COMP 9 / EN47 - Exploring Computer Science

Lecture 9: Object-Oriented Programming and Agent-Based Simulation

March 15, 2011

1 Introduction

Object-oriented programming is a way to *abstract* ideas away from the details of how they are implemented.

So far, you have used various objects and classes (`String`, `Array`, `Hash`, `Integer`, `Float`, `NilClass`, `Bio::FastaFormat`, `Bio::Blast::Report::Hit`) and called methods upon them. You have also created your own class to represent a pet of some sort, and written a game that calls methods on this class (such as `feed`, `put_to_bed`, `walk`).

However, objects can send methods to each other! Imagine a family, or litter, of virtual pets. How might we model them fighting over a toy? They might send `growl` messages to each other, or try to `steal` a ball!

Beyond games, we might simulate complex behaviors among entities and then *measure* their outcomes.

2 Traffic simulator

Imagine you're a city planner, trying to figure out how to time traffic lights. A car simulation — not just of one car, but of a whole bunch of commuters — might be helpful.

```
class Car
    attr_accessor :position, :crashed, :car_number, :speed
    def initialize(car_number, accel)
        @car_number = car_number
        @accel = accel
        @position = 0.0
        @speed = 0.0
        @crashed = false
```

```
    end

    def accelerate(time_step)
        @speed += @accel * time_step
    end

    def decelerate(time_step)
        decel = 1.0 # assume braking at 1 G
        @speed -= decel * time_step
        @speed = 0.0 if @speed < 0
    end

    def advance(time_step)
        if not @crashed
            @position = @position + @speed * time_step
            check_for_crash
        end
    end

    def look_ahead(cars, time_step)
        if cars.any?{|car| (car.position - self.position).abs < 10 }
            decelerate(time_step)
        else
            accelerate(time_step)
        end
    end

    def hit!
        @crashed = true
        @speed = 0.0
    end

    def check_for_crash
        if cars.any?{|car|
                (car.position - self.position).abs <= 0 and
                car.car_number != car_number
                }
            car.hit!
            self.hit!
        end
    end

end

interval = 1 # second
num_cars = 0
```

```
cars = []
while num_cars < 1000 do
    cars << car.new(num_cars, rand(10))
    num_cars += 1

    cars.each do |car|
        car.look_ahead(cars, interval)
    end

    cars.each do |car|
        car.advance(interval)
    end
end

cars.each do |car|
    puts "Car #{car.car_number}: Pos: #{car.position}, Spd: #{car.speed} Crash?: #{car.crash
end

average_speed = cars.inject(0.0){|sum, car| sum += car.speed} / cars.length
```