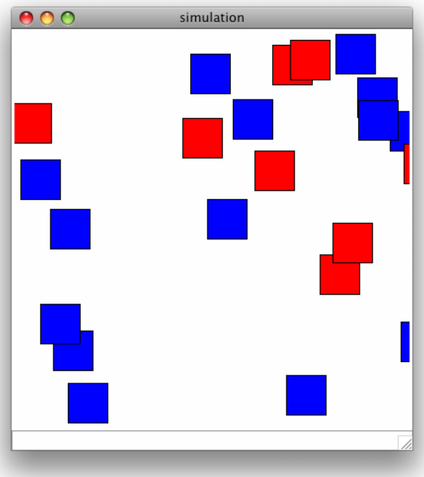
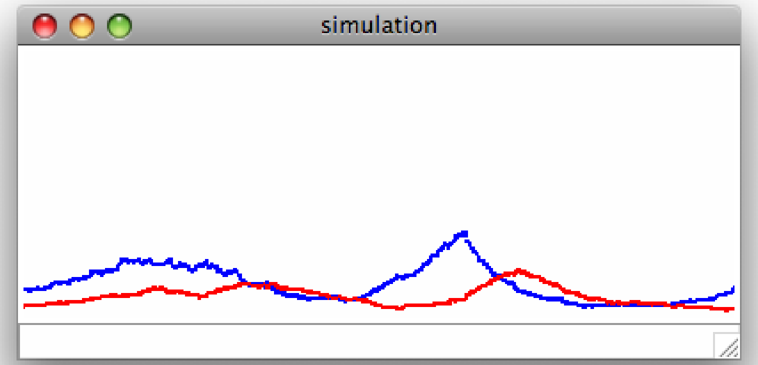
CMSC 143: Introduction to Object-Oriented Programming with Robots

Lab 11: Predator-Prey Simulation
Due April 21, 2010

Predator-prey simulations are used to understand how populations of animals interact. In this particular simulation, there will be two types of animals: Rabbits (prey) and Wolves (predators). Both animals run around randomly and with some probability reproduce every timestep. The wolves eat nearby rabbits, removing rabbits from the population. The wolves gain energy from eating the rabbits, but will die if their energy drops to zero. Wolves are born with 50 energy units and expend a unit of energy each timestep. Wolves are less likely to reproduce if their energy is low.

The `Animal` parent class and the base simulation are provided, your task is to implement the `Rabbit` and `Wolf` child clasess. You **should not** modify the main simulation loop. Next, you should run experiments with your simulation. Gather data about how the populations change as you vary one of the parameters: initial population size, speed, movement strategy, reproduction rate. Use `GraphWin()`[1] to plot the data and include the graph along with a brief reflection in your lab report. Your lab report should also include class diagrams for the `Animal`, `Rabbit` and `Wolf` classes.





Learning Objectives

◦ Apply Object-Oriented Design ◦ Use Inheritance ◦ Create Simulations

Deliverables

1. `cmsc143_lab11_LASTNAME_FIRSTNAME.pdf` – Your class diagrams and simulation results.
2. `cmsc143_lab11_LASTNAME_FIRSTNAME.py` – Your program.

[1]Use the printscrn button on your keyboard to take a screenshot of your window and then paste that into your report.

```
class Animal(object):
    animals = []
    SIZE = 40
    SENSING_RANGE = 30

    def __init__(self, win):
        ''' Create a new Anima'''
        self.win = win
        self.vx = 4
        self.vy = 4
        self.reproduction_rate = 0.02
        x = random.uniform(0, self.win.getWidth())
        y = random.uniform(0, self.win.getHeight())
        self.appearance = Rectangle(Point(x, y),
                                    Point(x + Animal.SIZE, y + Animal.SIZE))
        self.appearance.draw(self.win)
        Animal.animals.append(self)

    def eat(self):
        pass

    def reproduce(self):
        if random.random() < self.reproduction_rate:
            r = Animal(self.win)

    def die(self):
        '''remove this animal from the population'''
        self.appearance.undraw()
        Animal.animals.remove(self)

    def takeAStep(self):
        ''' move the animal for one timestep'''
        dx = random.uniform(-self.vx, self.vx)
        dy = random.uniform(-self.vy, self.vy)
        self.appearance.move(dx,dy)
        self.keepInWindow()

    def distance(self, other):
        ''' find the distance between myself and the other animal'''
        myx = self.appearance.getP1().getX()
        myy = self.appearance.getP1().getY()
        theirx = other.appearance.getP1().getX()
        theiry = other.appearance.getP1().getY()
        return ((myx - theirx)**2 + (myy - theiry)**2)**0.5
```

```
def nearbyAnimals(self):
    ''' find all the nearby animals within sensing range'''
    nearby = []
    for a in Animal.animals:
        if self.distance(a) < self.SENSING_RANGE and self != a:
            nearby.append(a)
    return nearby

def closestNeighbor(self, t):
    ''' find the closest animal of type t'''
    closestDistance = 100000000
    neighbor = None
    for a in self.nearbyAnimals():
        if isinstance(a, t) and self.distance(a) < closestDistance:
            neighbor = a
            closestDistance = self.distance(a)
    return neighbor

def keepInWindow(self):
    ''' keep the animal in the window'''
    x = self.appearance.getP1().getX()
    y = self.appearance.getP1().getY()
    if x >= self.win.getWidth():
        self.appearance.move(self.win.getWidth() - x, 0)
    elif x < 0:
        self.appearance.move(-x, 0)
    if y >= self.win.getHeight():
        self.appearance.move(0, self.win.getHeight() - y)
    elif y < 0:
        self.appearance.move(0, -y)

def checkCollision(self, a):
    '''returns True if self collides with a'''
    p1 = a.appearance.getP1()
    p2 = a.appearance.getP2()
    myp1 = self.appearance.getP1()
    myp2 = self.appearance.getP2()
    return myp1.getX() < p2.x and myp2.getX() > p1.x
           and myp1.getY() < p2.y and myp2.getY() > p1.y

def __str__(self):
    x = self.appearance.getP1().getX()
    y = self.appearance.getP1().getX()
    return "Animal at (" + str(x) + "," + str(y) + ")"
```

```
if __name__ == "__main__":

    random.seed(1)

    win = GraphWin("predator-prey simulation", 400, 400)
    win.setBackground("white")

    gwin = GraphWin("population size", 800, 150)
    gwin.setCoords(0, 0, 2000, 400)

    for i in range(32):
        r = Rabbit(win)

    for i in range(5):
        w = Wolf(win)

    t = 0
    while True:
        for a in Animal.animals:
            a.takeAStep()

        pr = Point(t, len(Rabbit.rabbits))
        pr.setFill("blue")
        pr.draw(gwin)
        pw = Point(t, len(Wolf.wolves))
        pw.setFill("red")
        pw.draw(gwin)
        t = t + 1
        if len(Rabbit.rabbits) == 0 or len(Wolf.wolves) == 0:
            print "Simulation Done"
            break
```