Discrete Event Simulation

CS1316: Representing Structure and Behavior

Story

- Discrete event simulation
  - Simulation time != real time
- Key ideas:
  - A Queue
    - A Queue is a queue, no matter how implemented.
  - Different kinds of random
  - Straightening time
    - Inserting it into the right place
    - Sorting it afterwards
- Building a discrete event simulation
  - Graphics as the representation, not the real thing: The Model and the View

Finally: A Discrete Event Simulation

- Now, we can assemble queues, different kinds of random, and a sorted EventQueue to create a discrete event simulation.

Running a DESimulation

```
Welcome to DrJava.
> FactorySimulation fs = new
  FactorySimulation();
> fs.openFrames("D:/temp/");
> fs.run(25.0)
```

The detail tells the story

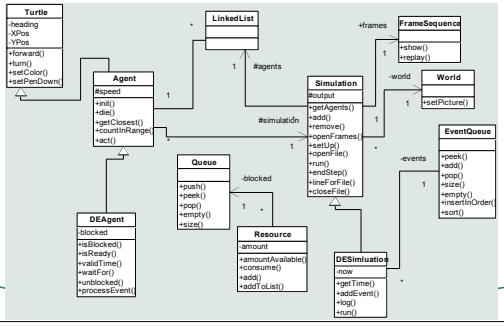
```
Time:      1.7078547183397625  Distributor: 0          Arrived at warehouse
Time:      1.7078547183397625  Distributor: 0          is blocking
>>> Timestep: 1
Time:      1.727166341118611   Distributor: 3          Arrived at warehouse
Time:      1.727166341118611   Distributor: 3          is blocking
>>> Timestep: 1
Time:      1.8778754913001443  Distributor: 4          Arrived at warehouse
Time:      1.8778754913001443  Distributor: 4          is blocking
>>> Timestep: 1
Time:      1.889475045031698   Distributor: 2          Arrived at warehouse
Time:      1.889475045031698   Distributor: 2          is blocking
>>> Timestep: 1
Time:      3.064560375192933   Distributor: 1          Arrived at warehouse
Time:      3.064560375192933   Distributor: 1          is blocking
>>> Timestep: 3
Time:      3.444420374970288   Truck: 2      Arrived at warehouse with load    13
Time:      3.444420374970288   Distributor: 0        unblocked!
Time:      3.444420374970288   Distributor: 0          Gathered product for orders of    11
>>> Timestep: 3
Time:      3.8869697922832698  Truck: 0      Arrived at warehouse with load    18
Time:      3.8869697922832698  Distributor: 3        unblocked!
Time:      3.8869697922832698  Distributor: 3          Gathered product for orders of    12
>>> Timestep: 3
```

Notice that time 2 never occurs!

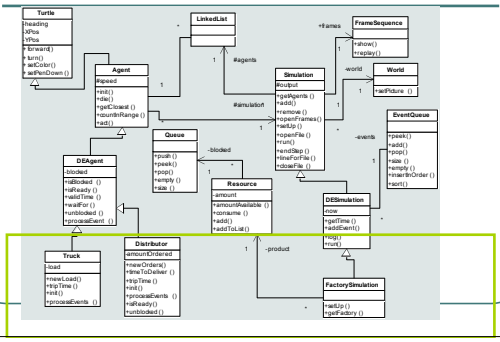
What questions we can answer

- How long do distributors wait?
  - Subtract the time that they unblock from the time that they block
- How much product sits in the warehouse?
  - At each time a distributor leaves, figure out how much is left in the warehouse.
- How long does the line get at the warehouse?
  - At each block, count the size of the queue.
- Can we move more product by having more distributors or more trucks?
  - Try it!

How DESimulation works



FactorySimulation: Extend a few classes



DESimulation: Sets the Stage

- DESimulation calls **setUp** to create agents and schedule the first events.
- It provides **log** for writing things out to the console and a text file.
- When it **run()'s**, it processes each event in the event queue and tells the corresponding agent to process a particular message.

What a DESimulation does:

```
// While we're not yet at the stop time,
   // and there are more events to process
   while ((now < stopTime) && (!events.empty())) {
     topEvent = events.pop();

     // Whatever event is next, that time is now
     now = topEvent.getTime();
     // Let the agent now that its event has occurred
     topAgent = topEvent.getAgent();
     topAgent.processEvent(topEvent.getMessage());

     // repaint the world to show the movement
     // IF there is a world
     if (world != null) {
       world.repaint();}

     // Do the end of step processing
     this.endStep((int) now);
   }
```

As long as there are events in the queue, and we're not at the stopTime:

Grab an event.

Make it's time "now"

Process the event.

What's an Event (SimEvent)?

```
/**
 * SimulationEvent (SimEvent) -- an event that occurs in a simulation,
 * like a truck arriving at a factory, or a salesperson leaving the
 * market
 **/
public class SimEvent{
  /// Fields ///
  /** When does this event occur? */
  public double time;

  /** To whom does it occur? Who should be informed when it occurred? */
  public DEAgent whom;

  /** What is the event?  We'll use integers to represent the meaning
   * of the event -- the "message" of the event.
   * Each agent will know the meaning of the integer for themselves.
   **/
  public int message;
```

It's a time, an Agent, and an integer that the Agent will understand as a message

DEAgent: Process events, block if needed

- DEAgents define the constants for messages: What will be the main events for this agent?
- If the agent needs a resource, it asks to see if it's available, and if not, it blocks itself.
- It will be told to unblock when it's ready.
- ***Agents are responsible for scheduling their OWN next event!***

An Example: A Truck

```
/**
 * Truck -- delivers product from Factory
 * to Warehouse.
 **/
public class Truck extends DEAgent {

  /////// Constants for Messages
  public static final int FACTORY_ARRIVE = 0;
  public static final int WAREHOUSE_ARRIVE = 1;

  ////// Fields /////
  /**
   * Amount of product being carried
   **/
  public int load;
```

How Trucks start

```
/**
  * Set up the truck
  * Start out at the factory
  **/
  public void init(Simulation thisSim){
    // Do the default init
    super.init(thisSim);
    this.setPenDown(false); // Pen up
    this.setBodyColor(Color.green); // Let green deliver!

    // Show the truck at the factory
    this.moveTo(30,350);
    // Load up at the factory, and set off for the warehouse
    load = this.newLoad();
    ((DESimulation) thisSim).addEvent(
            new SimEvent(this,tripTime(),WAREHOUSE_ARRIVE));
  }
```

The truck gets a load, then schedules itself to arrive at the Warehouse.

tripTime() uses the normal distribution

```
/** A trip distance averages 3 days */
  public double tripTime(){
    double delay = randNumGen.nextGaussian()+3;
    if (delay < 1)
      // Must take at least one day
    {return 1.0+((DESimulation) simulation).getTime();}
    else {return delay+((DESimulation) simulation).getTime();}
  }
```

newLoad() uses uniform

```
/** A new load is between 10 and 20 on a
   uniform distribution */
  public int newLoad(){
    return 10+randNumGen.nextInt(11);
  }
```

How a Truck processes Events

```
/**
  * Process an event.
  * Default is to do nothing with it.
  **/
 public void processEvent(int message){
   switch(message){
     case FACTORY_ARRIVE:
       // Show the truck at the factory
       ((DESimulation) simulation).log(this.getName()+"\t Arrived at factory");
       this.moveTo(30,350);
       // Load up at the factory, and set off for the warehouse
       load = this.newLoad();
       ((DESimulation) simulation).addEvent(
            new SimEvent(this,tripTime(),WAREHOUSE_ARRIVE));
       break;
```

Truck Arriving at the Warehouse

```
case WAREHOUSE_ARRIVE:
      // Show the truck at the warehouse
      ((DESimulation) simulation).log(this.getName()+"\t Arrived at
   warehouse with load \t"+load);
      this.moveTo(50,50);
      // Unload product -- takes zero time (unrealistic!)
      ((FactorySimulation) simulation).getProduct().add(load);
      load = 0;
      // Head back to factory
      ((DESimulation) simulation).addEvent(
           new SimEvent(this,tripTime(),FACTORY_ARRIVE));
      break;
```

What Resources do

- They keep track of what **amount** they have available (of whatever the resource is).
- They keep a *queue* of agents that are **blocked** on this resource.
- They can **add** to the resource, or have it **consume**(d).
  - When more resource comes in, the head of the queue gets asked if it's enough. If so, it can unblock.

How Resources alert agents

```
/**
  * Add more produced resource.
  * Is there enough to unblock the first
  * Agent in the Queue?
  **/
 public void add(int production) {
   amount = amount + production;

   if (!blocked.empty()){
     // Ask the next Agent in the queue if it can be unblocked
     DEAgent topOne = (DEAgent) blocked.peek();
     // Is it ready to run given this resource?
     if (topOne.isReady(this)) {
       // Remove it from the queue
       topOne = (DEAgent) blocked.pop();
       // And tell it it's unblocked
       topOne.unblocked(this);
     }
   }
 }
```

An example blocking agent: Distributor

```
/**
 * Distributor -- takes orders from Market to Warehouse,
 * fills them, and returns with product.
 **/
public class Distributor extends DEAgent {

  /////// Constants for Messages
  public static final int MARKET_ARRIVE = 0;
  public static final int MARKET_LEAVE = 1;
  public static final int WAREHOUSE_ARRIVE = 2;

  /** AmountOrdered so-far */
  int amountOrdered;
```

Distributors start in the Market

```
public void init(Simulation thisSim){
   //First, do the normal stuff
   super.init(thisSim);
   this.setPenDown(false); // Pen up
   this.setBodyColor(Color.blue); // Go Blue!

   // Show the distributor in the market
   this.moveTo(600,460); // At far right
   // Get the orders, and set off for the warehouse
   amountOrdered = this.newOrders();
   ((DESimulation) thisSim).addEvent(
           new SimEvent(this,tripTime(),WAREHOUSE_ARRIVE));
 }
```

Distributors have 3 events

- Arrive in Market: Schedule how long it'll take to deliver.
- Leave Market: Schedule arrive at the Factory
- Arrive at Warehouse: Is there enough product available? If not, block and wait for trucks to bring enough product.

Processing Distributor Events

```
/**
  * Process an event.
  * Default is to do nothing with it.
  **/
 public void processEvent(int message){
     switch(message){
       case MARKET_ARRIVE:
         // Show the distributor at the market, far left
         ((DESimulation) simulation).log(this.getName()+"\t Arrived at
    market");
         this.moveTo(210,460);
         // Schedule time to deliver
         ((DESimulation) simulation).addEvent(
             new SimEvent(this,timeToDeliver(),MARKET_LEAVE));
         break;
```

Leaving the Market

```
case MARKET_LEAVE:
      // Show the distributor at the market, far right
      ((DESimulation) simulation).log(this.getName()+"\t
  Leaving market");
      this.moveTo(600,460);
      // Get the orders, and set off for the warehouse
      amountOrdered = this.newOrders();
      ((DESimulation) simulation).addEvent(
         new
  SimEvent(this,tripTime(),WAREHOUSE_ARRIVE));
       break;
```

Arriving at the Warehouse

```
case WAREHOUSE_ARRIVE:
      // Show the distributor at the warehouse
      ((DESimulation) simulation).log(this.getName()+"\t Arrived at warehouse");
      this.moveTo(600,50);
      // Is there enough product available?
      Resource warehouseProduct = ((FactorySimulation) simulation).getProduct();
      if (warehouseProduct.amountAvailable() >= amountOrdered)
      {
        // Consume the resource for the orders
        warehouseProduct.consume(amountOrdered); // Zero time to load?
        ((DESimulation) simulation).log(this.getName()+"\t Gathered product for orders of
  \t"+amountOrdered);
        // Schedule myself to arrive at the Market
        ((DESimulation) simulation).addEvent(
             new SimEvent(this,tripTime(),MARKET_ARRIVE));
      }
      else {// We have to wait until more product arrives!
        ((DESimulation) simulation).log(this.getName()+"\t is blocking");
        waitFor(((FactorySimulation) simulation).getProduct());}
    break;
```

Is there enough product?

```
/** Are we ready to be unlocked? */
 public boolean isReady(Resource res) {
   // Is the amount in the factory more than our orders?
   return ((FactorySimulation)
   simulation).getProduct().amountAvailable() >=
   amountOrdered;}
```

If so, we'll be unblocked

```
/**
  * I've been unblocked!
  * @param resource the desired resource
  **/
 public void unblocked(Resource resource){
   super.unblocked(resource);

   // Consume the resource for the orders
   ((DESimulation) simulation).log(this.getName()+"\t unblocked!");
   resource.consume(amountOrdered); // Zero time to load?
   ((DESimulation) simulation).log(this.getName()+"\t Gathered product for
     orders of \t"+amountOrdered);
   // Schedule myself to arrive at the Market
   ((DESimulation) simulation).addEvent(
          new SimEvent(this,tripTime(),MARKET_ARRIVE));
 }
```

The Overall Factory Simulation

```
/**
 * FactorySimulation -- set up the whole simulation,
 * including creation of the Trucks and Distributors.
 **/
public class FactorySimulation extends DESimulation {

  private Resource product;

  /**
   * Accessor for factory
   **/
  public FactoryProduct getFactory(){return factory;}
```

Setting up the Factory Simulation

```
public void setUp(){
  // Let the world be setup
  super.setUp();
  // Give the world a reasonable background
  FileChooser.setMediaPath("D:/cs1316/MediaSources/");
  world.setPicture(new Picture(

    FileChooser.getMediaPath("EconomyBackground.jpg"))
    );

  // Create a warehouse resource
 product = new Resource(); //Track product

 // Create three trucks
 Truck myTruck = null;
 for (int i=0; i<3; i++){
   myTruck = new Truck(world,this);
   myTruck.setName("Truck: "+i);}

 // Create five Distributors
 Distributor sales = null;
 for (int i=0; i<5; i++){
   sales = new Distributor(world,this);
   sales.setName("Distributor: "+i);}
}
```

The Master Data Structure List: We use *almost everything* here!

- Queues: For storing the agents waiting in line.
- EventQueues: For storing the events scheduled to occur.
- LinkedList: For storing all the agents.