

Computer Systems Lab Senior Research  
Project Posters  
2007-2008  
2nd Quarter vers.

TJHSST

May 8, 2008

Projects  
7th Period Project Posters

# Music Analysis

Josiah Boning

TJHSST Senior Research Project  
Computer Systems Lab, 2007-2008

## Abstract

Although music is one of the most universal aspects of human culture, it is very difficult to define. Most definitions of music have been dependent on attributes such as rhythm, melody, and harmony, which are extremely subjective, so the ability to identify music has been limited to humans. This project aims to better define "music" by applying machine learning techniques to music analysis and recognition, allowing computers to autonomously identify whether a given audio sample is musical in nature.

## Background

Computers have already been used to perform analysis of music. Research has shown that different genres of music can be distinguished by fractal dimension and that machine learning techniques could successfully identify musical genres[2][1]. Other research has attempted to deconstruct music in terms of rhythmic and melodic patterns, and even looked at writing software to generate music conforming to such patterns[3]. However, each instrument has a different sound quality, and composers write music with these timbral differences in mind. Simply analyzing the notes on sheet music precludes the use of these differences in the analysis. Audio recordings, in contrast, allow analysis of exactly what the composer intended his audience to hear.

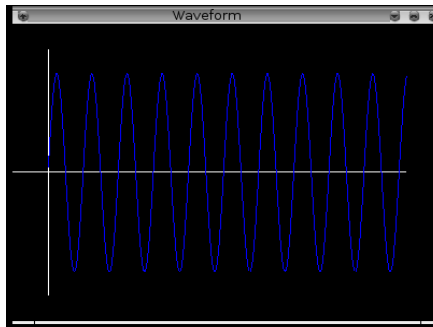
## Works Cited

[1] Basili, Roberto, Alfredo Serafini, and Armando Stellato. 2004. "Classification of Musical Genre: A Machine Learning Approach." Presented at the 5th International Conference on Music Information Retrieval.

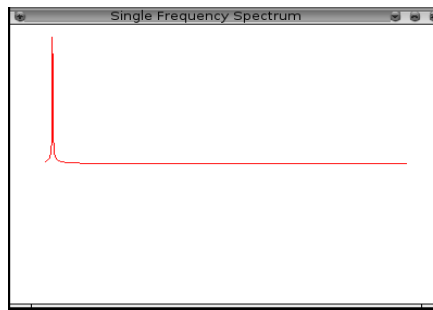
[2] Bigerelle, M., and A. Iost. 2000. "Fractal Dimension and Classification of Music." *Chaos, Solitons & Fractals*. 11(14):2179-92.

[3] Leach, Jeremy, and John Fitch. 1995. "Nature, Music, and Algorithmic Composition." *Computer Music Journal*. 19(2):22-23.

## Waveform



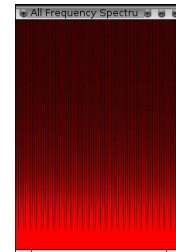
## Single Fourier Transform



## Fourier Transforms

A Fourier transform decomposes a wave into its constituent frequencies. A sine wave (left) has a Fourier transform with a spike at a single frequency (below left). The transforms can also be performed over several time windows (below).

## Multiple Fourier Transforms



## Fractal Dimension

The equations below are calculated by numeric methods to yield the fractal dimension of the audio waveform. Since different genres of music are distinguishable by their fractal dimensions, it is reasonable to suspect that music itself might be distinguishable by its fractal dimension[2]. The equations are evaluated over discrete audio data, so their accuracy will be increased by the use of cubic splines to interpolate between data points and allow smaller differential values when doing numerical integration.

$$\lim_{\tau \rightarrow 0} 2 - \frac{\log \left( \frac{1}{b-a} \int_a^b \left| \max_{|x-t| < \tau} (f(t)) - \min_{|x-t| < \tau} (f(t)) \right| dx \right)}{\log \tau}$$
$$\lim_{\tau \rightarrow 0} 2 - \frac{\log \left( \frac{1}{b-a} \int_{x=a}^{x=b} \left[ \frac{1}{\tau^2} \int_{t_1=0}^{\tau} \int_{t_2=0}^{\tau} |f(x+t_1) - f(x-t_2)|^\alpha dt_1 dt_2 \right]^{1/\alpha} dx \right)}{\log \tau}$$

# Implementation of an Artificial Neural Network Library in C

Jack Breese  
TJHSST Computer Systems Lab  
2007-2008

## Abstract

This project aims to implement a general purpose library for neural networks in the C programming language. This library will be well-suited to basic object and pattern recognition in images, from optical character recognition to shape classification, as well as simple face recognition.

## What are Neural Networks?

An artificial neural network is a computational model which emulates the biological structure of the brain. It is a system of interconnected virtual neurons which are capable of modifying their connections, adapting their responses based on accuracy. This modeling of biological networks has widespread use in the field of pattern recognition and object classification, and is well suited to tasks such as optical character recognition and junk email filtering.

## Implementation

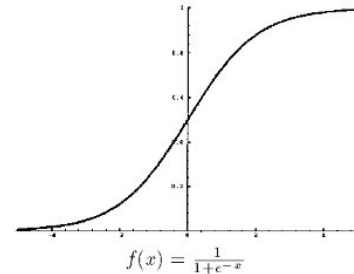
The library for neural networks is entirely implemented in C. There are several structs (shown below) which allow for a simpler way of keeping track of the network and iterating through it. Methods have been implemented for initializing networks of arbitrary size, testing said networks, performing calculations on the networks, and saving and loading the current states of the networks.

## Current Running Behavior

When run, my program initializes a network of specified size, and then sets the weights on each connection to either a specified or random value. It then saves the network to a specified filename, and deallocates the memory. It then reads in the values stored in the file, and initializes a new network from that data. The `initNetFromFile` method then returns an array of arrays of neurons, which contains each of the layers, and the values of this newly initialized network are checked, and then saved to a different filename. The makefile which runs my program then runs diff on these two files to verify that no data has been lost in the network saving/loading process.

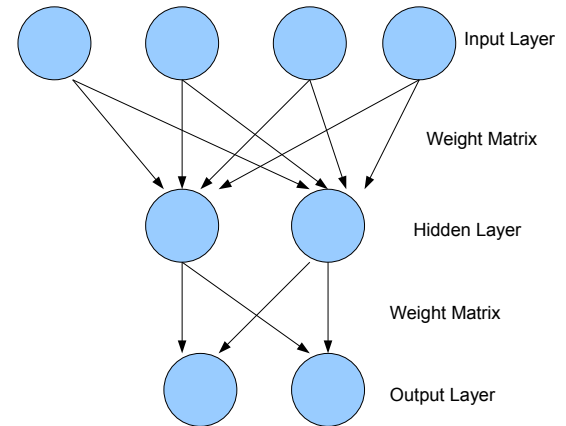
## Sample Code

```
typedef struct _connection {
    float weight;
    struct _neuron * from;
} connection;
typedef struct _neuron {
    float d;
    connection * cons;
} neuron;
neuron* mkneuron(int c) {
    neuron* n = malloc(sizeof(neuron));
    n->d = 0;
    connection * a = malloc(c*sizeof(connection));
    n->cons = a;
    return n;
}
int saveWeights(char* filename, neuron* hidden, neuron* outputs, int insize,
int hiddensize, int outsize) {
    FILE* output;
    output = fopen(filename, "wb");
    if(output == NULL) {
        fprintf(stderr, "Error: Unable to open output file for writing.");
        exit(1);
    }
    fprintf(output, "%d\n", insize);
    fprintf(output, "%d\n", hiddensize);
    fprintf(output, "%d\n", outsize);
    int i = 0;
    int j = 0;
    for(; j < hiddensize; j++) {
        for(i=0; i < insize; i++) {
            fprintf(output, "%f\n", hidden[j].cons[i].weight);
        }
    }
    i = 0;
    j = 0;
    for(; i < outsize; i++){
        for(j=0; j < hiddensize; j++){
            fprintf(output, "%f\n", outputs[i].cons[j].weight);
        }
    }
    fprintf(output, "%s\n", "[End]");
    fclose(output);
    return 0;
}
```



## 1.1 The Sigmoid Function

## 1.2 An Example Two-Layer Perceptron Neural Network



## Sample Program Run

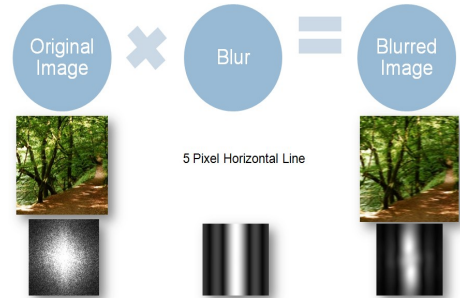
```
./fr
Debug: Successfully allocated memory for neural
network.
Value of weights of the hidden layer in Network 1:
0.450000
Saving neural network, please wait...
Successfully saved neural network.
Successfully deallocated memory for network.
Initializing network from file...
Successfully read in file and initialized empty
network.
Value of weights of the hidden layer in Network
initialized from file after network deallocation:
0.450000
Saving neural network, please wait...
Successfully saved neural network.
Run diff testfile.test testfile2.test to check for
differences between the first and reinitialized
networks.
diff testfile.test testfile2.test
```

# Implementation of Image Deblurring Techniques in Java

Peter Chapman

## Abstract

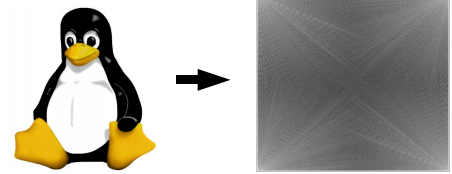
Countless numbers of photographs are taken every day, and inevitably, many images suffer from some sort of "blurring." A program with the power to take a blurred image create a much crisper and clearer "deblurred" form would be immensely valuable. Law enforcement making out a blurred photo of the getaway car's license plate, or even a family attempting to improve the clarity of their grandfather's smile would find such a piece of software useful. In my implementation I attempt to deblur images suffering from simple types of motion blur using the alternate domains granted by the use of Fourier transformations and a basic understanding of image deconvolution.



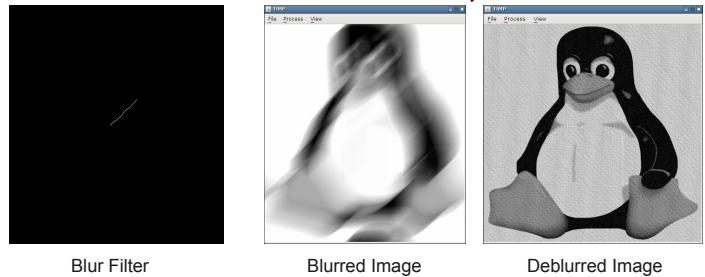
## Background

In order to reverse the blur on an image, it is necessary to approach the task mathematically. If the process that blurs the image is considered a mathematical function, it must be reversed in order to restore the image; however, to do so, it is necessary to understand how the image was blurred, characteristics such as direction, type (motion, out of focus image, etc.), and magnitude. The best way to approach such a complex task is to convert the image into a different domain. The way in which we normally view images is known as the spatial domain, but if the image is converted into a series of sin functions through a mathematical technique known as a Fourier transformation it is possible to view the image in the frequency domain. Once in the frequency domain, it is now possible to perform more advanced analysis on the image and perform mathematical operations in a more generalized way. It is understood that using the Fourier transformation of an unblurred image and the Fourier transformation of the blur (a five pixel horizontal line corresponds to a five pixel blur) with a process known broadly as image convolution produces the Fourier transformation of the blurred image. Thus, by performing the inverse, a deconvolution on the image, the unblurred image can be restored. The most difficult part of this process is determining what the "blur factor" was when the picture was taken. In theory, if one can determine how the image was blurred, it is possible to unblur the image.

### Fourier Transformation



$$F(u, v) = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x, y) e^{-2\pi i \left( \frac{xu}{N} + \frac{yv}{M} \right)}$$



## Procedure

### Rendering Fourier Transformations

The first step is to render the blurred image in the frequency domain using the Fourier transformation. The general formula for a Fourier transformation requires a continuous function. Since an image can seldom be represented as a continuous function, it is necessary to treat the image as a set of values in a limited domain. Using the formula for the discrete Fourier transformation it is possible to render the Fourier transformation of the image. A 2D discrete Fourier transformation requires every single point to perform a calculation on every other point, resulting in an extremely slow  $O(N^3)$ . As a result it is necessary to use a faster implementation of the Fourier transformation. The Fast Fourier transformation (FFT) is a process that allows one dimensional data sets to be rendered in the frequency domain in  $O(N \log N)$  time. Since the sums in the discrete Fourier transformation can be separated, a two dimensional Fourier transformation can be a rendered quickly by applying an FFT to the rows and then to the columns. The speed of the FFT is derived from the symmetric nature of the Fourier transformation, allowing much fewer calculations to be made on the data thus decreasing the run-time.

The inverse of the FFT, a step necessary for returning the deblurred image back to spatial domain, is easily performed by essentially taking the conjugate of the image in the frequency domain, realizing that the data resulting from the FFT is a series of complex numbers; then performing an FFT; and finally calculating the conjugate once again. The result of the entire process results in a significance amount of noise for which must be compensated.

## Blurring and Deblurring

Although time consuming, properly rendering the Fourier transformations work at the heart of the blurring and deblurring process, regardless of the method used. As a starting point, I decided to utilize the simplest image deblurring method known as inverse filtering. As discussed earlier, blurring, at its most basic level, is the multiplication of the Fourier representation of the image and the Fourier representation of some other filter. By dividing, while ensuring that one does not divide by zero. The result, shown above is fairly accurate, with the exception of a series of wave-like lines filling the image. This method must have a deblur filter that is nearly exactly the same as the filter used to blur the image. More advanced methods of deblurring calculate, or more accurately guess, the original blur filter; my application is not as advanced, but works well enough to experiment with the effect of certain filters on certain images. The next step will be to attempt to perform more advanced techniques of image deblurring.

## Abstract

This project aims to create a decompiler capable of processing outputted Java 6 bytecode into fully-recompilable and functionally-equivalent source code.

# Java 6 Decompiler

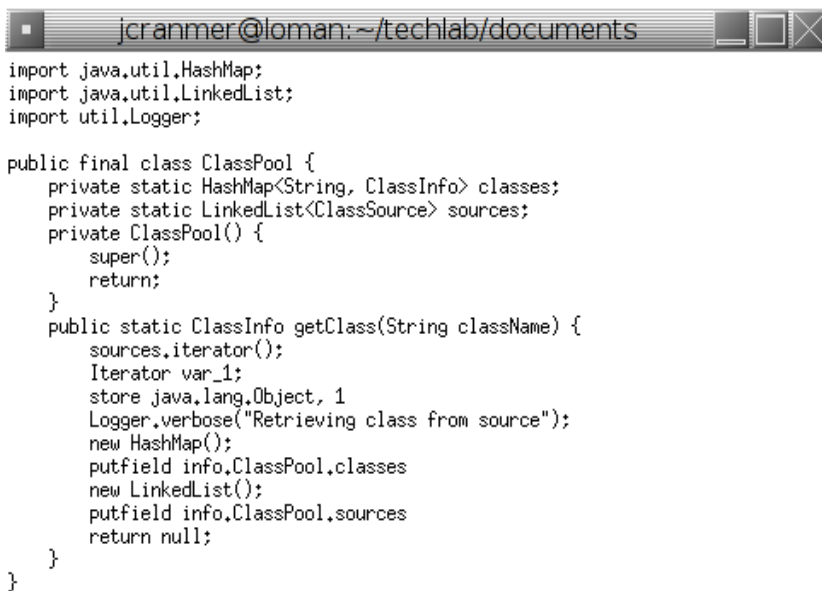
Joshua Cranmer  
TJHSST Computer Systems Lab  
2007-2008

## Reasons for Decompilation

- Finding bugs in program
- Finding vulnerabilities
- Finding malware
- Compiler code verification
- Comprehending algorithms
- Creating interoperability
- Induce customizability
- Porting code
- Create maintainable source code
- Fixing bugs without patching binaries
- Add features to a program

## Procedures and Methods

The decompiler works in a multi-phased approach. First, the class file is fully parsed and stored in memory. Then, the code execution bodies are processed through several transformation filters until readable source code is produced. Next, various filters are applied to make the source code more readable. Finally, everything is fully decoded and then printed out into class files.



```
jcranmer@loman: ~/techlab/documents
import java.util.HashMap;
import java.util.LinkedList;
import util.Logger;

public final class ClassPool {
    private static HashMap<String, ClassInfo> classes;
    private static LinkedList<ClassSource> sources;
    private ClassPool() {
        super();
        return;
    }
    public static ClassInfo getClass(String className) {
        sources.iterator();
        Iterator var_1;
        store java.lang.Object, 1
        Logger.verbose("Retrieving class from source");
        new HashMap();
        putfield info,ClassPool,classes
        new LinkedList();
        putfield info,ClassPool,sources
        return null;
    }
}
```

Example screenshot of running code. Note the use of proper indentation and (not seen here) proper 80-character overflow.

Generic signatures are decompiled, as well as the recovery of new variables, and the decompilation of certain simple bytecodes.

# Evolving Motor Techniques for Artificial Life

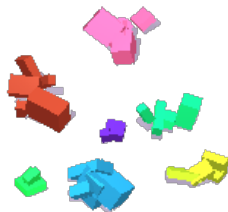
Kelley Hecker

TJHSST Computer Systems Lab 2007 - 2008

## Abstract

I have created a program for simulating unique creatures in a 3D environment using co-evolution of the creatures' mental and physical structures. Creature data is stored in a one-dimensional genome consisting of various nodes for each physical body segment. The brain of the creature is controlled by neuron modification of sensor inputs. There is a system for converting genomes to physical representations to allow for physical simulation in the environment, and eventual selection of prime candidates through a genetic algorithm.

Examples of possible creature genomes. When each creature is displayed, the genome tree must be converted to an object.



## Development

The entire simulation will be run by the Controller object. At the beginning of each simulation the Controller will create an array of genomes, and maintain this array throughout. It also displays the creatures in the physical environment and measures their fitness levels. After the fitness levels are compared, the Controller will manage the reproduction of the creatures and update the genome array with the next generation.

Each genome is made up of several nodes, each representing a body segment in the physical creature. The nodes store physical dimensions for the limb, a list of connected limbs or children, points where the segment connects to its parent and children segments, and the neurons which will control the joints.

At each time-step the joint-angle values for each node are measured and passed to the Creature Genetic Algorithm. This algorithm passes the sensor values through the neurons for that node and produces an effector, which will be the joint velocity. Possible neuron functions are sin, cos, atan, sum-threshold, sign-of, min, max, if, mem, saw-wave, log, expt, divide, interpolate, and differentiate.

## Background

### Related Research

Research done by Karl Sims is very similar to what I wish to accomplish. His work with evolving creatures led to a variety of organisms specialized in different areas and had very organic movements. The creatures were neuron controlled.

Yoon-Sik Shim and Chang-Hun Kim continued Sims' research and explored the possibility of flying creatures. They developed and explained a system of storing genomes as one-dimensional arrays.

### Methodology

The simulation stores creatures in genomes in a way similar to Shim and Kim, however rather than being an array the structure is more like a tree. Creatures are controlled by passing joint-angle velocities received from sensors through neurons to produce joint velocity values. The data follows a circular pattern, moving between sensor, neurons, and effectors (output) each timestep.

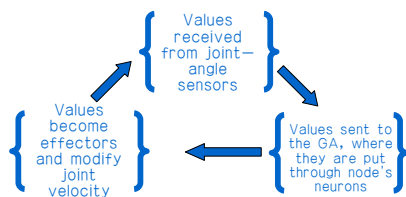
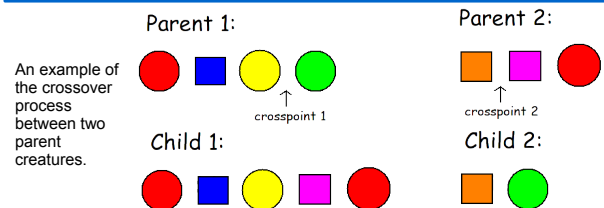


Image representing the circulation of data. Data originates from sensors measuring interaction with the environment, and then is modified by neurons within each node. Finally, the modified data is passed to the joint as velocity values. The cycle starts over again when sensors receive new data after the joints move.

## Evolution

At the end of a generation, the best creatures are chosen based on their fitness value, which is how far they have moved since the start of the simulation. The top 20% of creatures are reproduced asexually (copied directly to the next generation). The remaining creatures are crossed over to produce new offspring.



## Expected Results

My final goal is to create a simulation that can create a unique variety of creatures with advanced motor techniques. Since creatures are stored as genomes it should be easy to allow for both mental and physical evolution.

There is also a possibility for specialized creatures. Different fitness tests could be implemented to select creatures which excel at different techniques, such as swimming or jumping.

# First-Person PacMan

by Brett Jones  
TJHSST Computer Systems Lab 2007-2008

## Abstract

The purpose of this project is to create a 3D, first-person version of the classic PacMan arcade game in order to learn more about the concepts of 3D graphics programming and rendering algorithms. The project will also include a basic AI to control the ghosts.



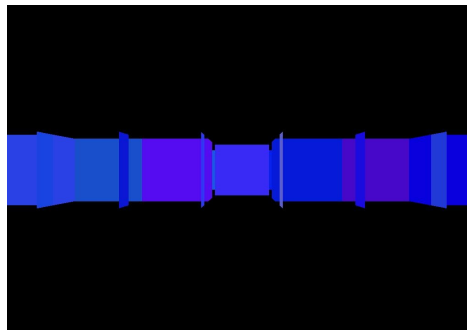
An initial view of the scene, slightly raised and with wall colors differentiated.

## Background

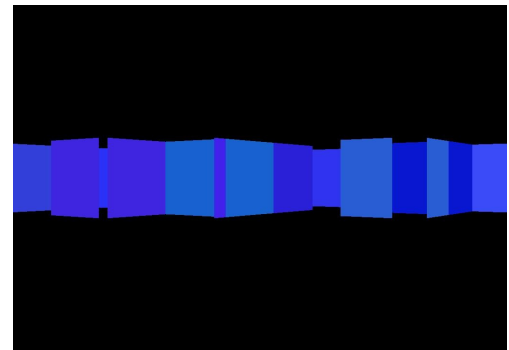
The field of 3D computer graphics has been explored quite extensively, and comprises of three major parts: 3D modeling, animation, and 3D rendering. The first part, 3D modeling, refers to creating a 3D representation of an object. Animation, the second part, is moving the object through time. The final part, 3D rendering, is drawing the animated 3D model to the screen. 3D rendering is the most complex of the three parts, and is accomplished through several algorithms: polygon modeling, ray tracing, ray casting, or scanline rendering. This project will use the ray tracing algorithm, which casts a ray from the eye through each pixel of the virtual screen to the environment, calculating the length of the ray and using that to determine view distance, and using the piece of the environment the ray intersects to determine what to display.

## Progress

Currently, the program is coded to run in fullscreen exclusive mode (FSEM) in order to display the game over the entire screen. The program runs without errors and displays the scene objects, and the view can be rotated. The menu consists of a title image and seven function buttons: New Game, Control, Sound, Save Game, Load Game, High Scores, and Quit. Quit exits the program, New Game creates an instance of the World class (which extends Frame) and sets the program to run in FSEM with the World class as the viewable display, and the other buttons do not have any coded functionality. The program displays a black background with randomly shaded blue cubes (the wall objects) connected in the fashion of contiguous walls, and accepts keyboard input for motion and returning to the main menu. The move method currently generates runtime errors, but the turnLeft and turnRight methods work appropriately.



A screenshot of the scene's initial view, as of 04/02/08.



A screenshot of a rotated view in the scene.

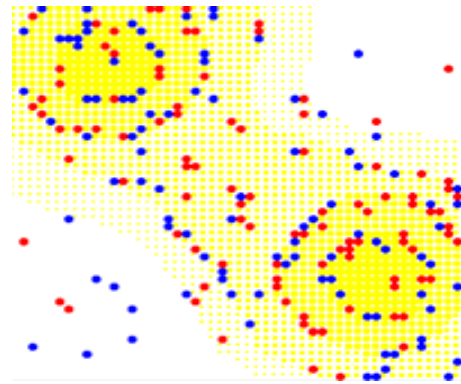


# Examining Leadership Dynamics in Agent Based Modeling

By Alex McGuigan

## Abstract

The project attempts to explore leadership dynamics in Sugarscape. The goal is to discover which methods are most frequently used in group formation, which leadership traits form the best groups, and which traits are valuable in followers. This topic was not addressed entirely by Sugarscape, and thus is a good topic for a Syslab project. In addition, Sugarscape spends very little time on combat, and this project intends to fill this gap as well.



## Introduction

I expect to obtain results regarding the original layout of stats. I suspect that Wealthy groups will be the largest, and Intelligent groups will be the best able to survive.

## Background

This area is dominated by the book *Growing Artificial Societies*, which was written about Sugarscape by the creators of Sugarscape. The Sugarscape model is the state of the art model in agent based modeling currently. I could also adapt features from the many agent based modeling programs created in Swarm or MASON. However, I still need more research that is not directly related to Sugarscape.

## Development

I have completely coded the combat, vision, and judgment system for the agents. In addition, I have made some progress on the coding for the group functionality. However, I have encountered a number of bugs in the programs that determine the sugar and opponents on surrounding squares, as well as parts of the movement system. Until these bugs are resolved, I cannot continue coding the group functionality.

## Preliminary Results

Due to the bugs in the combat system, I have been unable to test my combat system. Once I eliminate these bugs, then I should be able to test the veracity of my combat algorithm. I want to examine the presence of wealth in my combat model primarily.

# Simulation of the Spread of a Virus Using Agent Based Modeling

Matt Wade

## Abstract

My goal is to make an agent based modeling simulation that shows the spread of a cold through a school. It will start with an amount of infected students and healthy students received as inputs and will show how much the virus spreads or possibly recedes over time. The program will answer the question as to how quickly and fully different types of sicknesses will be able to spread through the population of a school once introduced. This will show how likely it is for a disease to be spread by a set amount of sick people coming to school with the sickness. This will show if the danger of infecting others is actually a valid excuse not to come to school or if you should come to school unless you actually don't feel like you will be able to do work.

Info	Comments	Room 1	Room 2	Room 3	Room 4	Room 5
Day: Period:						
# of Agents: # Sick:		Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0
# Healthy:						
Total Infections:						
Total Recoveries:						
		Room 6	Room 7	Room 8	Room 9	Room 10
		Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0
		Room 11	Room 12	Room 13	Room 14	Room 15
		Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0
		Room 16	Room 17	Room 18	Room 19	Room 20
		Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0
		Room 21	Room 22	Room 23	Room 24	Room 25
		Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0	Healthy: 0 Sick: 0

## Procedures

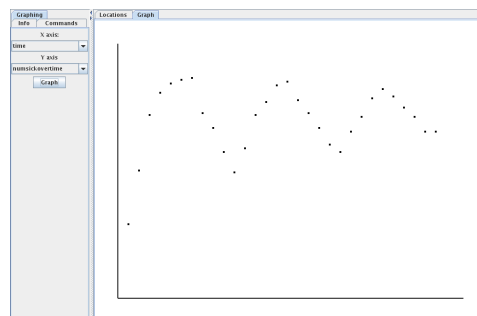
My program has three main classes. An Agent class which defines what values an Agent will store and how to construct it. A class that creates the GUI. And the most complicated of the three, the Model class which is where all of the calculations occur. In the Model class there is an arraylist containing all of the Agents in the simulation. With these Agents the class has to define a step() function which moves the simulation forward. This function has to update the locations of all of the agents, check to see if any of them get infected, and check to see if any recover from being sick. First it goes through the list of Agents and moves them all to the next location in their schedule. In order to check for any new infections it goes through the entire list of Agents finding each sick Agent. Whenever it finds a sick Agent it finds any healthy Agents in the same location and checks a randomly generated number against the sick Agent's infectiousness value. If the random number is lower than the healthy Agent is switched to sick and the method continues on through the rest of the list. To check if any Agents recover from sickness it goes through the list checking the recoverytime value and if it equals zero the Agent is switched to healthy. The GUI class contains another important function, the ability to graph data gathered through the simulation. In the Model class I added new arraylists to retain information over time for all of the main variables (number of sick agents, number of healthy agents, infections per step, etc.). Whenever the graphing method is called the Model class gets information from two drop down menus as to which arraylists are going to be the x and y axis variables and sends them to the graph class. The graph class then takes these variables and goes through the arraylist graphing a scatterplot of the data.

## Expected Results

My program will answer the question as to how quickly and fully different types of sicknesses will be able to spread through the population of a school once introduced.

This will show how likely it is for a disease to be spread by a set amount of sick people coming to school with the sickness. This will show if the danger of infecting others is actually a valid excuse not to come to school or if you should come to school unless you actually don't feel like you will be able to do work.

At the moment my program is pretty much in its final state in terms of the actual simulation. All of the methods relating to the simulation, such as step(), checkinfection(), or checkrecovery(), are all completed and working as they should. This means that the results I am getting right now are pretty much the same results I will have when the whole project is completed except for any analysis tools that I plan to add to the program, such as a picture showing the locations of all of the Agents, and a graph that shows the number of sick and healthy Agents over time. As of now though, it will give you data on the number of sick and healthy agents, the total number of infections and recoveries, the number of steps taken, and the locations of all the agents.

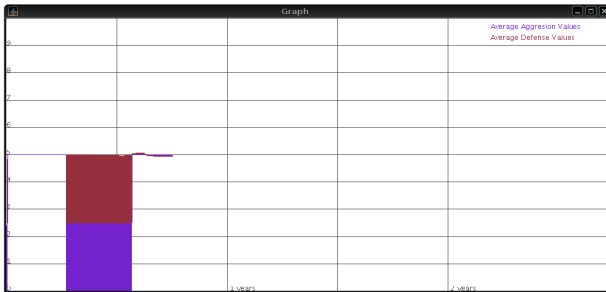
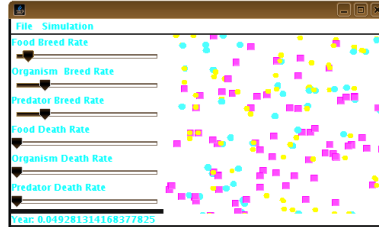


# Simulating Evolution

by Tasha Wallage  
TJHSST Computer Systems Lab  
2007-2008

## Abstract

The main purpose of this program is to accurately simulate the genetic evolution of a species. It will attempt to do so using methods such as genetic mutation, genetic drift, and natural selection by means of both microevolution and macroevolution.



## Background

"Genetic changes do not anticipate a species' needs and those changes may be unrelated to the selection pressures on the species. Nevertheless, evolution is not a fundamentally random process."

### Agent-Based Modeling

The actual evolution simulator is an ABMS with the Organism and Predator classes being the 'agents.' An agent is "autonomous and self-directed." It can "function independently in its environment and in its dealings with other agents." Mostly, an ABMS focuses on the interactions between the agents. In this project, I will be observing both the interactions between the agents and the interaction between the agents and their environment.

### Basic Concepts

A population of any given species is greatly affected by its environment. This is where an animal will get its food and raise its young. In order to do this, it has to be well adapted to the environment it lives, yet also able to change under stress (such as a change from the norm). This is when evolution will occur. The members of a species that are best able to handle stress are the ones that will live on to populate the species; therefore, their young will acquire the "better" traits and be able to live in the newly changed environment. The environment in which a population lives provides resources for the population such as food and shelter. If there is limited food, then the environment will only be able to support a given number of species, meaning that the population will have a max value. The function of the population over time should be logarithmic, approaching that max value. However, this is just a basic model of an environment, void of predators and many other factors that affect the size of the population. If there are predators, then the population size should oscillate in accord with the predators (though there is a slight lag in the predator's population graph).

## Procedures/Methodology

### Steps to Simulating Evolution

- 1) Create a changing environment with which a species may interact
- 2) Create a food source for the species
- 3) Create a species with designated traits to be tracked
- 4) Possibly create an herbivorous species and a predator
- 5) Define how the species may evolve (genetic algorithms)
- 6) Track the changes in traits and make observations
- 7) Adjust the model until a balance is achieved

### Algorithms

#### 1) Process for Recombination

The process for creating a new organisms with a new combination of genes mixed from its parents (and sometimes randomly mutated) takes the traits from both parents and gives the child a trait that is either equal to one of the parents, or is a mix of the two (something in between). The assignment of the trait is semi-random.

#### 2) Randomization for Mutation

The process by which genes are mutated is completely random. In fact, it is double random because the swapping of genes is random and the chance that it is mutated is also random.

# Reinforcement Learning in Connect 4

Michael Yura  
2007-2008

## Introduction:

Although an AI is often thought of as being only as intelligent as its programmer, this is not exactly the case; this project will attempt to create a dynamically learning Machine Learner for Connect 4 by using supervised reinforcement learning, with each Learner saving the way that it will play into text files, each with the way that it will play for a given board layout.

## Background:

I expect to have an ML that throughly and hopefully quickly learns to play Connect 4 to an advanced level. Through this project, I hope to learn how fast and to what quality reinforcement learning allows for the learning of a simple game; these methods can hopefully be extended to other, more complex tasks for machines to learn.

Connect 4 has already been solved by James D. Allen and Victor Allis; I will attempt to compare the way the ML plays to the strategies outlined in Allis's *A Knowledge-based Approach of Connect-Four*

## Expected Results:

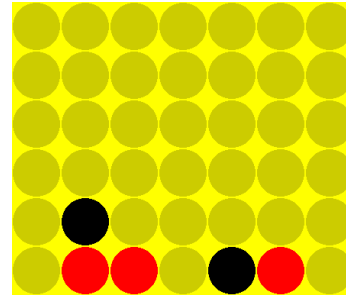
Through this project, I hope to find a degree of reinforcement learning that allows the computer to learn to play connect 4 quickly and thoroughly. Although I am quite sure that an ML that learns more progressively will in the end turn out to be better, it may not be the most efficient, due to the time and the size that it would take to create one that would surpass the abilities of an ML whose data is more hastily created. I hope that this project may add to the creation process of AI's.

## Procedures:

I have currently programmed the Connect 4 game itself, as well as created an "ML" (Machine Learner) abstract class that other ML's will be based upon, with methods to load save its board data. This ML appends all of its board data into a single text file and saves the corresponding probability data its own smaller file, which is rewritten after the ML plays a game.

I currently have an ML that does not change the way it places pieces, playing completely randomly; I plan to create ML's that will change the way they play to different degrees, some radically changing their strategies after each game, and others doing so to a more moderate degree. The way that each ML changes its strategy will be written by myself, meaning that this is not entirely independent learning, but Supervised Reinforcement Learning.

A board of:



Would be represented in the board data file as:

```
1 [0,0;0][0,1;0][0,2;0][0,3;0][0,4;0]
[0,5;0][1,0;1][1,1;2][1,2;0][1,3;0]
[1,4;0][1,5;0][2,0;1][2,1;0][2,2;0]
[2,3;0][2,4;0][2,5;0][3,0;0][3,1;0]
[3,2;0][3,3;0][3,4;0][3,5;0][4,0;2]
[4,1;0][4,2;0][4,3;0][4,4;0][4,5;0]
[5,0;1][5,1;0][5,2;0][5,3;0][5,4;0]
[5,5;0][6,0;0][6,1;0][6,2;0][6,3;0]
[6,4;0][6,5;0]
```

Similarly, a probability data file of:

```
[94.0,15.6,77.2,92.8,100.0,43.3,0.1,]
```

Would represent a:

```
94.0/423.0 (22.22%) chance of placing
in Column 0
15.6/423.0 (3.69%) chance of placing in
Column 1
77.2/423.0 (18.25%) chance of placing
in Column 2
92.8/423.0 (21.94%) chance of placing
in Column 3
100.0/423.0 (23.64%) chance of placing
in Column 4
43.3/423.0 (10.27%) chance of placing
in Column 5
0.1/423.0 (0.02%) chance of placing in
Column 6
```