Level 4

Event Queues

Riccardo Pucella

October 28, 2014

Proactive behavior

Recall:

> proactive behavior is behavior that arises spontaneously, as opposed to reaction to a player's behavior

Can think of it as a second player move in a two-player game

Proactive behavior

Proactive behavior in adventure games:

- Based on the notion of a round

- Register functions with a clock
- At every round, call every registered function, giving them a chance to do something

Proactive behavior in arcade games

Could adapt the previous approach

But it’s not great:

- in arcade games: no waiting for player input
- therefore: no notion of a round
  (or notion of round is not so useful)
  (thousand/million rounds per second)

- not so many things happening (30-40?)
- actions happening at human scale (seconds)

Event queues

Event queues record events rather than functions that perform something at every round

Event: a specific action happening at a specific point in the future

E.g.

in 1 time unit, move first baddie

in 4 time unit, fill hole

Event queues

Recurring events can be implemented by having an event record the next event

Event queue implementation intuition:

- keep events ordered by when they should be triggered (trigger time)
- Every time unit: check if there’s an event ready for triggering — if so, trigger it
- Every time unit: decrease trigger times for all events by 1

Event Queue data structure

Initially empty

Operations:

`enqueue (when,obj)`

- adds `obj` to the queue to be triggered in `when` t.u.
- `obj.event` will be called when triggered
  - arguments of the call up for debate

`dequeue_if_ready ()`

- triggers events with trigger time = 0
- remove them from queue
- decreases all the trigger times

High-level structure

```
def main ():
    create window
    initialize level
    initialize player
    initialize baddies

    while not player at exit :
        check key pressed
        if move key, process player move
        baddies move
```

High-level structure

```
def main ():
    create window
    initialize level
    initialize player
    initialize baddies   # and add them to event queue

    while not player at exit :
        check key pressed
        if move key, process player move
        ~~baddies move~~   # really, process event queue
```

High-level structure

```
def main ():
    cr
    in
    in
    in

    wh

        if move        , process player move
        baddies move   # really, process event queue
```

To:

(1) control rate of play across different machines

(2) pin down what time units are used

```
time.sleep (0.01)
```

(A time unit is 0.01s)

Example: baddie behavior

```
class Baddie (Character):
    …

    def event (self,q):
        if same position as the player, game over
        compute move to make to get to player (as dx, dy)
        self.move(dx,dy)
        # enqueue next event for the baddie, another move
        q.enqueue(BADDIE_DELAY,self)
```

At beginning of game, enqueue the baddie in the event queue.