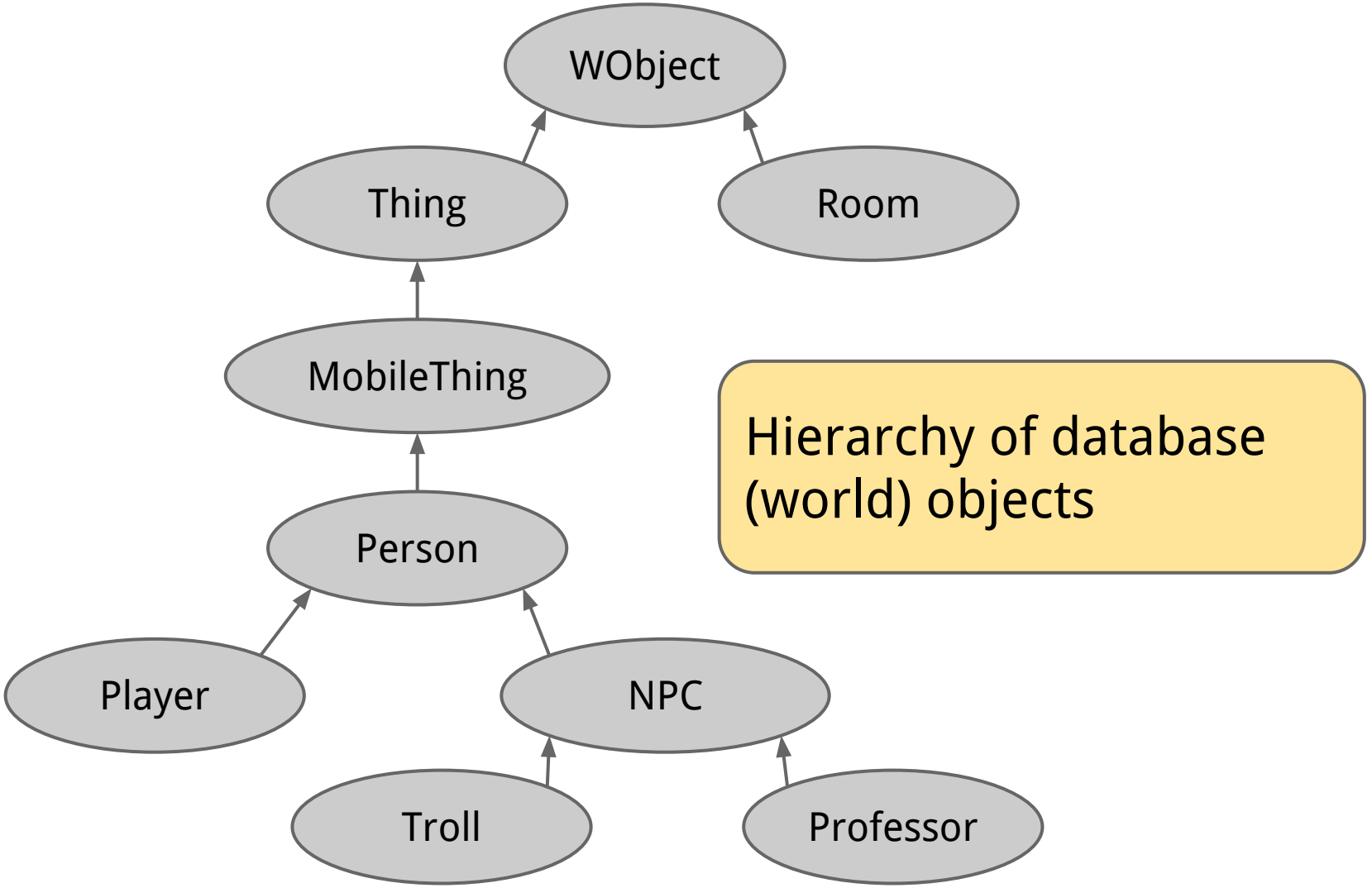
Level 3

Proactive Behavior

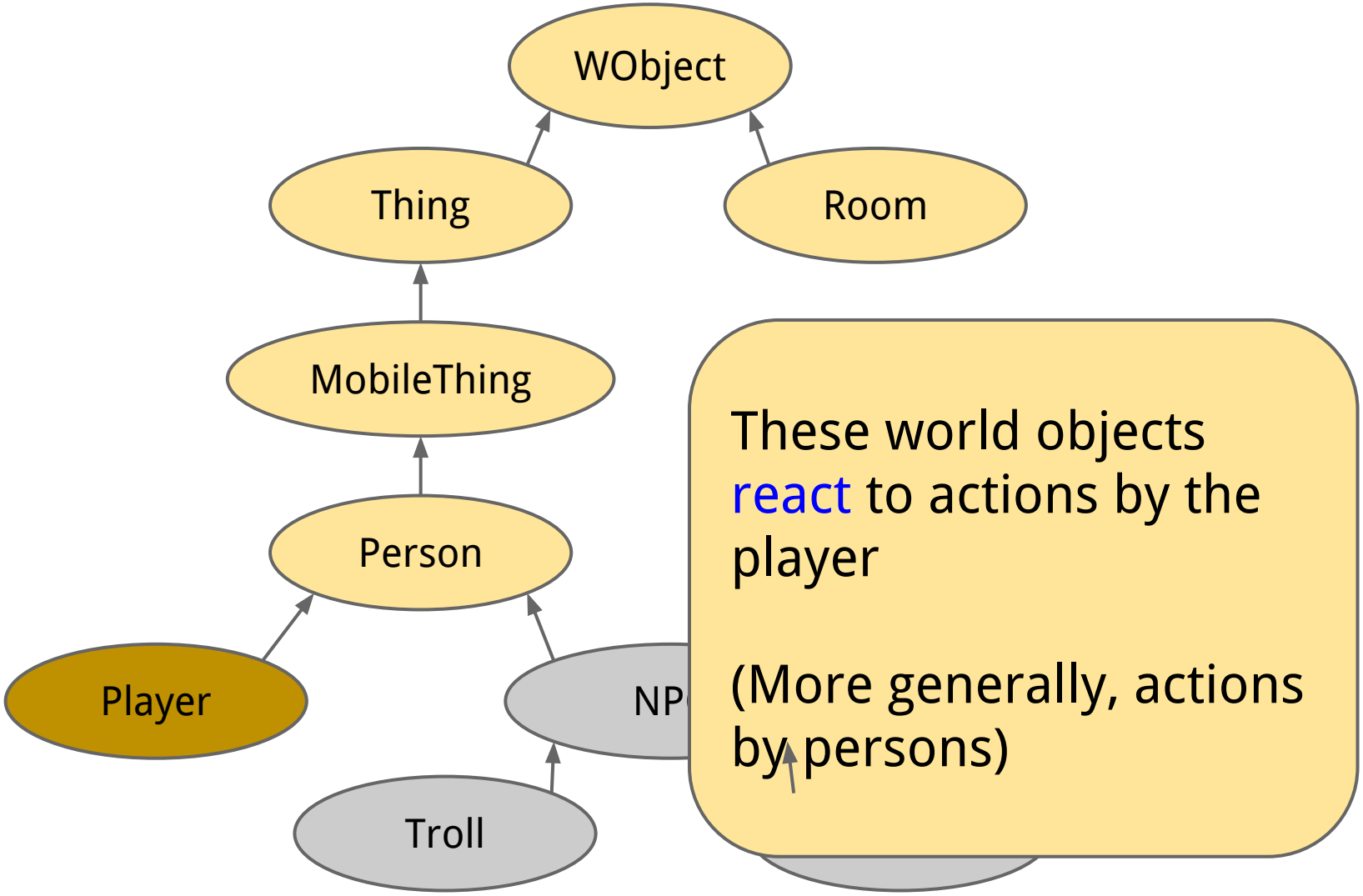
Riccardo Pucella

October 14, 2014

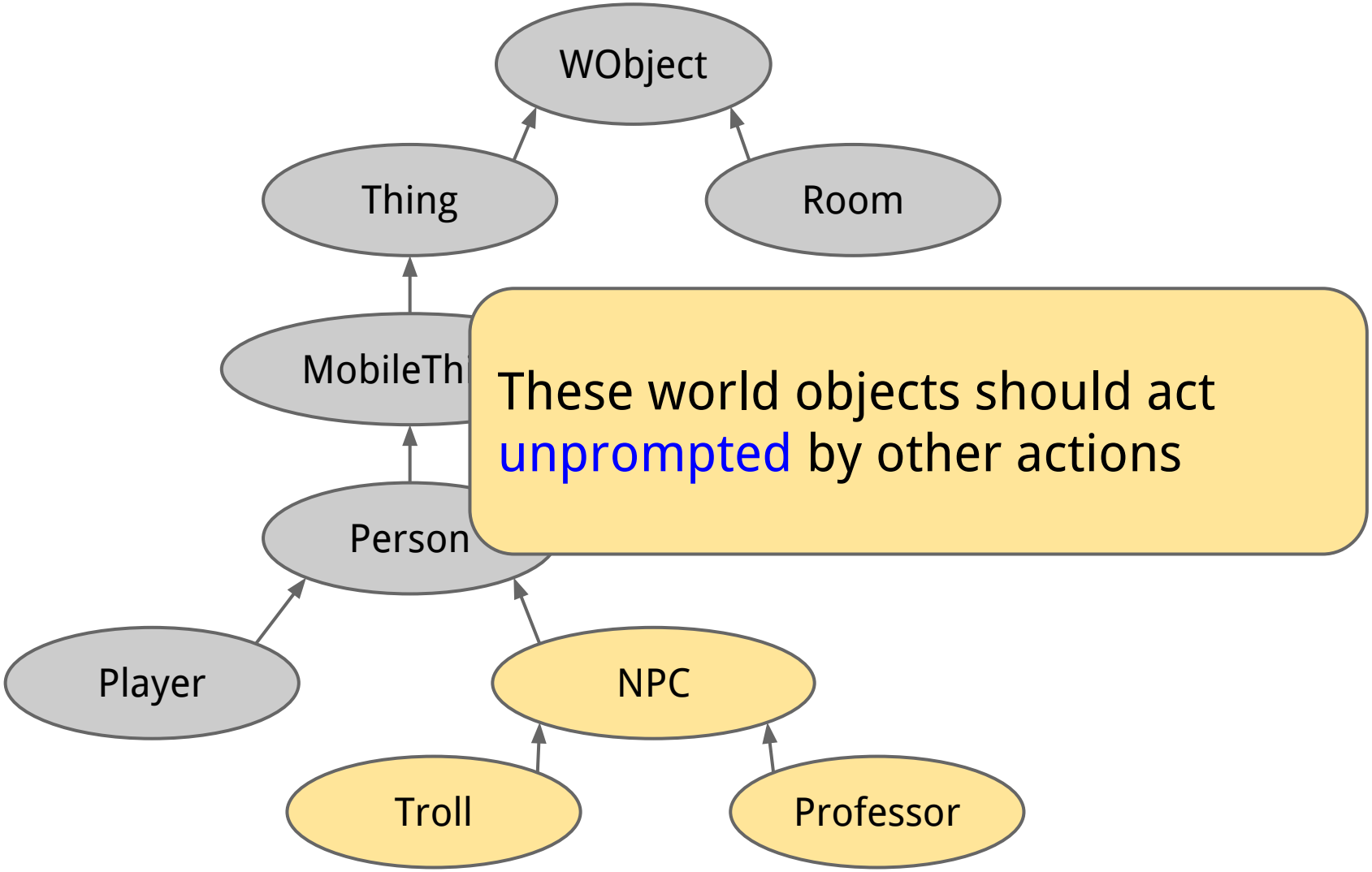
Previously on “Adventure games”…



Reactive behavior



Proactive behavior



Proactive behavior

What is proactive behavior anyway?

Negative example:

- An NPC that runs away when you when you try to talk to him

Positive example:

- An NPC that says hi when you enter
- An NPC that steals your possession

Proactive behavior

What is proactive behavior anyway?

Negative example:

- An NPC that runs
  try to talk to him

Positive example:

- An NPC that says
- An NPC that steals your possession

Working definition:

Behavior that does not arise out of a direct interaction with the player

Proactive behavior

What is proactive behavior anyway?

Negative example:

- An NPC that runs
  try to talk to him

Positive example:

- An NPC that says
- An NPC that steals your possession

Why do I care?

Since game control flow follows player actions, proactive behavior needs to be treated specially

Proactive behavior

What is proactive behavior anyway?

Negative example:

- An NPC that runs
  try to talk to him

Positive example:

- An NPC that says
- An NPC that steals your possession

Approach:

Treat proactive behavior as a computer move in a two-player game

Proactive behavior

Every round:

- Display current state (room description)
- Player move:
  - Get player input
  - Interpret input and perform action
- Computer move:
  - Select actions and perform

Approach 1: loop through objects

To select actions:

- loop through all world objects
- allow each object the chance to perform an action that round
  - method `wanna_do_something(time)`?
  - most objects will do nothing
  - proactive objects will do something
  - not restricted to NPCs

Not great: Wasteful

Inflexible (single point of control)

Approach 2: registration

Any world object interested in proactive behavior registers with a central authority

- loop through all registered world objects
- allow each object the chance to perform an action that round
  - method `wanna_do_something(time)`?

Better: Still inflexible

Approach 3: flexible registration

Instead of an object registering itself, it register a function to be called at every round

Generalizes approach 2:

- Register `self.wanna_do_something(time)`

Flexible:

- Register multiple functions per object
- Registered functions can register functions

Implementing registration

Central authority: the clock

- maintains a data structure recording all the functions that have been registered
- maintains current time (round #)

- method to register a function
  - `register()`

- method to call all registered functions
  - `tick()`

List implementation

Data structure:

- list of functions

```
register (f)  # add function f to list
tick ()       # call all functions in list
                 (passing current time)
```

Ordered list implementation

Data structure:

- list of functions ordered by priority (int)
- a function with priority *i* appears earlier in the list than any function with priority > *i*

```
register (f,priority)
tick ()          # call all functions in
                 # priority order
```

Unregistering functions

What happens if a world object that registered a proactive behavior function gets destroyed?

Need a way to remove a function from the clock

One approach:

- registration returns a unique identifier
- `unregister(id)` removes a registered function with unique identifier `id`

Example: NPC

```
def move_and_take_stuff (self,time):
    if random.randrange(self._restlessness) == 0:
         self.move_somewhere()
    if random.randrange(self._miserly) == 0:
         self.take_something()

def move_somewhere (self):
    exits = self.location().exits()
    if exits:
        dir = random.choice(exits.keys())
        self.go(dir)

def take_something (self):
    ...
```

Example: Troll

```
def eat_people (self,time):
    if random.randrange(self._hunger) == 0:
        people = self.people_around()
        if people:
            victim = random.choice(people)
            self.location().report(self.name() +
                                   ' takes a bite out of ' +
                                   victim.name())
            victim.suffer(random.randint(1,3))
        else:
            self.location().report(self.name() +
                                   "'s belly rumbles")
```

Example: Professor

```
_topics = ['Turing machines',
           'the lambda calculus',
           'Godel']

def lecture (self,time):
    if random.randrange(self._professorial) == 0:
        if self.people_around():
            self.location().report(self.name()+
                                   ' starts lecturing about '+
                                   random.choice(self._topics))
        else:
            self.location().report(self.name()+
                                  ' mutters to himself about '+
                                  random.choice(self._topics))
```

Example: Firecracker

```
class Firecracker (MobileThing):

    def __init__ (self,name,loc):
        MobileThing.__init__(self,name,loc)
        self._countdown = 1
        self._id = 0

    def use (self,actor):
        if self._id > 0:
            actor.say('It is already pulled.')
        else:
            actor.say('I pull on the firecracker.')
            self._id = Player.clock.register (self.pop,5)
```

Example: Firecracker

```
def pop (self,time):
   self._countdown -= 1
   if self._countdown == 0:
      loc = self.location()
      if loc.is_room():
          loc.report("The firecracker pops!")
      else:
          loc.location().report("A muffled pop...")
      Player.clock.unregister(self._id)
      self.destroy()
```