Spring 2009

Predator-Prey Simulations

Joe O'Connor
Carroll College, Helena, MT

Follow this and additional works at: https://scholars.carroll.edu/mathengcompsci_theses
Part of the Computer Sciences Commons

Recommended Citation
https://scholars.carroll.edu/mathengcompsci_theses/46

This Thesis is brought to you for free and open access by the Mathematics, Engineering and Computer Science at Carroll Scholars. It has been accepted for inclusion in Mathematics, Engineering and Computer Science Undergraduate Theses by an authorized administrator of Carroll Scholars. For more information, please contact tkratz@carroll.edu.
Predator-Prey Simulations

By
Joe O'Connor
December 2008

This thesis for honors recognition has been approved by: Department of Mathematics, Engineering, and Computer Science.

<table>
<thead>
<tr>
<th>Signature of Thesis Director</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12/16/08</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signature of Reader</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12/16/08</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signature of Reader</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12/15/08</td>
</tr>
</tbody>
</table>
Predator-Prey Simulations

Introduction

Representing our reality is the goal of simulation for computer scientists. This is done for many reasons, but my reason is just to show I can abstract the fundamental attributes of animals in a system. I decided to take baby steps to get this program fully running, starting with what I thought the simplest of motives for these creatures were and then building upon those till I had satisfactory chases. These programs, or classes in Java, built upon previous programs to create more complex actions and rules for choosing action.

The creatures in this thesis come from the same constructor class. They are made individual by having different values assigned to their attributes. For instance, one may be faster than another or be able to sense another creature coming from farther away. The creatures act in each program based on four fundamental properties:

1) They are points in the world
2) They move in the world in 3-dimensions and there are no boundaries or conditions that inhibit them from moving in one better than direction the others.
3) They move over discrete time.
4) After a certain amount of time their energy runs out and they die.

Program 1: Immobile Creatures

The foundation I chose to start on for these creatures was the reason for them to eat. This cause is the loss of energy over time.
In this simple program they have no way to replenish their energy and the only thing that is tracked is that they are losing it. There are two creatures. There is no environment that connects them, because they can’t move through it. This is the simplest stage of my thesis, before I give the creatures a way to increase their energy before they die. I evaluate the winner of this as whoever survives the longest. In the example, the predator starts out with less energy, so it loses. A traditional way to look at this situation in programming is that the creatures are essentially timers waiting to go off. This imminent death is counteracted in the next program.

![Graph showing energy over time for two creatures.](image)

**Moving Creatures**

The next level of this simulation has movement for one of the creatures. This movement is random. The creature’s full movement ability is put in random amounts in the x, y, and z direction each time step.
This means that a random amount of the whole distance the creature can move is assigned to the x direction. Then the y direction has a random amount of the distance not traveled assigned to it and so on for z. (The creature could move nowhere if each random amount was 0% of the total.) The moving creature is the prototype of my predator.

The other rule assigned to the predator, besides random moving each turn, is that if it gets within a certain distance of the prey it can "kill" it. Killing removes all the energy the prey has left and gives the predator energy. The predator starts out with less energy than the prey. This means the program can have one of two outcomes: either the predator finds the prey and kills it or doesn’t find it and dies first.

**Predator Movement**

 Creatures with Senses

The next addition to this creature behavior is to add senses. Senses let the creature know if another creature is in a certain distance of it. If the creature is a predator, it will chase after the prey, if it is the prey its goal will be to get the farthest it can from the
predator at any one time. If the creature does not sense another creature around, it will do a random walk for its movement.

There are still only two creatures in this simulation: one predator and one prey. The predator can succeed in killing the prey if the prey cannot sense the predator or if the prey cannot move as fast as the predator and gets caught before the predator dies. The predator finds the shortest path to the prey and moves as far along that path as it can. The prey does the exact opposite, running in the direction that will put it farthest away from the predator. In this graph, the predator randomly walks closer to the prey. The prey senses the predator before being sensed and starts running away. Then the predator senses the prey and moves toward it much faster than the prey can run away.

![Graph showing predator and prey movements](image)

Many Creatures

The next improvement in this program is to create a plethora of creatures. The prey now acts as predators on a new creature called flora. There are multiple prey, and multiple flora. They are all assigned random positions in the world around the center of a sphere, in this case the origin, with a radius of 50. The flora cannot move, but are not
destroyed by the prey when the prey “eats” them. The prey gains energy and nothing else happens.

The prey has three rules it will follow in descending priority. First, it will try to run away from any predator it senses. Second, if it has enough energy and no predator sensed, it will random walk. Third, if it is below half its starting energy it will try to get next to the closest flora it can sense. With these rules the prey will only care about its energy if it is below a certain amount, or “hungry”, and will always care about escaping predators over replenishing its energy.

There is only one predator in this model. The predator follows a similar model as the prey, and will only look for prey at below half its starting energy level. When it does chase after prey, it selects only the closest one in that time step to pursue. If another prey becomes closer next time step it will go after the closest prey in this next time step. After the predator has killed one prey, it moves on to the next one. Dead prey still exist in the time loop but do not act and are not looked for by the predator. In the example simulation (shown on the next page), the predator moves towards a selected prey about the $x = -20$ line. The prey respond by moving away from the predator and then towards flora locations about the $x = 20$ line. The predator dies before catching the prey.
Group Hunting

The next advancement is to have multiple predators in the environment. The predators are randomly assigned like the prey and flora about a sphere of radius 50 centered on the origin. Now the predators have two actions that they will follow depending on where they start: they will compete with other predators or will cooperate with them to chase down and kill prey. Predators are assigned to groups at the start of the simulation. They will be in a group with another predator if they sense that other predator. (Even if a predator is by itself it will be assigned to a group and will be its only member.) These groups now search for the prey much like the sole predator searched originally.

Competition for prey occurs when two groups have their closest prey be the same prey. The predator who is closest to the prey, and therefore its group, will go after that prey and the predators not assigned to the closest predators group will search for another prey. This is a first come, first serve sort of competition. Predators do not fight predators for prey.

Cooperation for prey in a group has a different sort of hunting strategy. The group now selects the prey closest to all the members of the group. The closest prey, on average, is selected. In a group, the predators do not try to minimize their distance to the prey. Instead, the predators try to minimize the area the prey can move to. The prey can move anywhere in a sphere around it. The predators try to minimize the area they can’t reach in this sphere by surrounding this sphere in a cube whose points are the predators. The predators try to create this cube by moving to points at the end of diagonals-stretching out from the center of the cube, the prey. The length of the diagonals is the
distance the predators can move in one turn minus the distance required to kill, or the farthest the predator could be away next turn to kill the prey if the prey didn’t move. The predator closest to a point moves to that point and other predators must move toward their closest point that isn’t taken. If there are more than eight predators, the predators not closest to the points go for the prey. If there are less than eight predators, the predators try to fill in points furthest away from each other to still minimize the area of movement for the prey.

All of this happens like clockwork if every predator senses the prey. If the predator doesn’t sense the prey, it will move towards the closest predator in its group, if it doesn’t sense another predator, it will random walk. When the group kills the prey, energy is distributed throughout the group and the group chooses another prey if the group’s average energy is below half the starting energy for the group. If a predator’s energy is below half its starting energy, and the group’s energy is sufficient to not hunt, the predator will hunt alone till the group’s energy is low enough to warrant it searching for prey as well.

The prey’s optimal path has been changed to match the multiple predators it must avoid. The prey now will take into account all the predators it senses, and will avoid as many predators as it can. This decision is weighted to avoid the closest predators first, the distance one predator is from the prey is weighted against the total distance all sensed predators are away from the prey and is prioritized accordingly. The new direction the prey takes is the opposite of all the predators with the closest predators changing the direction further away from them. If the prey were pinned in the x, y, and z coordinates in a perfect cube it would not move. Perfection of this cube is extremely unlikely for the
predators to construct unless they move at ridiculous speeds, and then the cube wouldn’t be necessary. In the example, the predators kill one prey quite rapidly and then move on to construct a cube around the second prey, which they kill in twice the time.

**Conclusions**

The predator prey chase scene has progressed from immobile creatures waiting for death to packs searching to pin down the closest prey. There is competition, in its simplest form, and three types of creatures. Energy levels of a creature determine what actions it will take to keep its energy up. This system has recreated shortest path finding for creatures in a modest predator-prey relationship. The predators’ movements in group hunting alone is quite impressive at catching prey.

Now that the program is done, what affect did it have on me? I learned five things by heart during this process that are taught in coding classes but that you really don’t appreciate without spending hours of your time on them:
1) All computer languages are 90% the same except for syntax, which will kill your code; so get a book on each language.

2) Building simple programs that work and then expanding on them is the best way to create later simulations that at first seem hideous and confusing to code.

3) Even after your simple programs are coded; write out pseudo-code to make sure your code (which will error the first time) is sound at least logically.

4) Debugging takes up way too much of a coder’s time, and if scripts aren’t created to debug for you, then you are setting yourself up for the most mind-numbing task known to man.

5) Decoding the code you wrote into real ideas other people can understand, after encoding those real ideas into a code that the computer will run, has a lot of translation errors and unknown assumptions; and even more if you don’t look back on your original encoding process.

Now I’m going to go watch my predators circle prey and kill them.
Bibliography


Software Used

Eclipse Java Environment Version 3.3.1.1.

Java 2 Platform Development Kit Version 5.0

Code Appendix

Creature Constructor

/** Created 8/15/08
 * @author Joe O'Connor
 * for Carroll College Honors Thesis
 */

package survival;

/** This class is the creature constructor for all creatures interacting for survival in my thesis.
 * This is the fundamental level of creature.
 * All distances in meters.
 */

class Creature {
  int level = 0;
  boolean dead = false;
  boolean[] grouped = {false, false}; // boolean for predators to cooperate or not to.
  double[] energy = {20}; // current energy creature possesses on related plane.
  double[] maxenergy = {40}; // total energy creature can store.
  double[] energyexpend = {1, 2, 4}; // 0 position is at rest and final position is full energy use.
  double[] position = {0, 0, 0}; // default position. Every creature has a separate position on the physical plane.
  double speed = 0; // every creature has a speed, even if 0. Defaulted to unmoving creature.
  double[] senses = {30, 30, 30, 30}; // senses arranged from top of head down: eye, ear, nose, touch. taste irrelevant at this stage.
public void setposition( double x, double y, double z) {  //allows creatures position
to be set to other than default.
    position[0]=x;
    position[1]=y;
    position[2]=z;
}

public void setspeed( double s) {  //allows creatures speed to be set to other than
default.
    speed = s;
}

public void setsenses( double eye, double ear, double nose, double touch) {  //allows creatures senses to be set to other than default.
    senses[0] = eye;
    senses[1] = ear;
    senses[2] = nose;
    senses[3] = touch;
}

public void setenergy(double physical){
    energy[0] = physical;
}

public void addenergy(double physical){
    energy[0] += physical;
}

public void subtractenergy(double physical){
    energy[0] = energy[0] - physical;
}

public void setmaxenergy(double maxphysical){
    energy[0] = maxphysical;
}

public void setlevel(int templevel){
    level = templevel;
}

Reality 2 (Movement functions and kill introduced)

package survival;
import java.util.Random;
/**This class represents the two creatures dying over time. One creature can kill the other but is
left without
* senses to wander aimlessly till its death. The chances of it finding its prey are heart-breaking.
public class Reality2{
    public static void main(String[] args) {
        Creature predator, prey; // declares new creatures predator and prey.
        predator = new Creature();
        prey = new Creature();
        double timestep = 1; // passage of time in seconds
        double distance = 0; // distance between creatures
        int numofsteps = 100; // # of time iterations in this program.
        int level = 0; // energy state of creatures. 0 is default and is at rest.
        predator.setenergy(6); // sets examples at different energy levels
        prey.setenergy(10);
        predator.setPosition(7, 20, -30); // setting predator and prey at different positions
        predator.setSpeed(1);
        prey.setPosition(10, 10, -10);
        for(int i = 0; i < numofsteps; i++) {
            distance = finddistance(predator.position, prey.position);
            prey.dead = kill(distance); // checks to see if predator kills prey.
            if(prey.dead) {
                predator.setenergy(predator.energy[0] + 10); // gives predator arbitrary amount of energy;
                prey.setenergy(0); // kills prey by setting energy to 0.
            }
            System.out.println("Predator is at " + predator.energy[0] + " energy."); // keeps track of energy levels.
            System.out.println("Prey wins!");
        }
        else if(prey.energy[0] <= 0) {
            System.out.println("Aww. You ran out of putties.");
            break;
        }
        else System.out.println("Prey wins!");
        break;
    }
    predator.setenergy(predator.energy[0] - predator.energyexpend[level]*timestep); // creature loses energy based on energy expenditure and timestep
    prey.setenergy(prey.energy[0] - prey.energyexpend[level]*timestep);
    predator.position = randommovement(predator.position, predator.speed, timestep);
}


```java
public static double[] randomMovement(double[] movepos, double speed, double timestep)
{ // random movement for a creature restricted by movement speed.
    Random generator = new Random(); // use random number with different seed each time.
    double random1, random2, random3; // store random number.
    random1 = 2*(generator.nextDouble()-0.5); // allow random number to be in negative or positive.
    movepos[0] = movepos[0] + speed/timestep * random1;
    random2 = 2*(generator.nextDouble()-0.5);
    movepos[1] = movepos[1] + speed/timestep * random2 * (1-absDouble(random1));
    random3 = 2*(generator.nextDouble()-0.5);
    movepos[2] = movepos[2] + speed/timestep * random3 * (1-absDouble(random1)) * (1-absDouble(random1)); // creature doesn't have to move the full distance it can.
    return movepos;
}

public static double findDistance(double[] movepos, double[] goal)
{ // function that finds distance due to its prevalence in coding.
    double x = movepos[0] - goal[0]; // space between creature and goal in x
    double y = movepos[1] - goal[1]; // in y
    double z = movepos[2] - goal[2]; // in z

    double distance = Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2) + Math.pow(z, 2)); // the magnitude of distance.
    return distance;
}

public static double[] movement(double[] movepos, double speed, double[] goal, double distance, double timestep)
{ // movement towards a goal for a creature. Movement is optimized to remove the distance from the creature and its goal with a straight line.
    double x = movepos[0] - goal[0]; // space between creature and goal in x
    double y = movepos[1] - goal[1]; // in y
    double z = movepos[2] - goal[2]; // in z

    if (distance >= 0 && (absDouble(distance) <= speed/timestep)) { // in case predator would move too far.
        movepos[0] = goal[0];
        movepos[1] = goal[1];
        movepos[2] = goal[2];
    } else { movepos[0] = movepos[0] - (x/distance) * (speed/timestep); // position is changed by difference times speed over total distance.
        movepos[1] = movepos[1] - (y/distance) * (speed/timestep);
    }
    return movepos;
}

public static boolean kill(double distance)
{ // checks to see if predator is reasonably close to kill prey.
}
```
```java
boolean killed;
if (distance <= 2) killed = true; // arbitrary kill value.
else killed = false;
return killed;
}
public static double absDouble(double number) { // absolute value of double function.
    if (number < 0) number = -1 * number;
    return number;
}
```

**Reality 4 (Vegetation and Prey hunting Introduced)**

```java
package survival;
import java.util.Random;
/** This class represents an environment that a predator goes through searching for prey.
Creatures are randomly assigned
* traits (except for the predator). Prey finds vegetation to eat and doesn't deplete it. The Predator
finds the prey
* to survive.
* */

public class Reality4 extends Reality3 {
    public static void main(String[] args) {
        int numprey = 10; // number of prey to be created.
        int numflora = 20; // number of flora to be created.
        Creature predator; // declares new creature predator.
        Creature[] flora = new Creature[numflora]; // declares array of creatures flora.
        Creature[] prey = new Creature[numprey]; // declares array of creatures prey.
       Random creaturegen = new Random(); // generator for creature random traits
double sensevariance = 3; // multiplier of random traits multiplied by .5 due to balancing
double speedvariance = 6;
double positionvariance = 100;
for(int i = 0; i < numflora; i++) { // creates flora which have no senses or movement speed.
    flora[i] = new Creature();
    flora[i].setPosition(0 + (creaturegen.nextDouble() - .5) * positionvariance,
0 + (creaturegen.nextDouble() - .5) * positionvariance,
0 + (creaturegen.nextDouble() - .5) * positionvariance);
}
for(int i = 0; i < numprey; i++) { // creates prey
    prey[i] = new Creature();
    prey[i].setSenses(30 + (creaturegen.nextDouble() - .5) * sensevariance,
20 + (creaturegen.nextDouble() - .5) * sensevariance, 20 + (creaturegen.nextDouble() - .5) * sensevariance,
2 + (creaturegen.nextDouble() - .5) * sensevariance);
    prey[i].setSpeed(3 + (creaturegen.nextDouble() - .5) * speedvariance);
    for(int j = 0; j < numflora; j++) { // predators hunt
        if (Math.abs(prey[i].getPosition().distance(flora[j].getPosition())) <= 2) {
            System.out.println("Predator " + i + " found prey " + j + ".");
            prey[i].setDead(true);
        }
    }
    for(int j = 0; j < numflora; j++) { // flora doesn't die
        flora[j].setDead(false);
    }
    System.out.println("Flora: " + Arrays.toString(flora));
    System.out.println("Prey: " + Arrays.toString(prey));
    System.out.println("Survivor: " + survival.getNumSurvivors());
    System.out.println("Total: " + survival.getCount());
}
```
prey[i].setPosition(0 + (creaturegen.nextDouble() - .5) * positionvariance, 0 + (creaturegen.nextDouble() - .5) * positionvariance, 0 + (creaturegen.nextDouble() - .5) * positionvariance);
prey[i].setEnergy(10 + (creaturegen.nextDouble() - .5) * speedvariance);
}

double timestep = .5; // passage of time in seconds

double[] distance = new double[numflora]; // distance between predator and prey.

These roles now can be switched depending on chase.
double closestdistance = 1000; // distance that predator decides to follow.
int selectedprey = 0; // number in array of prey that predator's following.
int selectedflora = 0; // number in array of flora that prey is following.
int numofsteps = 100; // # of time iterations in this program.
int level = 0; // energy state of creatures. 0 is default and is at rest.
predator.setEnergy(10); // predator's traits.
predator.setPosition(3, -10, 10);
predator.setSpeed(10);
predator.setSenses(50, 10, 20, 2);

for (int i = 0; i < numprey; i++) {
}

for(int i = 0; i < numofsteps; i++) { // timeloop
    for(int j = 0; j < numprey; j++) { // finds distance between predator and prey
        distance[j] = findDistance(predator.position, prey[j].position); // finds closest prey for predator to go after
        if(absDouble(distance[j]) < closestdistance && prey[j].dead == false) {
            closestdistance = distance[j];
            selectedprey = j;
        }
        if (sensed(distance[j], prey[j].senses)) {
            level = 2; // prey is at full speed.
            prey[j].position = movement(prey[j].position, prey[j].speed, predator.position, -distance[j], timestep); // prey escapes if predator is sensed
        }
    }
    else {
        for(int k = 0; k < numflora; k++) { // loop for prey finding food if predator is not sensed
            distance[k] = findDistance(prey[j].position, flora[k].position); // finds closest flora for prey to go after
            if(absDouble(distance[k]) < closestdistance) {
                closestdistance = distance[k];
                selectedflora = k;
            }
        }
        if (sensed(closestdistance, prey[j].senses)) {
        }
}
level = 1;
prey[j].position = movement(prey[j].position,
prey[j].speed/2, flora[selectedflora].position, distance[selectedflora], timestep);//prey goes to food
if(kill(closestdistance)) prey[j].addenergy(5);
}
else {
  if (prey[j].energy[0] <
prey[j].maxenergy[0]/2){//checks for prey's energy.
    level = 1;
    prey[j].position =
randommovement(prey[j].position, prey[j].speed/2, timestep);//prey randomly walks if no flora is found
    level = 0;//prey is at rest.
  }
}
prey[j].subtractenergy(prey[j].energyexpend[level]*timestep);//prey's energy is taken
away depending on what it did.
}

if(prey[selectedprey].dead == false) {prey[selectedprey].dead =
kill(closestdistance);//checks to see if predator kills prey.
if(prey[selectedprey].dead){
predator.addenergy(10);//gives predator arbitrary amount of
energy;
prey[selectedprey].setenergy(0);//kills prey by setting energy to
0.
}
}
System.out.println("Predator is at " + predator.energy[0] + 
energy.");//keeps track of energy levels.
System.out.println("Selected Prey is " + selectedprey + "+" and is at "+
prey[selectedprey].position[2]);
if ( sensed(distance[selectedprey],
predator.senses)) {distance[selectedprey] = finddistance(prey[selectedprey].position,
prey[selectedprey].position);
predator.level = 2;//predator is at full speed.
predator.position = movement(prey[selectedprey].position, predator.speed,
prey[selectedprey].position, distance[selectedprey], timestep);//if sensed creature reacts
}
else {
  if (predator.energy[0] < predator.maxenergy[0]/2){//predator is finding food at half speed.
    level = 1;
    predator.position = randommovement(prey[selectedprey].position,
predator.speed/2, timestep);//else random movement happens
  }
  level = 0;//predator is at rest.
}
predator.subtractenergy(predator.energyexpend[predator.level]*timestep); // creature loses energy based on energy expenditure and timestep
  System.out.println("Predator is at " + predator.position[0] + ", " + predator.position[1] + ", " + predator.position[2]); // shows predator movement over iterations
  if (predator.energy[0] <= 0) {
    System.out.println("Predator is dead in " + i + " timesteps.");
    break;
  }
  }
  // testing
  for (int i = 0; i < numflora; i++) {
  }
  for (int i = 0; i < numprey; i++) {
  }