Continuous Simulation (Wolf/Deer Populations)

CS1316: Representing Structure and Behavior

The main run() method

```java
public void run()
{
    World w = new World();
    w.setAutoRepaint(false);
    // Start the lists
    wolves = new AgentNode();
    deer = new AgentNode();
    // create some deer
    int numDeer = 20;
    for (int i = 0; i < numDeer; i++)
    {
        deer.add(new AgentNode(new Deer(w, this)));
    }
    // create some wolves
    int numWolves = 5;
    for (int i = 0; i < numWolves; i++)
    {
        wolves.add(new AgentNode(new Wolf(w, this)));
    }
    // declare a wolf and deer
    Wolf currentWolf = null;
    Deer currentDeer = null;
    AgentNode currentNode = null;
    // loop for a set number of timesteps (50 here)
    for (int t = 0; t < 50; t++)
    {
        // loop through all the wolves
        currentNode = (AgentNode) wolves.getNext();
        while (currentNode != null)
        {
            currentWolf = (Wolf) currentNode.getAgent();
            currentWolf.act();
            currentNode = (AgentNode) currentNode.getNext();
        }
    }
}
```

Head and Rest

- Wolves and deer are AgentNodes...but the real content starts at getNext().
- We call this the head of the list.
  - It's a placeholder.
- We call the rest the rest or body of the list.
  - This makes it possible to remove a node, even if it's the first one in the list.

Start our simulation loop

```java
// declare a wolf and deer
Wolf currentWolf = null;
Deer currentDeer = null;
AgentNode currentNode = null;
// loop for a set number of timesteps (50 here)
for (int t = 0; t < 50; t++)
{
    // loop through all the wolves
    currentNode = (AgentNode) wolves.getNext();
    while (currentNode != null)
    {
        currentWolf = (Wolf) currentNode.getAgent();
        currentWolf.act();
        currentNode = (AgentNode) currentNode.getNext();
    }
}
```

Make some wolves

```java
// create some wolves
int numWolves = 5;
for (int i = 0; i < numWolves; i++)
{
    wolves.add(new AgentNode(new Wolf(w, this)));
}
```
Give the deer a chance to act

```java
// loop through all the deer
currentNode = (AgentNode) deer.getNext();
while (currentNode != null) {
    currentDeer = (Deer) currentNode.getAgent();
    currentDeer.act();
    currentNode = (AgentNode) currentNode.getNext();
}
```

Show us what happened

```java
// repaint the world to show the movement
w.repaint();

// Let's figure out where we stand...
System.out.println("Wolves left: " + wolves.getNext().count());
System.out.println("Deer left: " + deer.getNext().count());

// Wait for one second
//Thread.sleep(1000);
```

Implementing a Wolf

```java
import java.awt.Color;
import java.util.Random;
import java.util.Iterator;

/**
 * Class that represents a wolf. The wolf class tracks all the living wolves with a linked list.
 *
 * @author Barb Ericson
 * ericson@cc.gatech.edu
 */
public class Wolf extends Turtle {

    // fields
    private static final Color grey = new Color(153,153,153);
    protected static final double PROB_OF_STAY = 1.0/10;
    protected static final int maxSpeed = 60;

    // constructor
    public Wolf (ModelDisplay modelDisplayer, WolfDeerSimulation thisSim) {
        super(modelDisplayer.getWidth(), modelDisplayer.getHeight(), modelDisplayer);  
        init(thisSim);
    }

    // constructors
    public Wolf (int x, int y, ModelDisplay modelDisplayer, WolfDeerSimulation thisSim) {
        super(x, y, modelDisplayer);
        init(thisSim);
    }

    // public methods
    public Wolf(ModelDisplay modelDisplay, WolfDeerSimulation theSim) {
        super(modelDisplayer.getWidth(), modelDisplayer);  
        init(thisSim);
    }
```

More Wolf fields

```java
/**
 * protected static final int maxSpeed = 60;
 * maxSpeed should probably be all-caps (or did you want to make it variable? Do wolves get slower as they get hungry?)
 */
```

Constructors

Remember that a constructor must match its superclass, if you want to use super(). These are like the ones in Turtle.

What's a ModelDisplay? The abstract superclass of the World

Using a Random: PseudoRandom Number Generator

```java
public Wolf(ModelDisplay modelDisplay, WolfDeerSimulation theSim) {
    super(modelDisplay.getWidth(), modelDisplay);  
    init(thisSim);
}
```
### Initialize a Wolf

**methods**

```java
public void init(WolfDeerSimulation thisSim) {
  // set the color of this wolf
  setColor(grey);
  // turn some random direction
  this.turn(randNumGen.nextInt(360));
  // set my simulation
  mySim = thisSim;
}
```

### Is there a Deer to eat?

**getClosest**

```java
public AgentNode getClosest(double distance, AgentNode list) {
  // get the head of the deer linked list
  AgentNode head = list;
  AgentNode curr = head;
  AgentNode closest = null;
  Deer thisDeer;
  double closestDistance = 0;
  double currDistance = 0;
  // loop through the linked list looking for the closest deer
  while (curr != null) {
    thisDeer = (Deer) curr.getAgent();
    currDistance = thisDeer.getDistance(this.getXPos(), this.getYPos());
    if (currDistance < distance) {
      if (closest == null || currDistance < closestDistance) {
        closest = curr;
        closestDistance = currDistance;
      }
    }
    curr = (AgentNode) curr.getNext();
  } return closest;
```

### Modeling what a Wolf does

**act**

```java
public void act() {
  // get the closest deer within some specified distance
  AgentNode closeDeer = getClosest(30, mySim.getDeer().getNext());
  if (closeDeer != null) {
    Deer thisDeer = (Deer) closeDeer.getAgent();
    this.moveTo(thisDeer.getXPos(), thisDeer.getYPos());
    thisDeer.die();
  } else {
    // if the random number is > prob of NOT turning then turn
    if (randNumGen.nextFloat() > PROB_OF_STAY) {
      this.turn(randNumGen.nextInt(360));
    } // go forward some random amount
    forward(randNumGen.nextInt(maxSpeed));
  }
```

### If can’t eat, then move

```java
else {
  // if the random number is > prob of NOT turning then turn
  if (randNumGen.nextFloat() > PROB_OF_STAY) {
    this.turn(randNumGen.nextInt(360));
  } // go forward some random amount
  forward(randNumGen.nextInt(maxSpeed));
}
```

### Deer

```java
import java.awt.Color;
import java.util.Random;

/**
 * Class that represents a deer. The deer class
 * tracks all living deer with a linked list.
 * @author Barb Ericson
 * ericson@cc.gatech.edu
 */
public class Deer extends Turtle {

  // fields

  /** class constant for the color */
  private static final Color brown = new Color(116, 64, 35);
  /** class constant for probability of NOT turning */
  private static final double PROB_OF_STAY = 1.0/5;
  /** random number generator */
  private static Random randNumGen = new Random();
  /** the simulation I'm in */
  private WolfDeerSimulation mySim;

  // constructor

  Deer() {
    // import java.awt.Color;
    // import java.util.Random;
    // set the color of this deer
    setColor(brown);
    // set my simulation
    mySim = null;
  }
```

### Deer fields (instance variables)

```java
/** class constant for top speed (max num steps can move in a timestep) */
private static final int maxSpeed = 50;
/** random number generator */
private static Random randNumGen = new Random();
/** the simulation I'm in */
private WolfDeerSimulation mySim;
```

### Why getClosest? 

Because we need the body of the list, and that’s after the head.
### Deer Constructors

/* Constructor that takes the model display (the original position will be randomly assigned) */
public Deer (ModelDisplay modelDisplayer, WolfDeerSimulation thisSim)
{
    super(randNumGen.nextInt(modelDisplayer.getWidth()),
          randNumGen.nextInt(modelDisplayer.getHeight()),
          modelDisplayer);
    init(thisSim);
}

/* Constructor that takes the x and y and a model display to draw it on */
public Deer (int x, int y, ModelDisplay modelDisplayer,
            WolfDeerSimulation thisSim)
{
    // let the parent constructor handle it
    super(x, y, modelDisplayer);
    init(thisSim);
}

### What Deer Do

/* Method to act during a time step */
act()
{
    // if the random number is > prob of NOT turning then turn
    if (randNumGen.nextFloat() > PROB_OF_STAY)
    {
        this.turn(randNumGen.nextInt(360));
    }
    // go forward some random amount
    forward(randNumGen.nextInt(maxSpeed));
}

### When Deer Die

/* Method that handles when a deer dies */
die()
{
    // Leave a mark on the world where I died...
    this.setBodyColor(Color.red);
    // Remove me from the "live" list
    mySim.getDeer().remove(this);
    // ask the model display to remove this
    getModelDisplay().remove(this);
    System.out.println("<SIGH!> A deer died...");
}

### AgentNodes

- AgentNodes contain Turtles
  - That’s aggregation
- It’s a subclass of LLNode
  - It’s a specialization of LLNode

### AgentNode implementation

/* Class to implement a linked list of Turtle-like characters. */
public class AgentNode extends LLNode {
    /* This is the Turtle being held */
    private Turtle myTurtle;
AgentNode constructors

/** Two constructors: One for creating the head of the list */
 public AgentNode() {super();
/**
 * One constructor for creating a node with an agent
 */
 public AgentNode(Turtle agent){
 super();
 this.setAgent(agent);
 }

AgentNode getter/setter

/**
 * Setter for the turtle
 */
 public void setAgent(Turtle agent){
 myTurtle = agent;
 }
 /**
 * Getter for the turtle
 */
 public Turtle getAgent(){return myTurtle;}

AgentNode: Remove node where Turtle is found

/* Remove the node where this turtle is found. */
 public void remove(Turtle myTurtle) {
 // Assume we're calling on the head
 AgentNode head = this;
 AgentNode current = (AgentNode) this.getNext();
 while (current != null) {
 if (current.getAgent() == myTurtle) {
 // If found the turtle, remove that node
 head.remove(current);
 }
 current = (AgentNode) current.getNext();
 }
}

Think about it...

- What if AgentNodes contained Objects?
  - Object is a class that is the superclass of all classes (even if not explicitly extended).
  - AgentNodes that contain Objects could be general linked lists that contain anything
    - Just cast things as you need them as you pull them out.

Back to the simulation:
What might we change?

- Wolves that aren’t always hungry?
- Having wolves that chase deer?
  - Have deer run from wolves?
- And how do we look at the results?

Creating a Hungry Wolf

/* A class that extends the Wolf to have a Hunger level. */
 public class HungryWolf extends Wolf {
 /**
 * Number of cycles before I'll eat again
 */
 private int satisfied;
 /** class constant for number of turns before hungry */
 private static final int MAX_SATISFIED = 3;
Need to match

```java
public HungryWolf (ModelDisplay modelDisplayer, WolfDeerSimulation mySim)
{
    super(modelDisplayer, mySim);
}
```
New constants for Deer

/* class constant for probability of NOT turning */
private static final double PROB_OF_STAY = 1.0/5;

/* class constant for how far deer can smell */
private static final double SMELL_RANGE = 50;

/* class constant for top speed (max num steps can move in a timestep) */
private static final int maxSpeed = 30;

Deer new act()

Does this match the English description we had a few slides back?

Think about this in terms of the values that can be changed and their relative values.

How Wolf's smell deer

/* Method to act during a time step */
public void act()
{
  if (get the closest wolf within the smell range
  AgentNode closeWolf = getClosest(SMELL_RANGE, mySim.getWolves().getNextAgentNode());
  if (closeWolf != null)
  { Wolf closestWolf = (Wolf) closeWolf.getAgent();
    this.turnToFace(closeWolf);
    if (randomNumGen.nextFloat() > PROB_OF_STAY) // otherwise turn
      this.turn(180);
  } else // otherwise, wander aimlessly
  { if (randomNumGen.nextFloat() > PROB_OF_STAY)
      this.turn(randNumGen.nextInt(360));
    else // otherwise turn
      this.turn(180);
  } // end else
}

Wolf Constants

/* class constant for probability of NOT turning */
protected static final double PROB_OF_STAY = 1.0/10;

/* class constant for top speed (max num steps can move in a timestep) */
protected static final double maxSpeed = 40;

/* class constant for how far wolf can smell */
protected static final double SMELL_RANGE = 50;

/* class constant for how close before wolf can attack */
protected static final double ATTACK_RANGE = 30;

The rest of normal Wolf actions

if (get the closest deer within the attack distance
  closeDeer = getClosest(ATTACK_RANGE, (AgentNode) mySim.getDeer().getNextAgentNode());
  if (closeDeer != null)
  { Deer closestDeer = (Deer) closeDeer.getAgent();
    if (randomNumGen.nextFloat() > PROB_OF_STAY)
      this.turn(randNumGen.nextInt(360));
    else // otherwise turn
      this.turn(randNumGen.nextInt(360));
  } // end if
  else // otherwise wander aimlessly
  { if (randomNumGen.nextFloat() > PROB_OF_STAY)
      this.turn(randNumGen.nextInt(360));
    else // otherwise turn
      this.turn(randNumGen.nextInt(360));
  } // end else
} // end if
else // otherwise, wander aimlessly
{ if (randomNumGen.nextFloat() > PROB_OF_STAY)
    this.turn(randNumGen.nextInt(360));
  else // otherwise turn
    this.turn(randNumGen.nextInt(360));
Changes to WolfDeerSimulation...NOTHING!

- We have the same interface as we used to have, so nothing changes in WolfDeerSimulation.
- Very powerful idea:
  - If changes to a class keep the interface the same, then all users of the class don't have to change at all.

Running the new simulation

Welcome to DrJava.
> WolfDeerSimulation wds = new WolfDeerSimulation();
> wds.openFile("D:/cs1316/wds-chase.txt");
> wds.run();

Explorations

- What does the relative speed of Deer and Wolves matter?
  - Does it matter if Deer go faster? Wolves?
- What if Deer and Wolves can smell farther away?
  - What if one can smell better than the other?
- What's the effect of having more Deer or more Wolves?
- What if HungryWolves could starve (say at -10 satisfaction)? Do more deer live?

Doing More Simulations

- How much code would be in common in every simulation we'd build?
  - We already have lots of duplication, e.g., getClosest.
- Goal: Can we make an Agent/Actor class and Simulation class that we'd subclass with very little additional code to create new simulations?