Appendix B

Contents

1 EvolutionSimulator Source Code

1.1 Main Application

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import HelperFunctions.ModuleLoader;
import SimComponents.AgentObject;
import SimComponents.Module;
import SimComponents.SimObject;

public class EvolutionSimulator {
  final static String helpstring = "Commands available:" +
      "\nexit: exits the program" +
      "\nhelp: displays this help"+
      "\nsimstat: displays stats about sims runnning"+
      "\ngetoldest: displays oldest agents in running sims"+
      "\ngetbest: displays agent that has eaten the most energy in its
life";

  public static void main(String[] args) {
    int numthreads = 0;
    int simheight = 0;
    int simwidth = 0;
    double simmutation = 0;
    try//check args
    {
      numthreads = Integer.parseInt(args[0]);
      simwidth = Integer.parseInt(args[1]);
      simheight = Integer.parseInt(args[2]);
      simmutation = Double.parseDouble(args[3]);
      //System.out.println("Mutation chance: "+simmutation);
    }
    catch(Exception E)
    {
      System.out.println(E.toString());
      System.out.println("Usage: EvolutionSimulator <number_of_simulations>
<width_of_sim> <height_of_sim> <base_mutation_rate>");
      System.exit(1);
    }

    SimObject[] sims = new SimObject[numthreads];
    String moduleDirectory =
System.getProperty("user.dir")+"/bin/Modules/";  //C:\\simmods\\
    System.out.println("Loading modules from "+moduleDirectory);
    Class<Module>[] simModules = ModuleLoader.loadModules(moduleDirectory);
    System.out.println("Found "+simModules.length+" modules");
    System.out.println("Starting sims...");
    for(int i=0; i<numthreads; i++)
    {
      sims[i] = new SimObject(simheight,simwidth,simmutation,simModules);
      sims[i].initialize();
```

```
      sims[i].start();
    }

    //all sims created and initialized;
    //execute till exit
//    System.out.println("Type exit to end. Type help to see console
commands.");
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

    boolean stillalive = true;
    while(stillalive)
    {
      stillalive = false;
      for(int i=0; i<sims.length; i++)
      {
        if(sims[i].isAlive())
        {
          stillalive = true;
        }
        else
        {
          sims[i] = new
SimObject(simheight,simwidth,simmutation,simModules);
          sims[i].initialize();
          sims[i].start();
        }
      }
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {

      }
      String command = "";
      try {
        if(br.ready())
          command = br.readLine();
      } catch (IOException e) {
        //nothing
      }
      if (command.length() > 0)
      {
        if(command.equals("exit"))
        {
          break;
        }
        else if (command.equals("help"))
        {
          System.out.println(helpstring);
        }
        else if (command.equals("simstat"))
        {
          for(int j=0; j<sims.length; j++)
```

```
          {
            if(sims[j].isAlive())
            {
              long seconds = sims[j].getRunTime();
              long minutes = seconds / 60;
              seconds = seconds % 60;
              long hours = minutes / 60;
              minutes = minutes % 60;

              System.out.println(sims[j].ID+": Runtime:
"+hours+"h:"+minutes+"m:"+seconds+"s");
              System.out.println(sims[j].ID+": Population:
"+sims[j].agents.size());
              System.out.println(sims[j].ID+": Ticks:
"+sims[j].getTicks());
              System.out.println(sims[j].ID+": Ticks per second:
"+sims[j].getTicksPerSecond());
              System.out.println(sims[j].ID+": Average ticks per second:
"+sims[j].getAvgTicksPerSecond());
            }
          }
        }
        else if (command.equals("getoldest")) //getoldest and getbest ARE
NOT THREAD SAFE - wrapped in try's for now
        {
          try{
            for(int j=0; j<sims.length; j++)
            {
              if(sims[j].isAlive())
              {
                AgentObject a = new AgentObject();
                for(int k=0; k<sims[j].agents.size(); k++)
                {
                  if(a.getAge()<sims[j].agents.get(k).getAge())
                  {
                    a = sims[j].agents.get(k);
                  }
                }
                System.out.println(sims[j].ID+": Oldest Agent: "+a);
              }
            }
          }
          catch(Exception e)
          {
            e.printStackTrace();
          }


        }
        else if (command.equals("getbest"))
        {
          try{
            for(int j=0; j<sims.length; j++)
```

```
            {
              if(sims[j].isAlive())
              {
                AgentObject a = new AgentObject();
                double ascore = 0;
                for(int k=0; k<sims[j].agents.size(); k++)
                {
                  AgentObject b = sims[j].agents.get(k);
                  double bscore =
(b.getAte()*1.0)/((b.getAge()*1.0)/((b.getBred()+1)*1.0));

                  if(ascore < bscore)
                  {
                    a = b;
                    ascore = bscore;
                  }
                }                          System.out.println(sims[j].ID+":
Best Agent: "+a+"\nScore:"+ascore);
              }
            }
          }
          catch(Exception e)
          {
            e.printStackTrace();
          }
        }
        else
        {
          System.out.println("Unkown command: "+command);
        }
      }
    }

    for(int i=0; i<numthreads; i++)
    {
      sims[i].end();
    }
    while(stillalive)
    {
      stillalive = false;
      for(int i=0; i<sims.length; i++)
      {
        if(sims[i].isAlive())
        {
          stillalive = true;
        }
      }
    }
    System.out.println("Application ended");
    System.exit(0);
  }

}
```

1.2 SimComponents

1.2.1 AgentObject

```
package SimComponents;

import java.util.LinkedList;
import java.util.Random;
import java.io.Serializable;
import HelperFunctions.Node;

public class AgentObject implements Serializable {
  private static final long serialVersionUID = -6819599121262915208L;

  public long uniqueID;
  public int genus = 0;
  public int generation = 0;
  public double mutationrate = 0;
  private int lastbred = 0;
  public int dnahash = 0;
  public int x=0;
  public int y=0;
  public int energy = 0;
  private int age=0;
  private long bred = 0;
  private long ate = 0;
  public String dna = "";
  public int dnapos = 0;
  private char state = 'X';
  public int idMarker = 0; //self filled, for identification of own species
- no point using getters/setters for this
  private char rotation = 'N'; //N, E, S, W
  private int seeDepth = 5;

  private LinkedList<Integer> stack;
  private Node ancestry;

  public AgentObject()
  {
    this.x = 0;
    this.y = 0;
    this.dna = "";
    this.energy = 0;
    this.stack = new LinkedList<Integer>();
    this.dnahash = this.dna.hashCode();
    this.ancestry = new Node(this,null,null);
    Random random = new Random();
    uniqueID = random.nextLong();
  }

  public AgentObject(int x, int y, String dna, int energy, int genus,
double mutationrate)
  {
    this.x = x;
```

```
    this.y = y;
    this.dna = dna;
    this.energy = energy;
    this.stack = new LinkedList<Integer>();
    this.dnahash = this.dna.hashCode();
    this.genus = genus;
    this.mutationrate = mutationrate;
    this.ancestry = new Node(this,null,null);
    Random random = new Random();
    uniqueID = random.nextLong();
  }

  @SuppressWarnings("unchecked")
  public AgentObject(AgentObject a)
  {
    this.age = a.age;
    this.ate = a.ate;
    this.bred = a.bred;
    this.dna = a.dna;
    this.dnahash = a.dnahash;
    this.dnapos = a.dnapos;
    this.energy = a.energy;
    this.genus = a.genus;
    this.lastbred = a.lastbred;
    this.rotation = a.rotation;
    this.seeDepth = a.seeDepth;
    this.stack = (LinkedList<Integer>) a.stack.clone();
    this.state = a.state;
    this.x = a.x;
    this.y = a.y;
    this.mutationrate = a.mutationrate;
    this.generation = a.generation;
    this.ancestry = new
Node(this,a.ancestry.leftParent,a.ancestry.rightParent);
    //this.ancestry = (Node) a.ancestry.clone(); //clone
    this.uniqueID = a.uniqueID;
  }

  public void setParent(AgentObject parent, AgentObject secondParent)
  {
    if(secondParent == null)
    {
      this.ancestry = new Node(this, parent.ancestry ,null);
    }
    else
    {
      this.ancestry = new Node(this, parent.ancestry ,
secondParent.ancestry);
    }
  }

  public Node getAncestry()
  {
```

```
  return this.ancestry;
}

public boolean canBreed()
{
  return this.lastbred == 0;
}

public void resetBreedDelay()
{
  if(this.getState() != 'V')
  {
    this.lastbred = 50;
  }
  else
  {
    this.lastbred = 300;
  }
}

public char getState()
{
  return this.state;
}

public void setState(char c)
{
  this.state = c;
}

public char getRotation()
{
  return this.rotation;
}

public void setRotation(char r)
{
  if (r == 'N' ||r == 'E' ||r == 'S' ||r == 'W')
  {
    this.rotation = r;
  }

}



public void stackPush(int c)
{
  this.stack.addFirst(c);
  if(this.stack.size()>10)
  {
    this.stack.removeLast();
  }
```

```
  }

  public int stackPop()
  {
    if(stack.size()>0)
    {
      int tmp = stack.getFirst();
      stack.removeFirst();
      return tmp;
    }
    else
    {
      return 0;
    }
  }

  public int getSeeDepth()
  {
    return seeDepth;
  }

  public void incAge()
  {
  //  if(!this.ancestry.checkIntegrity()) //debug code
  //  {
  //    System.out.println("Ancestry failed integrity check!");
  //  }
    this.age++;
    if(this.lastbred > 0)
    {
      this.lastbred--;
    }
  }

  public int getAge()
  {
    return this.age;
  }

  public void recalcDNAHash()
  {
    this.dnahash = this.dna.hashCode();
  }

  public void jump(int jmploc)
  {
    if(jmploc < 0) //java's Math.abs() implementation is stupid.
Math.abs(Integer.MIN_VALUE) = Integer.MIN_VALUE. Who thought that was a
good idea?
    {
      if(jmploc == Integer.MIN_VALUE)
        jmploc = Integer.MAX_VALUE;
      else
```

```
        jmploc = jmploc * -1;
    }

    this.dnapos = jmploc % this.dna.length();
  }

  public void incBred()
  {
    this.bred ++;
  }

  public void consumedNrg(int nrg)
  {
    this.ate += nrg;
  }

  public long getBred()
  {
    return this.bred;
  }

  public long getAte()
  {
    return this.ate;
  }

  public void setSight(int dist)
  {
    dist = dist % 256;
    if(dist < 0) //java's Math.abs() implementation is stupid.
Math.abs(Integer.MIN_VALUE) = Integer.MIN_VALUE. Who thought that was a
good idea?
    {
      if(dist == Integer.MIN_VALUE)
        dist = Integer.MAX_VALUE;
      else
        dist = dist * -1;
    }
    this.seeDepth = dist;
  }

  public String toString()
  {
    String result = super.toString()+":{\n";

    result += "\tUID: "+uniqueID;
    result += "\n\tidMarker: "+idMarker;
    result += "\n\tx: "+x;
    result += "\n\ty: "+y;
    result += "\n\tenergy: "+energy;
    result += "\n\tdna: "+dna;
    result += "\n\tdnalength: "+dna.length();
    result += "\n\tdnapos: "+dnapos;
```

```
    result += "\n\trotation: "+rotation;
    result += "\n\tstate: "+state;
    result += "\n\tstack: "+stack;
    result += "\n\tage: "+age;
    result += "\n\tate: "+ate;
    result += "\n\toffspring: "+bred;
    result += "\n\tsight: "+seeDepth;
    result += "\n\tgenus:" +genus;
    result += "\n\tgeneration:" +generation;
    result += "\n\tmutationrate:" +mutationrate;
  //  result += "\n\tancestry:" +ancestry; //breaks things - basically
causes a full dump of the ancestry tree of every related agent. Not good.
    result +="\n}";
    return result;
  }
}
```

1.2.2 SimObject

```
package SimComponents;

import java.util.ArrayList;
import java.util.Random;
import java.io.Serializable;

public class SimObject extends Thread implements Serializable {

  private static final long serialVersionUID = 9031280818674869096L;
  //dynamic array to store agents (O(1) access & O(n) insert in worst case,
ie: needs to expand array beyond reserved mem);
  public ArrayList<AgentObject> agents = new ArrayList<AgentObject>();
  private ArrayList<AgentObject> toRemove = new ArrayList<AgentObject>();
  public transient AgentObject[][] positionGrid;
  public double mutationrate;
  public int height;
  public int width;
  public int ID;
  private int ticks;
  private Class<Module>[] moduleClasses;
  public transient Module[] modules; //transient because no need to
serialise the module instances. Classes are stored above anyways
  private transient float ticksPerSecond = 0;
  private transient long lastTickTime = 0;
  private transient long absoluteStartTime = 0;


  public SimObject(int height, int width, double mutationRate,
Class<Module>[] modules)
  {
    this.ticks = 0;
    this.width = width;
    this.height = height;
    this.mutationrate = mutationRate;
```

```
    positionGrid = new AgentObject[width+1][height+1];
    this.moduleClasses = modules;
    this.modules = new Module[modules.length];
    System.out.println("Simulation created...");
  }

  public void initialize()
  {
    Random mr = new Random();
    ID = mr.nextInt(Integer.MAX_VALUE); //between 0 & maxval
    System.out.println(ID+": Simulation initialzing...");
    //load modules
    for(int i=0; i<moduleClasses.length; i++)
    {
      try {
        modules[i] = moduleClasses[i].newInstance();
        System.out.println(ID+": Loading module:
"+modules[i].getClass().getName());
        modules[i].simInit(this);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }

    for(int i=0; i<agents.size(); i++)
    {
      positionGrid[agents.get(i).x][agents.get(i).y] = agents.get(i);
    }
  }

  public void end()
  {
    System.out.println(ID+": Simulation shutting down...");
    this.interrupt();
  }

  public void run()
  {
    int i=0;
    int j=0;
    int k=0;
    int exceptionsInARow = 0;
    System.out.println(ID+": Simulation started!");
    absoluteStartTime = System.nanoTime();
    while(!this.isInterrupted())
    {
      try
      {

        //  if(agents.size() == 0)
        //  {
        //    System.out.println("Errybody dead");
        //    this.interrupt();
```

```
        //  }
        long nanospertick = (System.nanoTime()-lastTickTime);
        if(nanospertick == 0) {nanospertick = 1; } //just in case,
otherwise div by 0
        ticksPerSecond = 1.0f/(nanospertick/1000000000.0f); //1 divided by
seconds per tick = ticks per second
        lastTickTime = System.nanoTime();
        //do methods for each sim tick
        ticks++;
        for(k=0; k<modules.length; k++)
        {
          modules[k].simTick(this);
        }

        for(i=0; i<agents.size(); i++)
        {
          AgentObject a = agents.get(i);

          //do agent tick methods
          for(j=0; j<modules.length; j++)
          {
            if(a.energy>=0) //modules could kill agent
            {
              modules[j].agentTick(this, a);
              if(a.energy <= 0)
              {
                //if module resulted in agent running out of energy, kill
it.
                killAgent(a);
              }

            }
            else
              break; //agent is dead, no point continuing to do stuff to it
          }
        }

        //agents processed, remove the dead
        //bring out your dead! bring out your dead!
        for(int p=0; p<toRemove.size(); p++)
        {
          agents.remove(toRemove.get(p));
        }
        toRemove.clear();
      }
      catch(Exception e)
      {
        exceptionsInARow++;
        if(k<modules.length) { System.out.println(ID+": simTick() in
"+modules[k].getClass().getName()+" caused failure"); }
        if(j<modules.length) { System.out.println(ID+": agentTick() in
"+modules[j].getClass().getName()+" caused failure"); }
```

```
        if(exceptionsInARow > 3)
        {
          System.out.println(ID+": Simulation died after
"+exceptionsInARow+" failures with error:"+e);
          System.out.println(ID+": Trying to unload modules cleanly.");
          //finalise modules when sim ends
          this.interrupt();
          for(i=0; i<modules.length; i++)
          {
            try{
              modules[i].simEnd(this);
            }
            catch(Exception me){
              System.out.println(ID+": Module
"+modules[i].getClass().getName()+" failed to unload with error: "+me);
            }
          }
        }
        e.printStackTrace();
      }
    }
    if(exceptionsInARow <= 3)
    {
      //finalise modules when sim ends
      for(i=0; i<modules.length; i++)
      {
        modules[i].simEnd(this);
      }
    }


  }

  public void killAgent(AgentObject a)
  {
    a.energy = 0;
    if(positionGrid[a.x][a.y]==a)
    {
      positionGrid[a.x][a.y] = null; //remove from potential collisions
    }
    //agents.remove(a);
    toRemove.add(a); //add to list of dead to remove
  }

  public int getTicks()
  {
    return this.ticks;
  }

  public float getTicksPerSecond()
  {
    return this.ticksPerSecond;
  }
```

```
  public float getAvgTicksPerSecond()
  {
    long nanospertick = (System.nanoTime()-absoluteStartTime)/this.ticks;
//nano seconds since start of sim/number of ticks
    if(nanospertick == 0) {nanospertick = 1; } //just in case, otherwise
div by 0
    return 1.0f/(nanospertick/1000000000.0f); //1 divided by average
seconds per tick = average ticks per second
  }
  public long getRunTime() //returns seconds since start
  {
    return (System.nanoTime()-absoluteStartTime) / 1000000000;
  }
}
```

1.2.3 Module

```
package SimComponents;

public abstract class Module {
  public void simInit(SimObject s)
  {

  }

  public void simTick(SimObject s)
  {

  }

  public void agentTick(SimObject s, AgentObject a)
  {

  }

  public void simEnd(SimObject s)
  {

  }
}
```

1.3 Modules

1.3.1 actionsModule

```
package Modules;
import java.util.Random;
import SimComponents.*;

public class actionsModule extends Module {
  int nrgEaten = 0;

  private void doMovement(char dir, AgentObject a, SimObject s)
  {
    if(dir != 'X')
    {
```

```
      int oldx = a.x;
      int oldy = a.y;
      int newy = oldy;
      int newx = oldx;

      switch(dir)
      {
      case 'N':
        newy--; break;
      case 'E':
        newx++; break;
      case 'S':
        newy++; break;
      case 'W':
        newx--; break;
      default: break;
      }

      //if you're going to move outside sim boundaries, don't move at all.
      if ((newx >= s.width) || (newx <= 0))
        newx = oldx;

      if ((newy >= s.height) || (newy <= 0))
        newy = oldy;

    //  newx = (s.width+newx) % s.width;
    //  newy = (s.height+newy) % s.height;
      if(s.positionGrid[newx][newy]==null)
      {
        s.positionGrid[oldx][oldy] = null;
        s.positionGrid[newx][newy] = a;
        a.x = newx;
        a.y = newy;
        a.energy = a.energy-10;
        //System.out.println("Moved from ("+oldx+","+oldy+") to
("+newx+","+newy+")");
      }
      else
      {
        if(doCollision(a,s.positionGrid[newx][newy],s))
        {
          s.positionGrid[oldx][oldy] = null;
          s.positionGrid[newx][newy] = a;
          a.x = newx;
          a.y = newy;
          a.energy = a.energy-10;
        }
      }
    }
  }

  private void breedAgents(AgentObject a, AgentObject b, SimObject s)
  {
```

```
    Random myrandom = new Random();
    if(b == null)
    {
      b = new AgentObject(); //create a placeholder empty agent with no dna
& no energy
    }

    //was 200
    int energyBarrier = 200;
    if(a.getState() == 'V' || b.getState()=='V')
    {
      energyBarrier = 1000;
    }
    if((a.energy+b.energy > energyBarrier) && (a.canBreed() &&
b.canBreed()))
    {
      int newx = (a.x+myrandom.nextInt(4)-2);
      int newy = (a.y+myrandom.nextInt(4)-2);

      boolean placed = false;
      int tries = 0;
      while((!placed) && (tries < 10)) //look for a place to put the new
agent
      {
        if(newx < a.x)
          newx--;
        else
          newx++;

        if(newy < a.y)
          newy--;
        else
          newy++;

        if ((newx >= s.width) || (newx <= 0))
          newx = (a.x+myrandom.nextInt(4)-2);

        if ((newy >= s.height) || (newy <= 0))
          newy = (a.y+myrandom.nextInt(4)-2);
        if((newx >= 0) && (newx <= s.width) && (newy >=0) && (newy
<=s.height))
        if(s.positionGrid[newx][newy] == null)
        {
          a.incBred();
          a.resetBreedDelay();
          b.resetBreedDelay();
          AgentObject c = new AgentObject();
          c.energy = a.energy / 2;
          c.energy += b.energy / 2;
          //if(c.energy > (a.energy + b.energy)/2) {
System.out.println("D:"); }
          a.energy = a.energy / 2;
          b.energy = b.energy /2;
```

```
          //if(c.energy > (a.energy + b.energy)) {
System.out.println("D:"); }
          if(b.mutationrate == 0)
          {
            c.mutationrate = a.mutationrate;
          }
          else
          {
            c.mutationrate = (a.mutationrate+b.mutationrate)/2;
          }

          int crossoverpoint = myrandom.nextInt(Math.min(a.dna.length(),
b.dna.length())+1);
          if(a.dna.length() < b.dna.length())
          {
            c.dna = a.dna.substring(0,crossoverpoint) +
b.dna.substring(crossoverpoint,b.dna.length());
          }
          else
          {
            c.dna = b.dna.substring(0,crossoverpoint) +
a.dna.substring(crossoverpoint,a.dna.length());
          }

          c.x = newx;
          c.y = newy;
          if(b.genus != 0)
          {
            c.genus = (a.genus+b.genus)/2;
          }
          else
          {
            c.genus = a.genus;
          }


          if(b.generation > a.generation)
          {
            c.generation = b.generation+1;
          }
          else
          {
            c.generation = a.generation+1;
          }

          if(b.mutationrate == 0)
          {
            b = null; //make sure we don't put placeholder agents into the
ancestry tree
          }

          c.setParent(a, b);
```

```
          s.agents.add(c);
          s.positionGrid[newx][newy] = c;
          placed = true;

        }

        tries++;
      }//while(!placed)
    }//if can breed
  }

  private boolean doCollision(AgentObject a, AgentObject b, SimObject s)
//returns boolean reflecting whether or not the move is possible
  {
    if(a == b) { return false; } //can't collide with yourself!

    if((a.getState()=='M') && (b.getState()=='M')) //both ready to mate
    {
      //mate
      breedAgents(a,b,s);
      return false;
    }
    else
    if((a.getState()=='E') && (b.getState()=='E'))
    {
      if(a.energy>=b.energy) //a stronger than b, a eats b
      {
        a.energy+=b.energy;
        nrgEaten += b.energy;
        s.killAgent(b);
        return true;
      }
      else  //b stronger than a, b eats a
      {
        b.energy+=a.energy;
        s.killAgent(a);
        return false;
      }
    }
    else if(a.getState()=='E')
    {
      a.energy += b.energy;
      nrgEaten += b.energy;
      s.killAgent(b);
      return true;
    }
    else if(b.getState()=='E')
    {
      b.energy += a.energy;
      s.killAgent(a);
      return false;
    }
    else //both aren't in mate state, neither in eat state
```

```
  {
    return false;
  }


}



private char getSeenState(AgentObject a, SimObject s)
{
  int modx = 0;
  int mody = 0;
  switch(a.getRotation())
  {
  case 'N': modx = 1; break;
  case 'E': mody = 1; break;
  case 'S': modx = -1; break;
  case 'W': mody = -1; break;
  }
  for(int i=0; i<a.getSeeDepth();i++)
  {
    int tmpx = (s.width+a.x+(i*modx))%s.width;
    int tmpy = (s.height+a.y+(i*mody))%s.height;
    if(s.positionGrid[tmpx][tmpy] != null)
    {
      return s.positionGrid[tmpx][tmpy].getState();
    }
  }
  return '\0';
}

private int getSeenMarkerID(AgentObject a, SimObject s)
{
  int modx = 0;
  int mody = 0;
  switch(a.getRotation())
  {
  case 'N': modx = 1; break;
  case 'E': mody = 1; break;
  case 'S': modx = -1; break;
  case 'W': mody = -1; break;
  }
  for(int i=0; i<a.getSeeDepth();i++)
  {
    int tmpx = (s.width+a.x+(i*modx))%s.width;
    int tmpy = (s.height+a.y+(i*mody))%s.height;
    if(s.positionGrid[tmpx][tmpy] != null)
    {
      return s.positionGrid[tmpx][tmpy].idMarker;
    }
  }
  return '\0';
```

```
  }

  private int getEyesight(AgentObject a, SimObject s)
  {
    int modx = 0;
    int mody = 0;
    switch(a.getRotation())
    {
    case 'N': modx = 1; break;
    case 'E': mody = 1; break;
    case 'S': modx = -1; break;
    case 'W': mody = -1; break;
    }
    for(int i=0; i<a.getSeeDepth();i++)
    {
      int tmpx = (s.width+a.x+(i*modx))%s.width;
      int tmpy = (s.height+a.y+(i*mody))%s.height;
      if(s.positionGrid[tmpx][tmpy] != null)
      {
        return i;
      }
    }
    return 0;
  }

  private void doDNATick(AgentObject a, SimObject s)
  {
    int commandsprocessed = 0;
    int maxcommandspertick = 50; //50
    char prevstate = a.getState();

    if(a.getState()=='V')
    {
      //You are a veg, so you get less DNA to play with
      maxcommandspertick = 10;
    }
    while((a.dna.length()>0) && (commandsprocessed<maxcommandspertick))
//has DNA to process
    {
      if(prevstate != a.getState())
      {
        if((a.getState() == 'V') || (prevstate == 'V'))
        {
          maxcommandspertick = -1;
          break;
        }
        prevstate = a.getState();
      }
      int tmpdnapos = 0;

      if(a.dnapos>=a.dna.length())
      {
        a.dnapos = 0;
```

```
      }

      char dnapiece = a.dna.charAt(a.dnapos);
      commandsprocessed++;
      switch(dnapiece)
      {
        case 'a': doMovement('N',a,s); break; //move north
        case 'b': doMovement('S',a,s); break; //move south
        case 'c': doMovement('W',a,s); break; //move west
        case 'd': doMovement('E',a,s); break; //move east
        case 'j': doMovement((char)a.stackPop(),a,s); break; //set movement
to stackPop
        case 'e': a.setRotation('N'); break;
        case 'f': a.setRotation('E'); break;
        case 'g': a.setRotation('S'); break;
        case 'h': a.setRotation('W'); break;
        case 'i': a.setRotation((char)a.stackPop()); break; //set rotation
to stackPop
        case 'k': breedAgents(a,null,s); break;  //split asexually
        case 'l': a.setState('M'); break;  //mate
        case 'm': a.setState('X'); break; //no action
        case 'n': a.setState('E'); break; //eat
        case 'o': a.setState('V'); break; //become a vegetable
        case 'p': a.setState((char)a.stackPop());
        case 'q': a.idMarker = a.stackPop();
        case 'r': a.setSight(a.stackPop());
        case 's': a.stackPush(a.dnapos);
        case 't': a.stackPush(getEyesight(a,s)); break; //attempt to look
for agents in the direction I'm facing
        case 'u': a.stackPush(a.getRotation()); break; //push my rotation
onto stack
        case 'v': a.stackPush(a.getState()); break; //push my state onto
stack
        case 'w': a.stackPush(getSeenState(a,s)); break; //push state of
agent I'm looking at onto stack
        case 'x': a.stackPush(getSeenMarkerID(a,s)); break; //push marker
ID of agent I'm looking at onto stack
        case 'y': a.stackPush(a.getAge()); break; //push age onto stack
        case 'z': a.stackPush(a.energy); break; //push energy onto stack
        case 'A': if (a.stackPop() < a.stackPop()) { a.jump(a.stackPop()-
1); } break;//JLT
        case 'B': if (a.stackPop() > a.stackPop()) { a.jump(a.stackPop()-
1); } break; //JGT
        case 'C': a.jump(a.stackPop()-1); break; //JMP
        case 'D': if(a.stackPop() == '\0') { a.jump(a.stackPop()-1); }
break; //JZ
        case 'E': if(a.stackPop() != '\0') { a.jump(a.stackPop()-1); }
break; //JNZ
        case 'F': a.jump(a.dnapos + a.stackPop()); break; //short jump
        case 'G': //push next DNA & skip
          tmpdnapos = a.dnapos+1;
          tmpdnapos = tmpdnapos % a.dna.length();
          a.stackPush(a.dna.charAt(tmpdnapos));
```

```
          a.dnapos = tmpdnapos;
          break;
        case 'H': a.stackPush(a.stackPop() + a.stackPop()); break; //add
        case 'I': a.stackPush(a.stackPop() - a.stackPop()); break;
//subtract
        case 'J': a.stackPush(a.stackPop() * a.stackPop()); break;
//multiply
        case 'K': //divide
          int numOne = a.stackPop();
          int numTwo = a.stackPop();
          if(numTwo != 0)
            a.stackPush(numOne / numTwo);
          else
            a.stackPush(Integer.MAX_VALUE); //divide by 0 = inf
          break;
        default : break;
      }
      a.dnapos++;

    }

  }

  @Override
  public void agentTick(SimObject s, AgentObject a)
  {
    nrgEaten = 0;

    doDNATick(a,s);

    a.consumedNrg(nrgEaten);
  }


}
```

1.3.2 energyControlModule

```
package Modules;

import SimComponents.AgentObject;
import SimComponents.Module;
import SimComponents.SimObject;

public class energyControlModule extends Module {
  final int sunCycleDuration = 500;

  @Override
  public void agentTick(SimObject s, AgentObject a)
  {
    a.incAge();

    if(a.getState() =='V')
    {
```

```
      if((s.getTicks() / sunCycleDuration) % 2 == 1)
        a.energy+=2;
    }
    else
    {
      a.energy--;
    }
  }
}
```

1.3.3 guiModule

```
package Modules;
import SimComponents.*;
import java.util.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
public class guiModule extends Module {
  Frame window;
  Image image;
  Image popGraph;
  Hashtable<Color,Integer> populationMappings;
  int highestPop = 0;
  long totalEnergy = 0;
  Color highestColor = Color.white;
  int popGraphHeight = 100;


  boolean saveToFile = false; //saves the image of the sim every 50 ticks
                //so I could use ffmpeg to make it into a movie for Little
Chip
                //FUTURE: make this a command line switch/config file?

  private void saveImage(BufferedImage bi,int frameNumber)
  {
    try {
        // retrieve image
        File outputfile = new File("image"+frameNumber+".png");
        ImageIO.write(bi, "png", outputfile); //jpegs have too many
artifacts for pixel level resolution,
                            //bitmaps are too damn big.
    } catch (IOException e) {
       System.out.println("Failed to save image!");
    }
  }
  @Override
  public void simInit(SimObject s)
  {
    if(window == null)
    {
```

```
      populationMappings = new Hashtable<Color,Integer>();
      System.out.println(s.ID+": Creating Window...");
      window = new Frame("Simulation Viewer - "+s.ID);
      window.setVisible(true);
      //window.setLocation((windows.size())*s.width,10);
      window.setSize(s.width+450, s.height+60);

      image = new
BufferedImage(window.getWidth(),window.getHeight(),BufferedImage.TYPE_INT_R
GB);
      popGraph = new
BufferedImage(400,popGraphHeight,BufferedImage.TYPE_INT_RGB);

    }
  }

  @Override
  public void simTick(SimObject s)
  {
    Graphics gc = window.getGraphics();
    window.setBackground(Color.black);
    gc.drawImage(image, 20, 40, null);
    if(s.getTicks() % 50 == 0 && saveToFile)
    {
      saveImage((BufferedImage) image, s.getTicks()/50);
    }

    Graphics ig = image.getGraphics();
    ig.clearRect(0,0,image.getWidth(null),image.getHeight(null));

    ig.setColor(Color.black);
    ig.fillRect(0, 0, s.width+1, s.height+1);

    Graphics pg = popGraph.getGraphics();
    pg.copyArea(0, 0, popGraph.getWidth(null), popGraphHeight, -1, 0);
//translate graph 1 pixel left

    Color[] pixelMap = new Color[popGraphHeight+1];
    pg.setColor(Color.black);
    pg.drawLine(popGraph.getWidth(null)-1,0,popGraph.getWidth(null)-
1,popGraphHeight);

    int distinctSpecies = 0;
    int distinctGenomes = 0;

    for(Color c : populationMappings.keySet())
    {

      distinctGenomes++;
      int spikeheight = (int) Math.round(((populationMappings.get(c)*1.0) /
s.agents.size()) * popGraphHeight);
      if(spikeheight > popGraphHeight) //TODO: figure out what is going on
here
```

```
      {
        spikeheight = 100;
      }
      if(spikeheight > 0)
      {
        distinctSpecies++;
        for(int i=spikeheight; i>=0; i--)
        {
          int pixel = popGraphHeight-i;
          if(pixel < 0)
          {
            System.out.println(i);
          }
          if(pixelMap[pixel] != c)
          {
            if((pixelMap[pixel] == null) ||
(populationMappings.get(pixelMap[pixel]) > populationMappings.get(c)))
            {
              pixelMap[pixel] = c;
              pg.setColor(c);
              pg.drawLine(popGraph.getWidth(null)-1, pixel,
popGraph.getWidth(null)-1,  pixel);
            }
            else
            {
              break;
            }
          }
        }
      }
    }
    ig.setColor(Color.white);

    ig.drawString("Population: "+s.agents.size(), s.width+20, 20);
    ig.drawString("Ticks: "+s.getTicks(), s.width+20, 40);

    ig.drawString("Ticks per second: "+s.getTicksPerSecond(), s.width+20,
60);
    ig.drawString("Total Energy: "+totalEnergy, s.width+20, 80);
    ig.drawString("Distinct genomes: "+distinctGenomes,  s.width+20, 100);
    ig.drawString("Distinct species: "+distinctSpecies,  s.width+20, 120);
    ig.setColor(highestColor);
    ig.drawString("Highest population: "+highestPop, s.width+20, 140);
    ig.drawImage(popGraph, s.width+20, 160, null);

    populationMappings.clear();

    totalEnergy = 0;
    highestPop = 0;
  }

  private Color getColor(AgentObject a)
  {
```

```
/*  Color agentColor = Color.white;
  if(colorMappings.containsKey(a.genus))
  {
    agentColor = colorMappings.get(a.genus);
  }
  else
  {
  /*  int G = 0;
    if(a.dna.length()>0)
    {
      for(int i=0; i<a.dna.length(); i++)
      {
        G+=(a.dna.charAt(i)/255.0) * (Integer.MAX_VALUE/a.dna.length());
      }
    }
    agentColor = new Color(a.genus,false);

    if(agentColor.equals(Color.black))
    {
      if(a.dna.length()>0)
      {
        agentColor = Color.gray;
      }
      else
      {
        agentColor = Color.PINK;
      }
    }
    colorMappings.put(a.genus, agentColor);
  }
*/
  return new Color(a.genus,false);
  //return agentColor;
}

@Override
public void agentTick(SimObject s, AgentObject a)
{
  totalEnergy += a.energy;
  Graphics gc = image.getGraphics();
  Color agentColor = getColor(a);
  if(populationMappings.containsKey(agentColor))
  {
    int pop = populationMappings.get(agentColor);
    pop++;
    populationMappings.put(agentColor, pop);
    if(pop > highestPop)
    {
      highestPop = pop;
      highestColor = agentColor;
    }
  }
  else
```

```
    {
      populationMappings.put(agentColor,1);
    }
    gc.setColor(agentColor);
    gc.drawLine(a.x, a.y, a.x, a.y);
    //windowgc.drawImage(tmp, 0, 0, null);

  }

  @Override
  public void simEnd(SimObject s)
  {
    if(window != null)
    {
      System.out.println(s.ID+": Closing window...");
      window.setVisible(false);
      window = null;
    }
  }
}
```

1.3.4 mutationModule

```
package Modules;

import java.util.Random;

import SimComponents.AgentObject;
import SimComponents.Module;
import SimComponents.SimObject;

public class mutationModule extends Module {

  @Override
  public void agentTick(SimObject s, AgentObject a)
  {
    Random myrandom = new Random();

    if(a.mutationrate <= Double.MIN_NORMAL*2)
    {
      a.mutationrate = Double.MIN_NORMAL*2;
    }
    for(int mutpos=0; mutpos<a.dna.length(); mutpos++)
    {
      double randnum = myrandom.nextDouble();
      double mutationchance = a.mutationrate * s.mutationrate; //repair
chance * environmental mutation chance a la real-life
      if(randnum < mutationchance) //do a mutation
      {
        if(myrandom.nextDouble() < (1/6.0)) //insert at mutpos with 1/6
chance of mutation rate
        {
          //select a random point in the DNA
          //  int mutpos = myrandom.nextInt(a.dna.length()+1);
```

```
          //split DNA at point
          String tempdnastart = a.dna.substring(0,mutpos);
          String tempdnaend = "";
          if(mutpos < a.dna.length())
          {
            tempdnaend = a.dna.substring(mutpos,a.dna.length());
          }
          //generate new piece
          char randchar = (char)(myrandom.nextInt(256));
          //stick it all back together;
          a.dna = tempdnastart + randchar + tempdnaend;
        }
        else if(myrandom.nextDouble() < (1/6.0)) //delete at mutpos with
1/6 chance of mutation rate
        {
          if(a.dna.length()>1)
          {
            //select a random point in the DNA
            //  int mutpos = myrandom.nextInt(a.dna.length()+1);
            //split DNA at point
            String tempdnastart = a.dna.substring(0,mutpos);
            //split DNA at point + 1 to skip one character
            String tempdnaend = "";
            if((mutpos+1) < a.dna.length())
            {
              tempdnaend = a.dna.substring(mutpos+1,a.dna.length());
            }
            //stick back together
            a.dna = tempdnastart + tempdnaend;
          }
          else
          {
            a.dna = "";
          }

        }
        else //mutate a random piece with 4/6 chance of mutation rate
        {
          if(a.dna.length()>0)
          {
            //select a random point in the DNA
            //  int mutpos = myrandom.nextInt(a.dna.length()+1);
            //split the DNA at point
            String tempdnastart = a.dna.substring(0,mutpos);
            //split the DNA at point +1 to skip a char
            String tempdnaend = "";
            if((mutpos+1) < a.dna.length())
            {
              tempdnaend = a.dna.substring(mutpos+1,a.dna.length());
            }
            //generate random new piece
            char randchar = (char)(myrandom.nextInt(256));
            //and stick it all back together
```

```
            a.dna = tempdnastart + randchar + tempdnaend;
          }
        }

        double change = (myrandom.nextDouble()*(0.01)) - (0.01/2.0);
        a.mutationrate += change;
        //System.out.println("Mutation change: "+change+" result:
"+a.mutationrate);

      }// if did a mutation (randnum < mutationchance)
    }//for each DNA byte

  }

}
```

1.3.5 networkModule

```
package Modules;

import HelperFunctions.NetworkModuleThread;
import SimComponents.*;

public class networkModule extends Module {
  NetworkModuleThread net;

  @Override
  public void simInit(SimObject s)
  {
    //create network thread
    net = new NetworkModuleThread();
    net.initialise(s.width, s.height, s.ID);
    net.start();
  }

  public void simTick(SimObject s)
  {
    if(!net.died)
    {
      AgentObject[] incomers = net.getIncomingAgents();
      for(int i=0; i<incomers.length; i++)
      {
        //validate agent
        if((incomers[i].x < s.width) && (incomers[i].x > 0) &&
(incomers[i].y < s.height) && (incomers[i].y > 0))
        {
          if(s.positionGrid[incomers[i].x][incomers[i].y] == null)
          {
            s.positionGrid[incomers[i].x][incomers[i].y]=incomers[i];
            s.agents.add(incomers[i]);
            //System.out.println("Agent came in:"+incomers[i]);
          }
          else
          {
```

```
            //TODO: modify position and try again
          }
        }
      }
    }
    else
    { //unrecoverable error occurred, plz try again
      net.interrupt();
      System.out.println(s.ID+": Network thread died! Trying again!");
      net = new NetworkModuleThread();
      net.initialise(s.width, s.height,s.ID);
      net.start();
    }
  }

  @Override
  public void agentTick(SimObject s, AgentObject a)
  {
    //is agent at the edge?
    if((a.x >= (s.width -1)) || (a.x <= 1) || (a.y >= (s.height -1)) ||
(a.y <= 1))
    {
      if(!net.died)
      {
        //send it upstream and kill it here
        AgentObject b = new AgentObject(a);
        //b.setNetworkedFlag(); //old debug code
        net.sendAgent(b);
        s.killAgent(a);
      }
      else
      { //unrecoverable error occurred, plz try again
        System.out.println(s.ID+": Network thread died! Trying again!");
        net.interrupt();
        net = new NetworkModuleThread();
        net.initialise(s.width, s.height,s.ID);
        net.start();
      }
    }
  }

  @Override
  public void simEnd(SimObject s)
  {
    net.interrupt();
  }
}
```

1.3.6 populateModule

```
package Modules;
import java.util.Random;
import SimComponents.*;
```

```
public class populateModule extends Module {

  @Override
  public void simInit(SimObject s)
  {
    final int popcount = 3000; //this should totally be loaded from a
config file or command line

    System.out.println(s.ID+": Populating with "+popcount+" random
agents");
    Random myrandom = new Random();

    for(int i=0; i<popcount; i++)
    {
      String randdna = "";
      int randsize = myrandom.nextInt(50)+50; //ensures every agent has at
least 50 DNA,
      for(int j=0; j<randsize; j++)
      {
        randdna += (char)(myrandom.nextInt(256));
      }
      s.agents.add(new
AgentObject(myrandom.nextInt(s.width),myrandom.nextInt(s.height),randdna,my
random.nextInt(10000),myrandom.nextInt(), s.mutationrate));
    }

  }


}
```

1.3.7 recordingModule

```
package Modules;
import java.io.ByteArrayOutputStream;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

import SimComponents.AgentObject;
import SimComponents.Module;
import SimComponents.SimObject;

public class recordingModule extends Module {
  FileOutputStream recordbuffer;
  ZipOutputStream zipbuffer;


  final int recordingPeriod = 100;

  @Override
  public void simInit(SimObject s)
  {
```

```
    String filename = ((int) (System.currentTimeMillis() / 1000L))+"-
"+s.mutationrate+"-"+s.width+"x"+s.height+"r"+s.ID+".sim";
    try {
      recordbuffer = new FileOutputStream(filename);
      zipbuffer = new ZipOutputStream(recordbuffer);


      ByteArrayOutputStream bytebuffer = new ByteArrayOutputStream();
      ObjectOutputStream recordObjectBuffer = new
ObjectOutputStream(bytebuffer);
      //format is: [tick count, system time, simulation state]
      recordObjectBuffer.writeInt(s.getTicks());
      recordObjectBuffer.writeLong(System.currentTimeMillis());
      recordObjectBuffer.writeObject(s);
      recordObjectBuffer.flush();
      ZipEntry ze = new ZipEntry(Integer.toString(s.getTicks()));
      ze.setSize(bytebuffer.size());
      zipbuffer.putNextEntry(ze);
      zipbuffer.write(bytebuffer.toByteArray());

      recordObjectBuffer.close();
      zipbuffer.closeEntry();

      System.out.println(s.ID+": Recording to file:"+filename+" every
"+recordingPeriod+" frames");
    } catch (Exception e) {
      System.out.println(s.ID+": Failed to create record file: "+filename);
      e.printStackTrace();
    }
  }

  @Override
  public void simTick(SimObject s)
  {
    if(s.getTicks() % recordingPeriod == 0)
    {
      try {
        ByteArrayOutputStream bytebuffer = new ByteArrayOutputStream();
        ObjectOutputStream recordObjectBuffer = new
ObjectOutputStream(bytebuffer);
        recordObjectBuffer.writeInt(s.getTicks());
        recordObjectBuffer.writeLong(System.currentTimeMillis());
        recordObjectBuffer.writeObject(s);
        recordObjectBuffer.flush();
        ZipEntry ze = new
ZipEntry(Integer.toString(s.getTicks()/recordingPeriod));
        ze.setSize(bytebuffer.size());
        zipbuffer.putNextEntry(ze);
        zipbuffer.write(bytebuffer.toByteArray());

        recordObjectBuffer.close();
        zipbuffer.closeEntry();
      } catch (Exception e) {
```

```
        e.printStackTrace();
      }
    }
  }

  @Override
  public void agentTick(SimObject s, AgentObject a)
  {
    if((s.getTicks() % recordingPeriod == 1) && (s.getTicks() >
recordingPeriod))
    {
      //iterate through ancestry tree, remove anything older than
recordingPeriod

      a.getAncestry().prune(); //remove all from the tree except self
    }
  }



  @Override
  public void simEnd(SimObject s)
  {
    try {
      //zipbuffer.closeEntry();
      zipbuffer.flush();
      zipbuffer.close(); //in java, close() calls to all connected streams,
so this will close recordfile too
    } catch (Exception e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

}
```

1.3.8 tickCounterModule

```
package Modules;
import SimComponents.*;
public class tickCounterModule extends Module {

    @Override
    public void simEnd(SimObject s)
    {
      long seconds = s.getRunTime();
      long minutes = seconds / 60;
      seconds = seconds % 60;
      long hours = minutes / 60;
      minutes = minutes % 60;

      System.out.println(s.ID+": Runtime:
"+hours+"h:"+minutes+"m:"+seconds+"s");
      System.out.println(s.ID+": Population: "+s.agents.size());
```

```
      System.out.println(s.ID+": Ticks: "+s.getTicks());
      System.out.println(s.ID+": Ticks per second:
"+s.getTicksPerSecond());
      System.out.println(s.ID+": Average ticks per second:
"+s.getAvgTicksPerSecond());
    }
}
```

1.4 HelperFunctions

1.4.1 ModuleLoader

```
package HelperFunctions;

import java.io.File;
import java.io.FilenameFilter;
import java.net.URL;
import java.net.URLClassLoader;
import SimComponents.Module;

public class ModuleLoader {
  @SuppressWarnings("unchecked")
  public static Class<Module>[] loadModules(String directoryPath)
  {
    try
    {
      File dir = new File(directoryPath);
      File [] files = dir.listFiles(new FilenameFilter() {
        @Override
        public boolean accept(File blah, String name) {
          return name.endsWith(".class");
        }
      });
      if(files == null)
      {
        System.out.println("The provided directory is not valid:
"+directoryPath+" == "+dir);
        return new Class[0];
      }
      Class<Module>[] result = new Class[files.length];

      URL classUrl = new URL("file:///"+directoryPath);
        URL[] classUrls = { classUrl };
        @SuppressWarnings("resource")
      URLClassLoader ucl = new URLClassLoader(classUrls); //this resource
isn't closed because JRE 1.6 does not
                                  //support URLClassLoader.close()

      for (int i=0; i<files.length; i++) {
        String tmpname =
"Modules."+files[i].getName().substring(0,files[i].getName().indexOf(".clas
s"));
        Class<Module> c = (Class<Module>) ucl.loadClass(tmpname);
```

```
          result[i] = c;
      }
      //ucl.close(); //not jre6 compatible
      return result;
    }
    catch(Exception e)
    {

      System.out.println("MODULE LOADING FAILED WITH ERROR:");
      e.printStackTrace();
      return new Class[0];
    }
  }
}
```

1.4.2 NetworkModuleSendThread

```
package HelperFunctions;
import java.io.ObjectOutputStream;
import java.util.concurrent.ConcurrentLinkedQueue;

import SimComponents.AgentObject;


public class NetworkModuleSendThread extends Thread {
  ConcurrentLinkedQueue<AgentObject> agentsToSend = new
ConcurrentLinkedQueue<AgentObject>();
  ObjectOutputStream cos;
  public boolean died = false; //like interupted(), except it actually
works.

  public void initialise(ObjectOutputStream oos)
  {
    cos = oos;
  }

  public void run()
  {
    while(!this.died)
    {
      AgentObject a = agentsToSend.poll();
      if(a != null)
      {
        try {
          if(a.energy != 0)
          {
            cos.writeObject(a);
            cos.flush();
          }
        }
        catch (Exception e) {
          this.died = true;
          e.printStackTrace();
        }
```

```
      }
    }
  }
}
```

1.4.3 NetworkModuleThread

```
package HelperFunctions;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.util.ArrayList;
import SimComponents.AgentObject;

public class NetworkModuleThread extends Thread {
  private String centralhost = "www.ameliapollard.co.uk";
  final int centralport = 3456;

  Socket connection;
  ObjectOutputStream oos;
  ObjectInputStream ois;
  int randomID;

  public boolean died = false;

  NetworkModuleSendThread sendThread;

  private boolean blocked = false;

  ArrayList<AgentObject> incomingAgents;

  private int simWidth;
  private int simHeight;

  public void initialise(int width, int height, int simID)
  {
    try {
      BufferedReader br = new BufferedReader(new FileReader("net-
config.ini"));
      String line;
      while ((line = br.readLine()) != null) {
         if(line.length()>4)
         {
          centralhost = line;
          break;
         }
      }
      br.close();
    } catch (Exception e) {
      centralhost = "www.ameliapollard.co.uk";
```

```
      System.out.println(e+" error reading net-config.ini, host defaulted
to "+centralhost);
    }
    this.simWidth = width;
    this.simHeight = height;
    this.randomID = simID;
    sendThread = new NetworkModuleSendThread();
    incomingAgents = new ArrayList<AgentObject>();
    died = false;
  }


  public void run()
  {
    //start connections
    try {
      System.out.println(randomID+": Connecting to "+centralhost+" on
"+centralport);
      connection = new Socket(centralhost,centralport);
      System.out.println(randomID+": Connected!");

      oos = new ObjectOutputStream(connection.getOutputStream());
      System.out.println(randomID+": Output stream initialised.");

      ois = new ObjectInputStream(connection.getInputStream());
      System.out.println(randomID+": Input stream initialised.");

      sendThread.initialise(oos);
      System.out.println(randomID+": Send thread initialised");
      sendThread.start();
      System.out.println(randomID+": Send thread started!");

      System.out.println(randomID+": sending ID and details...");
      oos.writeInt(randomID);
      oos.writeInt(simWidth);
      oos.writeInt(simHeight);
      oos.writeObject(new AgentObject()); //no idea why, but server won't
read info from stream till an agent gets sent.
      //agents with 0 energy get dropped serverside, so this is okay to do,
just not pretty.
      System.out.println(randomID+": Finished! Network now ready for
transport.");
    } catch (Exception e) {
      this.died = true;
      if(sendThread != null)
        sendThread.interrupt();
      System.out.println(randomID+": Network module failed to connect to
host "+centralhost+" on port "+centralport+" ERROR:"+e);
    }
    while(!this.died && connection.isConnected())
    {
      AgentObject a;
      try {
```

```
        connection.setSoTimeout(1000);
        a = (AgentObject) ois.readObject();
      //  System.out.println("Incoming agent at ("+a.x+", "+a.y+")");
        while(blocked){} //TODO: REPLACE WITH SYNCHRONIZATION BLOCKS
        incomingAgents.add(a);
        //System.out.println("... Done!");
      }
      catch (SocketTimeoutException e)
      {
        //do nothing, just chillin'
      }
      catch (Exception e) {
        this.died = true;
        if(sendThread != null)
          sendThread.interrupt();
        System.out.println(randomID+": Network module: Agent read failed!
"+e);
        try { //try to shut things down cleanly
          oos.close();
          ois.close();
          connection.close();
        } catch (Exception e1) {
          //do nothing with errors, no point trying to recover stream
        }

      }
    }
    this.died = true;
  }

  public void sendAgent(AgentObject a)
  {
    if(sendThread != null)
    {
      //using a concurrent linked queue to handle synchronization, so no
need to synchronize here
      if(sendThread.died)
      {
        System.out.println(randomID+": Agent send thread dead!");
        this.died = true;
      }
      else
      {
        sendThread.agentsToSend.add(a);
      }
    }
  }

  public AgentObject[] getIncomingAgents()
  {
    blocked = true;
    AgentObject[] result = incomingAgents.toArray(new AgentObject[0]);
    incomingAgents.clear();
```

```
    blocked = false;
    //if(result.length > 0)
    //  System.out.println("Adding "+result.length+" agents to sim from
incomingAgents");
    return result;
  }
}
```

1.4.4 Node

```
package HelperFunctions;

import java.io.Serializable;
import SimComponents.AgentObject;

public class Node implements Serializable {

  private static final long serialVersionUID = -4540232526872961239L;
  private AgentObject value = null;
  public Node leftParent = null;
  public Node rightParent = null;

  public Node(AgentObject value, Node left, Node right) {
    this.value = value;
    this.leftParent = left;
    this.rightParent = right;
  }

  public AgentObject getValue()
  {
    return this.value;
  }

  public void setValue(AgentObject a)
  {
    this.value = a;
  }

  public boolean checkIntegrity()
  {
    if(this.value.mutationrate == 0)
    {
      return false;
    }
    boolean result = true;
    if(this.leftParent != null)
    {
      result = result && this.leftParent.checkIntegrity();
    }
    if(this.rightParent != null)
    {
      result = result && this.rightParent.checkIntegrity();
    }
    return result;
```

```
  }

  public void prune()
  {

    if(this.leftParent != null)
    {
      this.leftParent.prune();
      this.leftParent = null;
    }
    if(this.rightParent != null)
    {
      this.rightParent.prune();
      this.rightParent = null;
    }
  }

}
```

2 EvoSimServer code

2.1 Main Application

```
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;


public class EvoSimServer {

  final static int centralport = 3456;


  public static void main(String[] args) {
    System.out.println("Starting...");
    try {
      MonitorThread monitor = new MonitorThread();
      monitor.start();
      ServerSocket serv = new ServerSocket(centralport);
      System.out.println("Listening on port "+centralport);
      while(true)
      {
        Socket clientSock = serv.accept();
        System.out.println("Recieved new connection. Processing...");
        ClientThread ct = new ClientThread();
        ct.intitialise(clientSock);
        monitor.addClient(ct);
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

2.2 ClientThread

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.util.ArrayList;
import SimComponents.AgentObject;

public class ClientThread extends Thread {

  public int randomID;
  public int simWidth,simHeight;
  public int offsetx,offsety = 0;
  public int xmax,ymax = 0;
  private long lastcontact;
  public boolean dead;
  public boolean started;
  Socket clientSock;
  ObjectInputStream cis;
  ObjectOutputStream cos;
  ArrayList<ClientThread> clients;
  SendThread sendThread;

  public void intitialise(Socket clientSock)
  {
    this.started = false;
    this.dead = false;

    this.clientSock = clientSock;
    this.lastcontact = System.currentTimeMillis();
    try {
      cis = new ObjectInputStream(clientSock.getInputStream());
      cos = new ObjectOutputStream(clientSock.getOutputStream());

      sendThread = new SendThread();
      sendThread.initialise(cos);
      sendThread.start();
      clients = new ArrayList<ClientThread>();
      System.out.println("Client thread ready, attempting to read ID");
      clientSock.setSoTimeout(5000);
      this.randomID = cis.readInt();
      clientSock.setSoTimeout(5000);
      this.simWidth = cis.readInt();
      clientSock.setSoTimeout(5000);
      this.simHeight = cis.readInt();
      System.out.println("Done! "+this.randomID+" is
"+simWidth+"x"+simHeight);
    } catch (Exception e) {
      System.out.println("Error initialising client! "+e);
      this.started=true;
      this.dead=true;
      try {
```

```
        clientSock.close();
      } catch (IOException e1) {

      }

    }
  }

  public void setOffsetX(int x)
  {
    this.offsetx = x;
  }

  public void setOffsetY(int y)
  {
    this.offsety = y;
  }

  public void sendAgent(AgentObject a)
  {
    if(sendThread.dead || !sendThread.isAlive())
    {
      System.out.println(randomID+": Detected dead send thread, killing
client.");
      this.dead = true;
    }
    else
    {
      sendThread.agentsToSend.add(a);
    }
  }

  public ClientThread getSimAtPosition(int x, int y)
  {
    for(int i=0; i<clients.size();i++)
    {
      ClientThread tmp = clients.get(i);
      //between offset and offset+size
      if((tmp.offsetx <= x) && ((tmp.offsetx + tmp.simWidth) >= x) &&
(tmp.offsety <= y) && ((tmp.offsety + tmp.simHeight) >= y))
      {
        return tmp;
      }
    }
    //System.out.println("WARNING: COULD NOT LOCATE SIM AT ("+x+", "+y+")
SO DEFAULTED TO SELF");
    return this;
  }

  public void run()
  {
    System.out.println(randomID+": Starting main loop");
    this.started = true;
```

```
    while(!clientSock.isClosed() && !dead)
    {
      try{
        if((System.currentTimeMillis() - this.lastcontact) > 60*1000)
        {
          this.dead = true;
          System.out.println(randomID+": Network thread recieved no contact
for 60 seconds, killing.");
          break;
        }
        if(sendThread.dead || !sendThread.isAlive())
        {
          System.out.println(randomID+": Detected dead send thread, killing
client.");
          this.dead = true;
          break;
        }

        clientSock.setSoTimeout(1000);
        AgentObject a = (AgentObject) cis.readObject();
        if(a==null || a.energy <= 1) //no point wasting any sim's time with
agents with barely any energy
        {
          continue;
        }
        this.lastcontact = System.currentTimeMillis();

        int newx = a.x+offsetx;
        int newy = a.y+offsety;
        //System.out.print("Sending agent from "+randomID+" at ("+a.x+",
"+a.y+")");
        //System.out.print(" [WORLD COORD: ("+newx+", "+newy+")]");


        if(a.x >= simWidth -1) //gone off right of screen
        {
          newx += 3;
        }
        if(a.y >= simHeight -1) //gone off bottom of screen
        {
          newy += 3;
        }
        if(a.x <= 1) //gone off left of screen
        {
          newx -= 3;
        }
        if(a.y <= 1) //gone off top of screen
        {
          newy -= 3;
        }

        newx = (newx+xmax) % xmax;
        newy = (newy+ymax) % ymax;
```

```
        ClientThread ct = getSimAtPosition(newx,newy);

        a.x = (newx - ct.offsetx) % ct.simWidth;
        a.y = (newy - ct.offsety) % ct.simHeight;

        //System.out.print(" to "+ct.randomID+" at ("+a.x+", "+a.y+")");
        //System.out.println(" [WORLD COORD: ("+newx+", "+newy+")]");
        ct.sendAgent(a);
      }
      catch(SocketTimeoutException e)
      {
        //ignore me, I just have nothing to do
      }
      catch (Exception e)
      {
        this.dead = true;
        System.out.println(randomID+": threw error "+e+" in recieve loop,
marking as dead.");
      }
    }
    this.dead = true;
    sendThread.dead = true;
    try {
      sendThread.join(1000);
    } catch (InterruptedException e) {
      System.out.println(randomID+": Unable to kill send thread!");
    }
  }

}
```

2.3 MonitorThread

```
import java.awt.Rectangle;
import java.util.ArrayList;

public class MonitorThread extends Thread {

  public ArrayList<ClientThread> clients;
  int enclosex,enclosey = 0;

  public void addClient(ClientThread ct)
  {
    if(!ct.started && !ct.dead)
    {
      System.out.println("Client added, waiting on clients array lock...");
      synchronized(clients)
      {
        System.out.println("Rebuilding graph...");
        clients.add(ct);
        rebuildGraph(clients); //client added, force graph rebuild
      }
      ct.start();
```

```
    }
  }


  public void run()
  {
    clients = new ArrayList<ClientThread>();
    while(!this.isInterrupted())
    {
      try {
        sleep(500);
        if(checkHealth(clients))
        {
          System.out.println("Graph rebuild required, waiting on clients
array lock");
          synchronized(clients)
          {
            System.out.println("Rebuilding graph...");
            rebuildGraph(clients);
          }
        }
      } catch (Exception e) {
        e.printStackTrace();
      }

    }
  }

  public boolean checkHealth(ArrayList<ClientThread> clients) //is graph
rebuild necessary?
  {
    for(int i=0; i<clients.size(); i++)
    {
      if(clients.get(i).dead || (!clients.get(i).isAlive() &&
clients.get(i).started))
      {
        return true;
      }
    }
    return false;
  }


  public void rebuildGraph(ArrayList<ClientThread> clients)
  {
    enclosex = 0;
    enclosey = 0;
    //remove the dead clients
    ArrayList<ClientThread> deadClients = new ArrayList<ClientThread>();
    for(int i=0; i<clients.size(); i++)
    {
      if(clients.get(i).dead || (!clients.get(i).isAlive() &&
clients.get(i).started))
```

```
      {
        deadClients.add(clients.get(i));
      }
    }
    for(int i=0; i<deadClients.size(); i++)
    {
      System.out.println("Client "+deadClients.get(i).randomID+" is dead,
removing from graph.");
      try {
        deadClients.get(i).clientSock.close();
      } catch (Exception e) {
        //ignore errors, just try and close sockets;
      }
      clients.remove(deadClients.get(i));
    }

    //sort rectangles by height, greatest first
    //store sum of area while you're at it

    int areaSum =0;
    int maxHeight = 0;
    for(int i=0; i<clients.size(); i++)
    {
      areaSum += clients.get(i).simHeight * clients.get(i).simWidth;
      maxHeight = Math.max(clients.get(i).simHeight, maxHeight);
    }
    //quick inline insertion sort
    for (int i = 1; i < clients.size(); i++){
      int j = i;
      ClientThread B = clients.get(i);
      while ((j > 0) && (clients.get(j-1).simHeight < B.simHeight)){
        //array[j] = array[j-1];
        clients.set(j, clients.get(j-1));
        j--;
      }
      //array[j] = B;
      clients.set(j, B);

    }
    System.out.println("Area: "+areaSum+" maxHeight: "+maxHeight);

    //create enclosing rectangle with height = max height of client * 2,
width = infinity (or arbitarily large at least)
    enclosey = maxHeight*2;
    enclosex = Integer.MAX_VALUE;
    int bestArea = Integer.MAX_VALUE;

    boolean found = false;

    while(!found)
    {
      //while enclosing area < sum of area, increase height by one
      //while((enclosey * enclosex)<areaSum)
```

```
      //{
      //  enclosey++;
      //}
      //for each rectangle, places as far left as possible, if there are
several far left places, place it at highest one.
      //if unable to place a rectangle, break and increase height by one,
restart.
      int totalWidth = 0;
      ArrayList<Rectangle> gaps = new ArrayList<Rectangle>();
      gaps.add(new Rectangle(0,0,enclosex,enclosey)); //first gap is area
of entire enclosure
      boolean placed = false;
      for(int i=0; i<clients.size(); i++)
      {
        ClientThread thisClient = clients.get(i);
        //sort gaps by x position
        //quick inline insertion sort
        for (int k = 1; k < gaps.size(); k++){
          int j = k;
          Rectangle B = gaps.get(k);
          while ((j > 0) && (gaps.get(j-1).x > B.x)){
            //array[j] = array[j-1];
            gaps.set(j, gaps.get(j-1));
            j--;
          }
          //array[j] = B;
          gaps.set(j, B);

        }
        //System.out.println(gaps);


        for(int j=0; j<gaps.size(); j++)
        {
          Rectangle thisGap = gaps.get(j);
          if((thisGap.width >= thisClient.simWidth) && (thisGap.height >=
thisClient.simHeight))
          {
            //Found a gap this client will fit in
            thisClient.setOffsetX(thisGap.x);
            thisClient.setOffsetY(thisGap.y);

            gaps.remove(thisGap);
            //split this gap into two new gaps representing the area left
over from thisGap - thisClient
            /*
             * +-----+------------+
             * | sim |            |
             * |-----+  gap1      |
             * | gap2|            |
             * +-------------------+
             */
            //first is from right edge of sim to far edge of enclosure
```

```
            Rectangle newGapOne = new
Rectangle(thisClient.offsetx+thisClient.simWidth,thisClient.offsety,thisGap
.width-(thisClient.offsetx+thisClient.simWidth), thisGap.height -
thisClient.offsety);
            //second is from bottom of sim to bottom of enclosure
            Rectangle newGapTwo = new
Rectangle(thisClient.offsetx,thisClient.offsety+thisClient.simHeight,thisCl
ient.simWidth, thisGap.height-(thisClient.offsety+thisClient.simHeight));
            if(newGapOne.width > 5 && newGapOne.height > 5)
            {
              gaps.add(newGapOne);
            }

            if(newGapTwo.width > 5 && newGapTwo.height > 5)
            {
              gaps.add(newGapTwo);
            }

            placed = true;
            break;
          }
        }
        if(!placed)
        {
          enclosey++;
          break;
        }
        totalWidth = Math.max(totalWidth,
thisClient.offsetx+thisClient.simWidth);
      }
      //reduce enclosing rectangle width to fit the enclosed
      enclosex = totalWidth;
      //if best, store enclosing rectangle
      if(bestArea > (enclosex * enclosey))
      {
        bestArea = enclosex * enclosey;
        found = true;
      }

      //decrease width by one, restart
      //enclosex--;
    }


    //Make sure every client has the new graph details.
    System.out.println("New map ["+enclosex+"x"+enclosey+"]:");
    for(int i=0; i<clients.size(); i++)
    {
      clients.get(i).xmax = enclosex;
      clients.get(i).ymax = enclosey;
      clients.get(i).clients = clients;
```

```
      System.out.println("Client "+clients.get(i).randomID+" at
("+clients.get(i).offsetx+", "+clients.get(i).offsety+") with dimensions
"+clients.get(i).simWidth+"x"+clients.get(i).simHeight);
    }

  }

}
```

2.4 SendThread

```
import java.io.ObjectOutputStream;
import java.util.concurrent.ConcurrentLinkedQueue;
import SimComponents.AgentObject;

public class SendThread extends Thread {
  ConcurrentLinkedQueue<AgentObject> agentsToSend;
  ObjectOutputStream cos;
  public boolean dead = false;

  public void initialise(ObjectOutputStream oos)
  {
    agentsToSend = new ConcurrentLinkedQueue<AgentObject>();
    cos = oos;
  }

  public void run()
  {
    while(!this.dead)
    {
      AgentObject a = agentsToSend.poll();
      if(a != null)
      {
        try {
          cos.writeObject(a);
          cos.flush();
        }
        catch (Exception e) {
          this.dead = true;
          System.out.println("Error in send thread, marking as dead! "+e);
        }
      }
    }
  }
}
```

3 Sim Stat code

3.1 Main Application

```
public class AncestryTreeBuilder {

  public static void main(String[] args) {
    MainWindow window = new MainWindow();
    window.setVisible(true);
  }

}
```

3.2 Main Window

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import javax.swing.*;
import javax.swing.border.BevelBorder;
import SimComponents.AgentObject;
import SimComponents.SimObject;
import CustomComponents.ImagePanel;
import CustomComponents.SimReadThread;
import CustomComponents.GraphNode;
import HelperFunctions.Node;


public class MainWindow extends JFrame {
  private static final long serialVersionUID = 665600370356328750L;

  private JButton loadSimBtn;
  private JButton downBtn;
  private JButton upBtn;
  private JButton leftBtn;
  private JButton rightBtn;
  private JPanel btnPanel;
  private ImagePanel graphView;
  private JScrollPane graphPanelSP;
  private JFrame miniGraphWindow;
  private ImagePanel miniGraphView;
  private JScrollPane miniGraphPanelSP;
  private File simFile,outFile;
  int offsetx,offsety = 0;
  int graphWidth = 0;
  int graphHeight = 0;
  private Hashtable<Long,Node> nodeHashes;
  private Hashtable<Long,GraphNode> graphNodes;
```

```
  public MainWindow()
  {
    super("Player Window");

      try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      } catch(Exception e) {
          System.out.println("Error setting native LAF: " + e);
      }
      this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setSize(1024, 512);

    loadSimBtn = new JButton("Load Simulation");
    loadSimBtn.addActionListener(new ActionListener()
        {
          @Override
          public void actionPerformed(ActionEvent arg0) {
            JFileChooser jfc = new JFileChooser();
            JFileChooser jfc2 = new JFileChooser();

            if(jfc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
            {
              if(jfc2.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
              {
                simFile = jfc.getSelectedFile();
                outFile = jfc2.getSelectedFile();
                proccessFile();
                miniGraphWindow.setVisible(true);
              }
            }
          }
        }
    );

    upBtn = new JButton("^");
    upBtn.addActionListener(new ActionListener(){

      @Override
      public void actionPerformed(ActionEvent arg0) {
        offsety -= 0.9 * graphPanelSP.getHeight();
        renderGraph(graphNodes,offsetx,offsety);
        renderMiniGraph(graphNodes,graphWidth,graphHeight);
      }
    });
    downBtn = new JButton("v");
    downBtn.addActionListener(new ActionListener(){

      @Override
      public void actionPerformed(ActionEvent arg0) {
        offsety += 0.9 * graphPanelSP.getHeight();
        renderGraph(graphNodes,offsetx,offsety);
        renderMiniGraph(graphNodes,graphWidth,graphHeight);
```

```
      }
    });
    leftBtn = new JButton("<");
    leftBtn.addActionListener(new ActionListener(){

      @Override
      public void actionPerformed(ActionEvent arg0) {
        offsetx -= 0.9 * graphPanelSP.getWidth();
        renderGraph(graphNodes,offsetx,offsety);
        renderMiniGraph(graphNodes,graphWidth,graphHeight);
      }
    });
    rightBtn = new JButton(">");
    rightBtn.addActionListener(new ActionListener(){

      @Override
      public void actionPerformed(ActionEvent arg0) {
        offsetx += 0.9 * graphPanelSP.getWidth();
        renderGraph(graphNodes,offsetx,offsety);
        renderMiniGraph(graphNodes,graphWidth,graphHeight);
      }
    });


    miniGraphView = new ImagePanel();
    miniGraphPanelSP = new JScrollPane(miniGraphView);
    miniGraphWindow = new JFrame("Minigraph");
    miniGraphWindow.setPreferredSize(new Dimension(512,512));

    miniGraphWindow.getContentPane().add(miniGraphPanelSP);

    graphView = new ImagePanel();

graphView.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
    graphPanelSP = new JScrollPane(graphView);
    graphPanelSP.setPreferredSize(new Dimension(768,768));

    btnPanel = new JPanel();

btnPanel.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
    btnPanel.add(upBtn);
    btnPanel.add(downBtn);
    btnPanel.add(leftBtn);
    btnPanel.add(rightBtn);

    Container content = this.getContentPane();
    content.setLayout(new BorderLayout());
    content.add(loadSimBtn, BorderLayout.NORTH);
    content.add(btnPanel, BorderLayout.SOUTH);
    content.add(graphPanelSP, BorderLayout.CENTER);
    //content.add(graphPanelSP, BorderLayout.WEST);
    nodeHashes = new Hashtable<Long,Node>();
```

```
  }


  private void placeParents(GraphNode realRoot) //ypos is determined by
generation
  {
    //System.out.println(realRoot.node.getValue().uniqueID); //debug code
to make sure it's not stuck in a cycle
    if(realRoot.node.leftParent != null)
    {

if(!graphNodes.containsKey(realRoot.node.leftParent.getValue().uniqueID))
//same as a visited list, more or less
      {
        GraphNode lgn = new GraphNode(realRoot.node.leftParent,realRoot.x-
100,0);
        lgn.y = 20+(100 * lgn.node.getValue().generation);
        graphNodes.put(lgn.node.getValue().uniqueID, lgn);
        placeParents(lgn);
      }
    }
    if(realRoot.node.rightParent != null)
    {

if(!graphNodes.containsKey(realRoot.node.rightParent.getValue().uniqueID))
      {
        GraphNode rgn = new
GraphNode(realRoot.node.rightParent,realRoot.x+100,0);
        rgn.y = 20+(100 * rgn.node.getValue().generation);
        graphNodes.put(rgn.node.getValue().uniqueID, rgn);
        placeParents(rgn);
      }
    }
  }

  private void drawGraph()
  {
    /*
    NOTE:

    AgentObject.generation IS NOT EQUIVELENT TO DEPTH IN TREE
    the reason for this is that agents can be exchanged over the network,
can have parents from 0..nth generation etc.
    TL;DR: cos it's not quite a binary tree
    having said that, real leaves (those created at the start of the sim)
should have a generation of 0, and leaves with a higher
    generation must have come in from other sims, or else data was lost

    Clarification of terminology:
    An agent can have up to two parents. These parents are children of the
node (like in a binary tree)
    A root node is an agent with no offspring.
```

```
    A leaf node is an agent with no parents. - these are the agents the sim
was first populated with
    The root nodes of the trees are the agents alive in the last save file
- All paths lead to them.


    */
    graphNodes  = new Hashtable<Long,GraphNode>();


    System.out.println("Positioning nodes...");

    Node[] nodes = nodeHashes.values().toArray(new Node[0]);
    int realroots = 0; //last surviving agents = roots of each tree
    double mutationAverage = 0;
    Hashtable<Integer,ArrayList<Double>> mutationData = new
Hashtable<Integer,ArrayList<Double>>();
    //first get a list of root nodes
    System.out.println("Total nodes: "+nodes.length);

    for(int i=0; i<nodes.length; i++) //instantiate all graph nodes
    {
      if(nodes[i].getValue().energy > 0) //was this node alive at the end?
      {
        boolean realroot = true;
        for(int j=0; j<nodes.length; j++) //is this a root? Does any node
claim to be it's child?
        {
          if(nodes[j].leftParent == nodes[i] || nodes[j].rightParent ==
nodes[i])
          {
            realroot = false;
            break;
          }
        }
        if(realroot) //this is a root, use it to recursively position
parents
        {
          realroots++;
          GraphNode gn = new GraphNode(nodes[i],0,0);
          gn.y = 20+(100 * nodes[i].getValue().generation);
          graphNodes.put(nodes[i].getValue().uniqueID, gn);
          gn.x = realroots * 1000;
          placeParents(gn);
          graphHeight = Math.max(gn.y, graphHeight);
          graphWidth = Math.max(gn.x, graphWidth);
          mutationAverage += nodes[i].getValue().mutationrate;
          ArrayList<Double> data =
mutationData.get(nodes[i].getValue().genus);
          if(data == null)
          {
            data = new ArrayList<Double>();
          }
```

```
          data.add(nodes[i].getValue().mutationrate);
          mutationData.put(nodes[i].getValue().genus, data);

        }
      }
    }

    try
    {
      BufferedWriter jsonout = new BufferedWriter(new FileWriter(outFile));
      //JSON output:
      // {data : [{"genus": 123321432, "mutationrate": [0.1,0.4,0.1,0.1]},
{"genus": 6789, "mutationrate":[0.01,0.1,0.2]}]}
      Integer[] rootGenusArray = mutationData.keySet().toArray(new
Integer[0]);
      jsonout.write("{\"data\": [");
      for(int i=0; i<rootGenusArray.length; i++)
      {
        ArrayList<Double> data = mutationData.get(rootGenusArray[i]);
        jsonout.write("{\"genus\": "+rootGenusArray[i]+", \"mutationrate\":
[");
        for(int j=0; j<data.size(); j++)
        {
          jsonout.write(data.get(j).toString());
          if(j != (data.size()-1))
          {
            jsonout.write(",");
          }
        }
        jsonout.write("]}");
        if(i != rootGenusArray.length-1)
        {
          jsonout.write(",");
        }
      }
      jsonout.write("]}");
      jsonout.close();
    }
    catch(Exception e)
    {
      System.out.println("Failed to write JSON file!");
    }

    System.out.println("Average mutation rate of final
agents:"+(mutationAverage/realroots));

    System.out.println("Real roots: "+realroots);



    System.out.println("Drawing graph
("+graphWidth+"x"+graphHeight+")...");
```

```
    renderGraph(graphNodes,offsetx,offsety);
    renderMiniGraph(graphNodes,graphWidth,graphHeight);
    System.out.println("Done!");
  }

  public void renderMiniGraph(Hashtable<Long,GraphNode> graphNodes,int
graphWidth, int graphHeight)
  {

    int scalex = graphWidth/1280; //100;
    int scaley = graphHeight/768; //5;

    BufferedImage minigraph = new
BufferedImage(graphWidth/scalex,graphHeight/scaley,BufferedImage.TYPE_INT_R
GB);
    Graphics mgc = minigraph.getGraphics();
    GraphNode[] finalNodes = graphNodes.values().toArray(new GraphNode[0]);

    //render minigraph
    for(int i=0; i<finalNodes.length; i++)
    {
      GraphNode gn = finalNodes[i];
      int x = gn.x / scalex;
      int y = gn.y / scaley;

      mgc.setColor(new Color(gn.node.getValue().genus,false));
      mgc.drawString("X", x, y);


      if(gn.node.leftParent != null)
      {
        GraphNode lgn =
graphNodes.get(gn.node.leftParent.getValue().uniqueID);
        mgc.drawLine(gn.x/scalex, gn.y/scaley, lgn.x/scalex, lgn.y/scaley);
      }
      if(gn.node.rightParent != null)
      {
        GraphNode rgn =
graphNodes.get(gn.node.rightParent.getValue().uniqueID);
        mgc.drawLine(gn.x/scalex, gn.y/scaley, rgn.x/scalex, rgn.y/scaley);
      }
    }

    //finally, draw viewfinder rectangle
    mgc.setColor(Color.red);
    mgc.drawRect(-offsetx/scalex, -offsety/scaley,
graphPanelSP.getWidth()/scalex, graphPanelSP.getHeight()/scaley);

    miniGraphWindow.setExtendedState(MAXIMIZED_BOTH);
    miniGraphView.setImage(minigraph);
    miniGraphView.repaint();
  }
```

```
  public void renderGraph(Hashtable<Long,GraphNode> graphNodes, int
offsetx, int offsety)
  {
    GraphNode[] finalNodes = graphNodes.values().toArray(new GraphNode[0]);

    int maxFrameWidth = graphPanelSP.getWidth();
    int maxFrameHeight = graphPanelSP.getHeight();


    BufferedImage graph = new
BufferedImage(maxFrameWidth,maxFrameHeight,BufferedImage.TYPE_INT_RGB);
    Graphics gc = graph.getGraphics();

    gc.setColor(Color.white);
    gc.fillRect(0, 0, graph.getWidth(), graph.getHeight());

    for(int i=0; i<finalNodes.length; i++)
    {
      GraphNode gn = finalNodes[i];
      //if(gn.x >= offsetx && gn.x < (maxFrameWidth+offsetx) && gn.y >=
offsety && gn.y < (maxFrameHeight+offsety))
      {
        int x = gn.x + offsetx;
        int y = gn.y + offsety;
        AgentObject a = gn.node.getValue();
        gc.setColor(new Color(a.genus,false));
        gc.drawString("UID:"+a.uniqueID, x, y);
        gc.drawString("Mutation rate:"+a.mutationrate, x, y+10);
        gc.drawString("DNA:"+a.dna, x, y+20);
        gc.drawString("Generation:"+a.generation, x, y+30);
        gc.drawString("Genus:"+a.genus, x, y+40);

        if(gn.node.leftParent != null)
        {
          GraphNode lgn =
graphNodes.get(gn.node.leftParent.getValue().uniqueID);
          gc.drawLine(gn.x+ offsetx, gn.y+ offsety, lgn.x+ offsetx, lgn.y+
offsety);
        }
        if(gn.node.rightParent != null)
        {
          GraphNode rgn =
graphNodes.get(gn.node.rightParent.getValue().uniqueID);
          gc.drawLine(gn.x+ offsetx, gn.y+ offsety, rgn.x+ offsetx, rgn.y+
offsety);
        }
      }
    }
    graphView.setImage(graph);
    graphView.repaint();
  }

  private void proccessFile()
```

```
  {
    int cores = 500; //bottleneck is unzipping, no need to limit number of
threads processing data

    SimReadThread[] fileThreads = new SimReadThread[cores];
    try {

      int i = 0;
        ZipInputStream zipFile = new ZipInputStream(new
FileInputStream(simFile));
        ZipEntry ze = zipFile.getNextEntry();

        while((ze != null) )
        {
          BufferedInputStream bytebuffer = new
BufferedInputStream(zipFile);
          ObjectInputStream ois = new ObjectInputStream(bytebuffer);

          int frameNumber = ois.readInt();
          long sysTime = ois.readLong();

          System.out.println("Trying to read frame "+frameNumber+" sys time
was "+sysTime);
          SimObject s = (SimObject) ois.readObject();
          zipFile.closeEntry();

          boolean found = false;
          i = 0;
          while(!found)
          {

            if((fileThreads[i % cores] == null) || (!fileThreads[i %
cores].isAlive()))
            {
              found = true;
              System.out.println("Thread"+(i % cores)+" pulling info from
"+frameNumber);
              fileThreads[i % cores] = new SimReadThread();
              fileThreads[i % cores].setHashes(nodeHashes);
              fileThreads[i % cores].setObject(frameNumber,s);
              fileThreads[i % cores].start();
            }
            i++;
          }
          s = null;
          ois = null;
          bytebuffer = null;
          ze = zipFile.getNextEntry();
          System.gc(); //ew
        }
        zipFile.close();
        System.out.println("Waiting for final threads...");
        boolean done = false;
```

```
        while(!done)
        {
          done = true;
          for(int j=0; j<fileThreads.length; j++)
            if(fileThreads[j] != null)
              done = done && !fileThreads[j].isAlive();
        }
      drawGraph();

    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }
}
```

3.3 CustomComponents

3.4 GraphNode

```
package CustomComponents;

import HelperFunctions.Node;

public class GraphNode  {

  public Node node;
  public int x,y;
  public int children;

  public GraphNode(Node node, int x, int y)
  {
    this.node = node;
    this.x = x;
    this.y = y;
    this.children = 0;

  }
}
```

3.5 ImagePanel

```
package CustomComponents;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JPanel;

public class ImagePanel extends JPanel{

  private static final long serialVersionUID = 8559702878784618894L;
```

```
  private BufferedImage image;

    public ImagePanel()
    {
      this.image = new BufferedImage(1, 1, BufferedImage.TYPE_INT_RGB);
    }

    public ImagePanel(BufferedImage image) {
      super();
      this.image = image;
    }

    public void setImage(BufferedImage image)
    {
      this.image = image;
      this.setPreferredSize(new
Dimension(image.getWidth(),image.getHeight()));
    }


    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(image, 0, 0, null);
    }

}
```

3.6 SimReadThread

```
package CustomComponents;

import java.awt.Color;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;

import HelperFunctions.Node;
import SimComponents.AgentObject;
import SimComponents.SimObject;

public class SimReadThread extends Thread {

  private int frameNumber;
  private Hashtable<Long,Node> nodeHashes;
  private SimObject s;

  public void setHashes(Hashtable<Long,Node> nodes)
  {
```

```
    this.nodeHashes = nodes;
  }


  public void setObject(int frameno, SimObject s)
  {
    this.s = s;
    this.frameNumber = frameno;

  }

  public Color getColor(AgentObject a)
  {
    return new Color(a.genus, false);
  }

  public void putTree(Node tree)
  {
    if(tree.leftParent != null)
    {
      putTree(tree.leftParent);
      //tree.leftParent = nodeHashes.get(tree.leftParent.value.uniqueID);
    }
    if(tree.rightParent != null)
    {
      putTree(tree.rightParent);
      //tree.rightParent = nodeHashes.get(tree.rightParent.value.uniqueID);
    }
    if(tree.getValue().mutationrate == 0)
    {
      System.out.println("Bad agent in putTree!");
    }
    nodeHashes.put(tree.getValue().uniqueID, tree);
  }




  public void run()
  {
    for(int j=0; j<s.agents.size(); j++)
    {
      AgentObject a = s.agents.get(j);
      putTree(a.getAncestry());
    }
    System.out.println("Finished frame "+frameNumber);
    this.interrupt();
  }
}
```