# Exploring Tradeoffs with Evolving Animats in Dynamic and Static Environments

Viet T. Nguyen
*University of California, Los Angeles*

## Abstract

Although it is expected that evolving animat behaviors in dynamic environments will lead to more robust behaviors, it is not entirely clear if that robustness would trump an animat behavior evolved in a specific, static environment. This possibility is explored using a newly developed system for evolving animats controlled by neural networks in a complex and potentially dynamic environment. Animats are evolved in environments that are some permutation of restored/randomized on initialization and static/dynamic on simulation. Each resulting behavior is then tested in different environments and their performance measured.

## 1 Introduction

This project initially started off as an exploration of evolving animat behaviors in dynamic environments to improve adaptability. However, the premise behind that is uninteresting because the results of such a study are obvious. Instead we explore whether the additional adaptability of animats evolved in dynamic environments comes at a cost. More specifically, will adaptable animats perform better than animats evolved for a specific environment when placed in that specific environment?

The hypothesis is expanded on in Section 2 along with the methodology of testing and experimentation. Since it involves developing a fairly complex environment, the details of the environment design are explained in Section 3. Afterwards the design of the animats and their various components are outlined in Section 4 along with their evolution in Section 5. Preliminary experimental results are discussed in Section 7. For the code-curious implementation is detailed in Section 6. Finally we close with a discussion of the results and future experiments in Section 8 and Section 9.

## 2 Hypothesis

Evolutionary algorithms will locally optimize animat fitness to an environment, assuming that the evolutionary operators act on the factors that determine an animat's fitness in an environment. In the context of this project, fitness is determined by an animat's ability to survive in an environment; essentially how long the animat can survive (see section 4.2). We expect that animats evolved in dynamic environments will exhibit behavior that is more robust (better survivability) to change in comparison to animats evolved in static environments. Similarly, we also expect animats evolved in randomized environments to develop behavior that is more robust compared to animats evolved in an environment that remains the same across generations.

When animats are evolved in the same, static environment there is selectional pressure to specialize the behavior for that specific environment. Dynamic and random environments reduce that pressure because specialization can be brittle to changes. Despite the benefits of evolution in dynamic and random environments, we hypothesize that the robust, adaptive behavior comes at a fitness cost in specialized environments.

### 2.1 Methodology

To test this we will develop an animat-versus-environment simulation system. The system will be able to initialize the environment randomly or from a saved instance. The simulation will involve not only animat dynamics but environmental dynamics as well. This creates four environment permutations (and their reference characters):

- Randomized and dynamic (`rd`)
- Randomized and static (`rs`)
- Non-randomized and dynamic (`nd`)
- Non-randomized and static (`ns`)

Here randomized means the environment is initialized randomly while non-randomized means the environment is loaded from a saved instance. Dynamic refers to an environment that changes during simulation.

Separate animat populations will be evolved in each of these environment permutations. Experiments will then be run where the populations will be transplanted into different environment permutations and their performance evaluated.

## 2.2 Simulations

The simulation process is composed of three distinct phases: initialization, simulation, and evolution.

### 2.2.1 Initialization

Initialization creates the terrain, whether by randomization or instantiating a previous terrain save. It also resets the calamity counter and environmental globals such as water level and global temperature.

### 2.2.2 Simulation

Simulation involves the actual simulation ticks for environment dynamics, animat brain processing, and animat behavior execution. The simulation proceeds until all animats have been killed by the environment, 6000 simulation ticks have elapsed, or the simulation times out when an animat has not died in the last 500 simulation ticks.

A series of increasingly calamitous environmental changes occurs during simulation of dynamic environments. These environment changes involve modifying global environmental parameters such as water level and global temperature to mimic environmental calamities. At a periodic interval a random calamity will begin, selected from the following list:

- Flood
- Drought
- Heat wave
- Cold snap
- Famine

Each calamity's severity is dictated by a global parameter that is incremented with each calamity.

### 2.2.3 Evolution

Evolution occurs once all animats are dead. Truncation and roulette-wheel selection are used. There are three proportions used to generate the next population: proportion of "winners" that moves on, proportion of new population that will be composed of children of the "winners," and proportion of new population that will be randomized. While truncation selection is used to determine animats that will proceed, mating and child generation is done using roulette-wheel selection. Afterwards the population is padded to the full population count with randomized animats which act as a sort of agent-level mutation. The evolutionary operators involve cross-over and mutation at the genome and gene level (see section 4.3).

## 2.3 Experiments

The experiments essentially involve evolving a population in one environment permutation and transplanting it into another to compare with the "native" population. To do this we need to first evolve native populations. Afterwards we can take the populations and evaluate them in different environments.

### 2.3.1 Phase I: Evolving Native Populations

Native populations are evolved through a similar number of generations. The system is set up to save the genome of each animat in the population every 20 generations. Each population is evolved from the same population seed which is composed of 400 randomly initialized animats. This seed is kept as a control and baseline population.

### 2.3.2 Phase II: Transplant Populations

In order to test our hypothesis, we are interested in the following transplantations:

- Randomized and dynamic in non-randomized and static (`rd-in-ns`)
- Non-randomized and static in randomized and dynamic (`ns-in-rd`)
- Randomized and dynamic in randomized and static (`rd-in-rs`)
- Randomized and static in randomized and dynamic (`rs-in-rd`)

For brevity, we will refer to these transplantations with the reference strings in the parentheses.

Based on our hypothesis, we expect the following general results compared to the native population:

- `rd-in-ns`: slightly worse performance
- `ns-in-rd`: much worse performance
- `rd-in-rs`: better performance
- `rs-in-rd`: worse performance

In the `rd-in-ns` experiment we expect the randomized/dynamic population to perform slightly worse in the

non-randomized/static environment because of specialization by the native population specifically for that environment. This would be in spite of the much better ability of a randomized/dynamic population to adapt to environments that should be demonstrated by the `ns-in-rd` experiment.

The last two experiments, `rd-in-rs` and `rs-in-rd`, are meant to test whether evolving populations in dynamic environments truly produces more robust behaviors overall.

#### 2.3.3 Phase III: Compare Performances

Each evaluation of a population in an environment involves simulating the population in the environment 60 times. Each population contains 400 animats giving us with 24,000 total animat lifespans. The distribution of these lifespans are used to compare with other evaluations via three methods: mean, median, and visual histogram comparison. We expect a lower mean, lower median, and/or left-skewed distribution to mean poorer performance by a population in an environment.

## 3 Environment

The environment is 2.5 dimensional. It is represented as a two dimensional grid with height/altitude values associated with each cell. Although discrete, the cells are meant to approximate continuous dynamics. In addition to height, each cell also holds a number of other values associated with different layers. Figure 1 shows a visualization of how these layers come together to form the environment.

### 3.1 Layers

Local environmental information is represented as layers across the same two-dimensional grid. The heights are simply considered another layer, though the height layer is the most important. In fact, the environment is essentially determined by the height layer as all other environmental effects are dependent on it. No environmental effects alter the height layer (no erosion).

Other layers include temperature, moisture, vegetation, and animat density.

The implementation of these layers has been abstracted allowing for the quick addition of new environmental effects. The abstract layer implementation has built in functionality for symmetric interactions between neighboring cells as well as running functions on each cell individually. Animats operate in the continuous two-dimensional plane. Interaction with the environment's discrete two-dimensional grid involves interpolation. Simple bilinear interpolation [2] is built into the abstract layer implementation. Future refinements may include upgrading the interpolation scheme to bicubic [1].

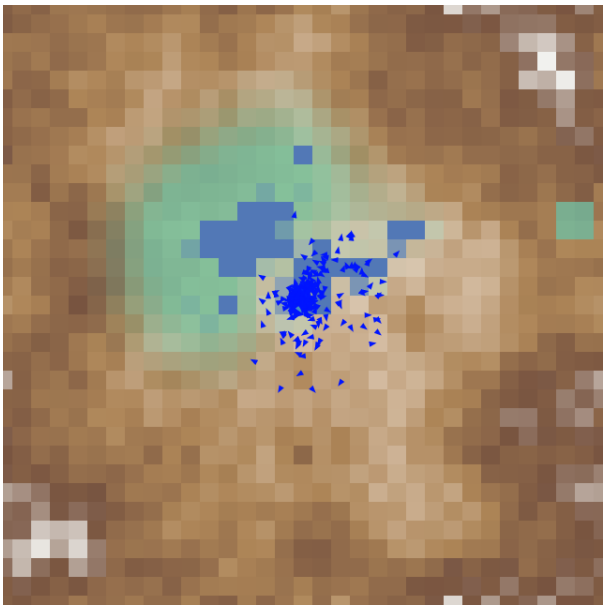

Figure 1: Example of the normal environment display. This view can convey altitude (sand to white), water level (blue), and vegetation (green). In this view temperature and animat density are invisible.

#### 3.1.1 Height

Height values range from 0.0 to 1.0. The subjective interpretation for the values, as used in fine tuning environment dynamics, is as follows:

- 0.0 to 0.15: Akin to Death Valley
- 0.15 to 0.5: Coastal to temperate
- 0.5 to 0.8: Temperate to upper foothills
- 0.8 to 1.0: Mountainous

When an environment is randomly initialized the terrain is generated randomly using an terrain generator based on the Diamond-Square algorithm [7].

Terrain height is a fully independent variable meaning it is not affected by any environmental or animat effects.

Saving an environment means saving the height values of the terrain grid. This is done to ensure the same terrain is used for non-random terrain permutations (`nd` and `ns`).

#### 3.1.2 Temperature

Temperature values range from 0.0 to 1.0. The subjective interpretation for the values, as used in fine tuning the environment dynamics, is as follows:

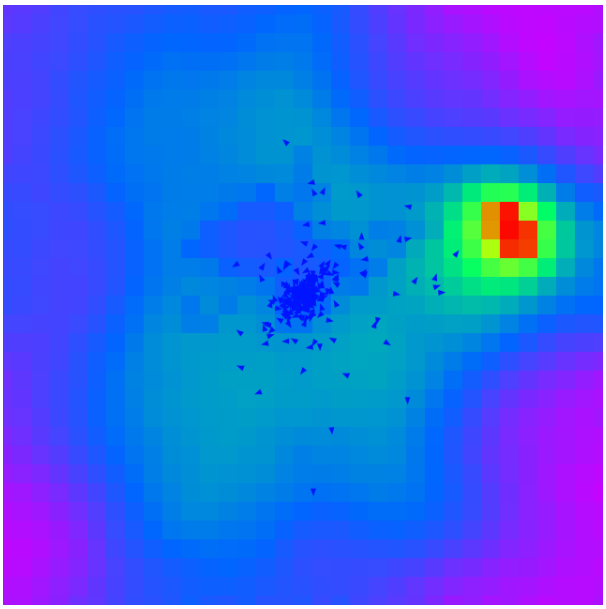

Figure 2: Example of the temperature environment display. This view displays temperature in a rainbow scale with low wavelength colors (e.g. red) denoting high temperatures. Note the dependence on height. The hot spot to the right is a randomly spawning fire.

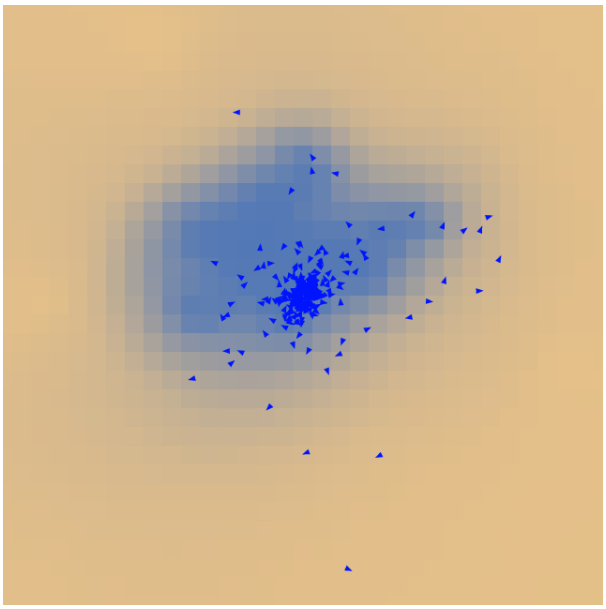

Figure 3: Example of the moisture environment display. This view displays moisture in an sand to blue scale. Fully submerged tiles (under the water level) show as completely blue.

- 0.0 to 0.2: Freezing to very cold, very high altitude
- 0.2 to 0.4: Cold to mild, high altitudes
- 0.4 to 0.6: Mild to warm, low altitudes
- 0.6 to 0.8: Warm to very hot, very low altitudes
- 0.8 to 1.0: Very hot to dangerous, low altitudes in heat waves, fires

Using the subjective interpretations a scale to determine altitude-dependent temperature was developed. Dynamically the temperate will tend towards that altitude-dependent values. At the same time the temperature diffuses outward using a simplistic (and therefore non-realistic) diffusion model.

Water will also act as a temperature regulator. Temperature over submerged tiles will tend towards a relatively low temperature (0.25) at fast rate.

Temperature on all tiles is uniformly offset by a global temperature value. This is the primary mechanism for generating heat waves and cold snaps.

A visualization of temperature in an environment can be seen in figure 2.

#### 3.1.3 Moisture

Moisture is a key factor in the environment as it dictates the survivability of vegetation. Like the other layers, moisture also varies from 0.0 to 1.0. There are no subjective measures that the moisture levels map to.

The important dynamic with moisture is that tiles that are submerged (i.e. height less than global water level) have full moisture (value of 1.0). The only other dynamic is that moisture diffuses, much like temperature. However, diffusion of moisture is affected by the gradient of the terrain making it more difficult for moisture to diffuse uphill.

Moisture also evaporates with respect to temperature and altitude. However, moisture does not affect temperature.

A visualization of moisture in an environment can be seen in figure 3.

#### 3.1.4 Vegetation

Vegetation is the most important environmental factor as it is, currently, the only energy source available to the animats. Vegetation also ranges from 0.0 to 1.0 without subjective measures.

The growth of vegetation is dependent on height, moisture, temperature, and vegetation itself. There are comfort bands of height and temperature for good vegetation growth. Vegetation grows from other vegetation. The more vegetation there is in a tile's neighbors the faster vegetation will grow in that tile. Moisture is also required by vegetation to grow. Without moisture vege-

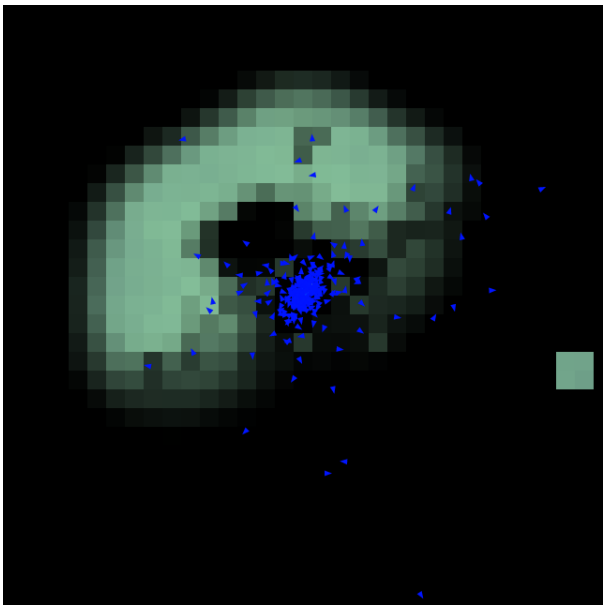

Figure 4: Example of the vegetation environment display. This view displays vegetation on a black to green scale. The vegetation cluster on the right is a randomly spawning "oasis."

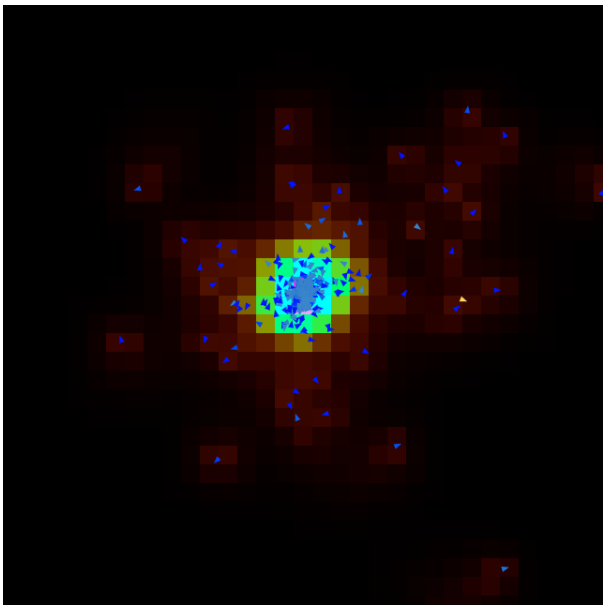

Figure 5: Example of the animat density environment display. This view displays animat density on a complex red to green scale on top of black. Note that the density diffuses and lingers.

tation will die. An interesting dynamic between vegetation and moisture is that vegetation will consume moisture out of the environment. During high environmental stress this can cause the vegetation to stripe into alternating tile patterns.

A visualization of vegetation in an environment can be seen in figure 4.

#### 3.1.5 Animat Density

Animat density is the most unique environmental factor. It is independent of other environmental factors. The value range is not limited except that it is always non-negative. It also has no subjective interpretations as the interpretation is direct and straightforward.

Although representing animat density this layer more represents pheromones left behind my animats. Each simulation tick an animat will add value to the animat density layer at its location. The animat density values degrade over time and also diffuse. This allows animats to leave behind faint trails. The scents will pile up providing a measure of how many animats are in the area.

A visualization of animat density in an environment can be seen in figure 5.

### 3.2 Design

The design of the environment dynamics and interactions was a fine tuned affair. Although against the principle of indirection, in order to ensure that the environment dynamics are interesting and stable at the same time a game of fine tuned parameters was played.

For example, under early iterations of the environment design vegetation growth was too slow to maintain itself and would die out. This was fixed with modification of the parameters and forced trace vegetation around water sources so that there is always a source for vegetation growth.

### 3.3 Global Values

The environment also consists of the following global variables: water level, global temperature, and vegetation growth factor.

Water level is used to determine which cells are considered submerged and which are not. Moisture diffusion, vegetation growth, and temperature regulation are secondary effects determined primarily from whether or not a cell is submerged. The water level is also used to figure out if an animat is swimming or not.

Global temperature is a uniform temperature offset in every cell. By default it rests at zero allowing the temperature to go to its natural altitude dependent value.

Vegetation growth factor is a uniform offset for vegetation in every cell. By default it also rests at zero.

### 3.4 Calamities

The dynamism of the environment is implemented with periodic calamities of increasing severity. The calamities are implemented by modulating the global values as follows:

- Flood: increase water level, slightly decrease temperature
- Drought: decrease water level, slightly increase temperature
- Heat wave: increase temperature, slightly decrease water level
- Cold snap: decrease temperature
- Famine: decrease vegetation growth factor

The modulation values are tuned to cause subjectively equal damage given the same severity factor. The end of each calamity increments the global severity factor. Increasing the severity ensures that the simulation will terminate at some point by killing off all remaining animats. Even the smartest animat cannot survive a biblical flood.

Static environments simply do not utilize this calamity mechanism. Instead the environment is allowed to simply proceed towards steady state.

## 4 Animats

Animats represent generic prey-like animals in the environment. Despite being prey-like there are no predator animats. An animats goal is to simply survive as long as it can in the environment.

### 4.1 Attributes

The animats are quite simple and consist of only eight attributes.

- Ticks: length of lifespan in simulation ticks
- Direction: stored as an angle
- Position: stored as continuous x and y coordinates
- Position history: moving average of the animats position
- Energy: life of the animat
- Stomach: amount of contents in the animat's stomach
- Vulnerability: probabilistic vulnerability to sudden predation
- Brain: processes and dictates animat behavior

### 4.2 Energy Dynamics

The primary currency for an animat is energy. It is a value that ranges from 0.0 to 100.0. Energy allows existence and, upon depletion, causes death and an early exit from the gene pool. An animat can gain energy only through digestion of vegetation. However, there are numerous ways for an animat to lose energy including: homeostasis, temperature regulation, motion, swimming, eating, starvation, and predation.

Simply living causes an animat to lose energy. This dynamic is meant to capture simple homeostasis. An extension of this is temperature regulation. There is a preferred temperature band between 0.35 and 0.51. When an animat's local temperature is outside this band it expends additional energy quadratically proportional to the amount of temperature outside the comfort band. This is intended to capture temperature regulation, a part of homeostasis.

Moving in the environment expends energy. Turning expends much less energy than movement. The amount of energy expended is exponential (to the power 1.5) to the amount of movement or turning. Additionally, an animat is considered swimming if its local height is lower than the global water level. When an animat is swimming its movement speed is reduced and energy expenditure increased.

The act of eating also consumes energy. This is because gaining energy from vegetation is a two-step process. The animat must first expend energy to eat the vegetation. Eating fills the animats stomach. When the stomach is not empty the animat digests the contents of the stomach at a constant rate for a constant energy gain. Although the immediate energy expenditure is higher than the immediate energy gain over time the energy gained from complete digestion far outweighs the initial cost. When the stomach is near empty (less than 1% full) the animat sustains an additional "starvation" cost.

Predation is abstracted since predatory animats have not yet been introduced. Instead predation is modeled as a stochastic process. Animats maintain an internal (i.e. not sensed) vulnerability level. This vulnerability level represents a probability per tick of becoming attacked. An attack will reduce an animat's energy by 80.0 points but will reduce the vulnerability by half.

Vulnerability can be reduced by either moving around or clustering with other animats. Specifically, vulnerability is reduced by maintaining a healthy distance from the animat's position history. This is meant to encourage actual movement and discourage faked movement involving simply moving in a very tight circle. The animat's position history is a moving average of the animat's position. Being in a location with high animat density will also lower vulnerability. This provides selectional

pressure for grouping and herding while providing loners with the ability to maintain a low vulnerability as well.

## 4.3 Brain

The brain of an animat is implemented using a neural network. The neural network contains a number of input neurons (sensors), output neurons (motors), hidden excitatory neurons, and hidden inhibitory neurons. The output, hidden excitatory, and hidden inhibitory neurons are fully connected (i.e. connected with every other neuron). Input neurons are not connected with other neurons.

Input neurons have their input set by the animat's sensors. The inputs to these neurons are generally fairly normalized values.

The default brain contains 36 input, 7 output, 14 hidden excitatory, and 15 inhibitory neurons. The set number of neurons and fully connected nature means every animat shares the same brain structure. This fact plays an important role in determine the animat genome (see section 5). The original project planned for unfixed neural network topology, but the idea was cut due to time constraints.

Neuron processing involves summation of inputs weighed by a connection strength, scaling it with a factor, squashing it with a sigmoid function, and then checking the result against a threshold. If the threshold is exceeded the neuron will fire. A neuronal stochasticity acts as a linear interpolating parameter between 1.0 and a random normalized number to determine the output value of a firing neuron.

Simulation of the brain processing occurs in lock step with the environment simulation. That is, for every one simulation tick of the environment there is one simulation tick of a brain and all its neurons.

### 4.3.1 Sensors

Animats brains contain a large array of input neurons. These neurons can be grouped into the following sensors: height, temperature, moisture, vegetation, animat density, swimming, energy, stomach, avoidance.

The first five are sensors for an environmental layer. Each of these groups contains five input neurons. One neuron detects the value of the layer at the animats location (e.g. the animat's height). The other four detect the local gradient (e.g. slope) of that layer at the animat's location in four animat-relative directions: front, back, left, right.

Energy is sensed with four input neurons. One neuron's input simply scales with the animat's energy. Two other neurons fire when the energy is considered low (less than 25.0) or high (greater than 75.0). One more neuron is used to sense the change in energy.

One neuron fires depending on whether or not the animat is swimming. Another fires depending on the fullness of the animat's stomach. The last fires to signal how much the animat is succeeding in avoiding predators.

### 4.3.2 Motors

The large number of input neurons is contrasted by a small set of output neurons organized into three motor groups: turning, movement, and eating.

The turning group contains four neurons: lean left, turn left, lean right, and turn right. The total turn amount is the summation of all four neuron outputs with extra weight given to the turn outputs (four times as much as the lean outputs). This provides essentially 16 permutations for turning and, therefore, 16 gradations of turn amount.

Movement has a similar system except with only two neurons: walk forward and run forward. The two neurons provide four permutations and, therefore, four gradations of forward movement. These gradations can be interpreted as stand, walk, run, and sprint. Animats cannot move backward.

Finally a single output neuron dictates whether or not the animat is eating. If the animat is eating then it adds 0.1 units to the stomach (the stomach is considered full at 1.0) per unit of vegetation consumed. This consuming removes 0.5 units of vegetation from the terrain. Removal of vegetation from the terrain by consumption is an important effect as it introduces limited resources to the dynamics. A food source over saturated with animats may pressure other animats to search for other food sources.

## 4.4 Designing Interestingness

Much like designing the environmental dynamics, designing the animat energy dynamics required plenty of manual tweaking. This was done to ensure that animat dynamics were interesting and balanced. Early iterations saw animats dying too quickly, surviving too long, spinning in tight circles to take advantage of the avoidance system, and avoiding movement due to poor selection pressure. Although this designing is counter to the principle of indirection, the hypothesis is tested via comparison of populations operating under the same dynamics. Maintaining balanced dynamics helps the system avoid degenerate behaviors that are incomparable.

# 5 Evolution

Animat behavior is adapted to the environment using evolution. No other learning mechanism is used in the neural network or otherwise. Since the animats contain

no unique, dynamic altering attributes an animat's behavior and fitness is determined solely by its brain. The brain alone is encoded in the animats genome and it is on the brain that evolution operates.

## 5.1 Genotype

Since each animat shares brain structure in regards to number of neurons and number of connections the genome can be written as a sequential dump of each neurons parameters. An animat's brain is the genome.

Constructing a genome is simply matter of walking through all of the neurons in the brain in the same order and writing out each neuron's gene to the genome. Reconstructing the animat's brain from the genome is then the inverse of this process in the same order.

Each single neuron is encoded as a gene. The process of encoding and decoding a single neuron is much like with the entire brain. The attributes of a neuron are written and then the connections are iterated in the same order to write out the connection strengths. Reconstructing a neuron is a matter of doing the inverse process in the same order.

Neuron genes start with the neuron's threshold, domain scale, and stochasticity (see section 4.3) followed by all of their connection strengths. The attributes and connection strengths are all written as floats.

Due to the direct encoding of the animat's brain into the genotype the phenotypic expression of the genome is simply the brain itself. Unfortunately, the structure of the genome also prevents mixing genomes of two different brain structures.

## 5.2 Operators

Mating between two animats results in evolutionary operations on the two genomes to produce the child genome. These evolutionary operations include cross-over and mutation. Both operators act on the genome and gene level stochastically.

Each step along the genome increases the chance of a cross-over. When a cross-over is triggered, one of three operators is randomly chosen: use A, use B, or mix. The selected operator is then applied to successive genes until the cross-over chance is triggered again. The first two cross-over operators are self-explanatory. The mix operator causes the two genomes to mix in a process similar to genome mixing.

Each gene visited in the genome during genome mixing has a small chance of becoming mutated. When this chance is triggered the gene is completely randomized using the same bounds as when a brain is randomly initialized.

Gene mixing follows the same procedure as genome mixing except on an neuronal attribute level. Each step of gene mixing visits a neuronal attribute (threshold, domain scale, stochasticity, or connection strength). The same cross-over procedure applies except the details of the mix cross-over operator are different. The mix operator will mix the two attributes using stochastic linear interpolation (with an interpolation parameter between 0.3 and 0.7).

Also like genome mixing, each attribute visited in a gene has a small chance of becoming mutated. When this occurs the attribute is multiplied by a random value between 0.5 and 1.5.

# 6 Implementation

The software for the project was written from scratch in JavaScript. Choosing JavaScript as the development language was driven by its prevalence, use for manipulating web pages, ease of coding, and familiar syntax.

JavaScript is of course an extremely prevalent language due to its use for manipulating web pages. Such prevalence translates directly to increased documentation in the developer community. Even more beneficial are the numerous libraries already written such as D3. The syntax is very familiar to any programmer previously exposed to C or C++. Ease of coding comes from its dynamic typing and garbage collection.

Utilizing JavaScript also allowed impressive visualization capabilities using the web browser (specifically Google Chrome) and immediate server side scalability using Node.js [4].

## 6.1 Visualization

Visualization was done using scalable vector graphics (SVG) manipulation in the document object model (DOM) with the data-driven-documents library (D3) [3]. The ability to manipulate vector graphics using query-like syntax made developing visualizations very easy. Lots of boiler plate and utility was taken care of by the D3 library. Impressively, since it utilizes standard web technologies the visualization can run on any compliant browser, including mobile phones.

## 6.2 Node.js

Node.js is a JavaScript engine (specifically Google's V8) adapted to run as a server side process using an asynchronous event architecture. Because it is adapted directly from a JavaScript engine from the world of web browsers nearly all web browser JavaScript code is portable. With some manipulation the simulation

software was adapted to be run by Node.js. This provided immediate computational bonuses as simulation runs could be spawned on powerful remote machines. Without this evolving all of the populations to the generation counts in this experiment's data would not have been possible in the time allotted. It is also entirely scalable as independent simulation processes can be spawned across clusters.

## 6.3 Source Code

Source code for the prototype can be found at the following URL:

`https://github.com/vietjtnguyen/ucla-winter13-cs263c-static`

Unfortunately the prototype lives up to its name. The prototype has not been designed for robust execution or even functionality. Instead it is only sufficiently developed to test the hypotheses for this study. Although extensible, a clean API and suitable documentation will require additional future work.

# 7 Results

Data was gathered from the simulation as just the lifespans of the animats as they died. The fitness of an animat is the length of time it survives against the environment so having just the lifespans of each animat allows us to view the distribution of fitness in a population.

## 7.1 Native Populations

Native animat populations were successfully evolved in each of the environment permutations. The following is a list of the native populations evolved and the number of generations used to evolved them:

- `rd` (randomized and dynamic): 780
- `rs` (randomized and static): 760
- `nd` (non-randomized and dynamic): 760
- `ns` (non-randomized and static): 760
- `c` (control): 0

The difference with the number of generations for `rd` is due to the loss of the 760 generation save. In addition to the four environment permutation populations is the seed population from which all of the populations evolved. This population is denoted `c` with an effective generation of zero.

The improvement in animat behavior was impressive as shown in table 1. After evolving populations for 60 runs of 400 animats per population in their native environments a collection of 24,000 lifespans per population was generated. The distributions of these lifespans provides us with a measure of the performance of the population.

| Pop./Env. | Mean ($\mu$) | Median ($Md$) |
|---|---|---|
| `rd`/`rd` | 2121.93275 | 1242.5 |
| `c`/`rd` | 798.006 | 637.0 |
| `rs`/`rs` | 4108.5705 | 6000.0 |
| `c`/`rs` | 1000.61325 | 691.0 |
| `nd`/`nd` | 2556.911875 | 2873.0 |
| `c`/`nd` | 821.451875 | 675.0 |
| `ns`/`ns` | 2894.5315 | 881.0 |
| `c`/`ns` | 884.189125 | 689.0 |

Table 1: Table of means and medians for native populations in native environments and control population in same environment.

The two distributions can be compared quantitatively (if simplistically) with the mean and median. Here the notation $\mu_{\text{a,b}}$ represents the mean and $Md_{\text{a,b}}$ represents the median of the population a in environment b. Higher means and medians indicate overall higher fitness in a population.

Table 1 displays means and medians for all native populations in their native environments compared to the control population in the same environment. A noteworthy entry is the median of `rs` in `rs` of 6000.0, the simulation tick cap. The evolution of the `rs` population has apparently been particularly successful.

## 7.2 Experimental Results

For the actual experimental runs we found our hypothesis to not hold. Quantitative data can be seen in table 2. As expected, the `ns` population performs worse in the `rd` environment, most likely due to the lack of selection pressure for the `ns` population to remain adaptive. However, our expectation that the `rd` population would perform slightly worse than the `ns` population in its native environment proved wrong (first group in table 2). Instead the `rd` population performed extremely well in the `ns` environment, exceeding the performance of either `ns` or `rd` in its native environment (second and third rows of first group in table 2).

This result can also be seen in figure 6 and figure 7. These figures represent proportions of histogram bins belonging to either the foreign, experimental population (blue), the native population (red), or the control population (gray). Because the lifespan increases along the y-axis a well performing experimental population should have a larger proportion at the top of the graph and vice versa for a poor performing experimental population.

As we can see in figure 6 the `ns` population fails to

| Pop./Env. | Mean ($\mu$) | Median ($Md$) |
|---|---|---|
| rd/ns | **3154.635875** | **2042.0** |
| ns/ns | 2894.5315 | 881.0 |
| rd/rd | 2121.93275 | 1242.5 |
| c/ns | 884.189125 | 689.0 |
| ns/rd | **1500.51025** | **687.0** |
| rd/rd | 2121.93275 | 1242.5 |
| ns/ns | 2894.5315 | 881.0 |
| c/rd | 798.006 | 637.0 |
| rd/rs | **3488.885** | **4988.0** |
| rs/rs | 4108.5705 | 6000.0 |
| rd/rd | 2121.93275 | 1242.5 |
| c/rs | 1000.61325 | 691.0 |
| rs/rd | **2638.57125** | **3119.5** |
| rd/rd | 2121.93275 | 1242.5 |
| rs/rs | 4108.5705 | 6000.0 |
| c/rd | 798.006 | 637.0 |

Table 2: Table of means and medians for experimental runs. Each grouping shows the transplanted populations performance in a foreign environment (bold face row), the performance of a population native to that environment, the performance of the transplanted population in its native environment, and the performance of the control population in the foreign environment.

take up a large proportion of the high-lifespan bins representing its poor performance in the rd environment, as expected. However, figure 7 shows that the rd population indeed performed well, occupying the larger proportion of high-lifespan bins. It is noteworthy, however, that the ns population occupies the majority proportion of mid-lifespan bins indicating that the ns population is still quite effective in its native environment. From this we can conclude that our hypothesis, although not holding, is off by a matter of degrees rather than the fundamental concept.

Not only did our hypothesis not hold, our assumption that a population evolved in a dynamic environment would outperform one evolved in a static environment proved incorrect as well! The rs population actually performed better than the rd population in the rd environment (fourth group in table 2). Surprising even further was that the rs population also performed better than the rd population in the rs environment (third group in table 2). Figure 8 and figure 9 show this result in more detail.

## 8 Discussion

Despite the data not supporting the hypothesis, we believe it remains fundamentally sound. Instead, the data's

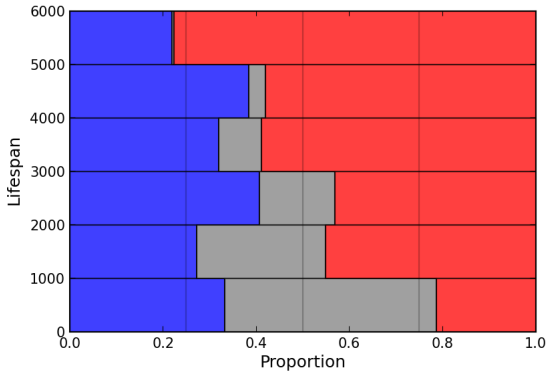


Figure 6: Lifespan histogram bin proportion for ns-in-rd experiment. Blue is the ns population (foreign), red is the rd population (native), and gray is the c population (control).

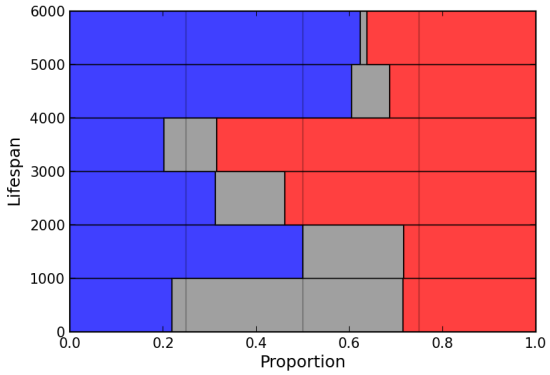


Figure 7: Lifespan histogram bin proportion for rd-in-ns experiment. Blue is the rd population (foreign), red is the ns population (native), and gray is the c population (control).

lack of support shows that there is a matter of degrees to specialized populations performing better than adaptive, robust populations in the specialized environment.

The specialized population, in this case ns, may have become stuck in a large local maximum. It would not be surprising if other, more effective behavior solutions can be found in the fitness space for that specialized environment (ns). These more effective behavior solutions may support the hypothesis.

Further experimentation may attempt to find these more maximal solutions using other techniques such as simulated annealing [5] or novelty search [6].

Finding that the rs population performed better than the rd population overall was very surprising. In fact, it

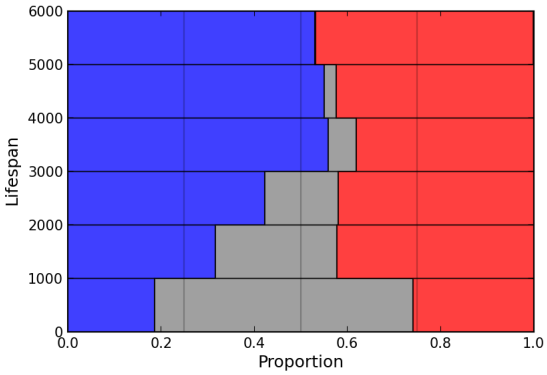


Figure 8: Lifespan histogram bin proportion for `rs-in-rd` experiment. Blue is the `rs` population (foreign), red is the `rd` population (native), and gray is the `c` population (control).

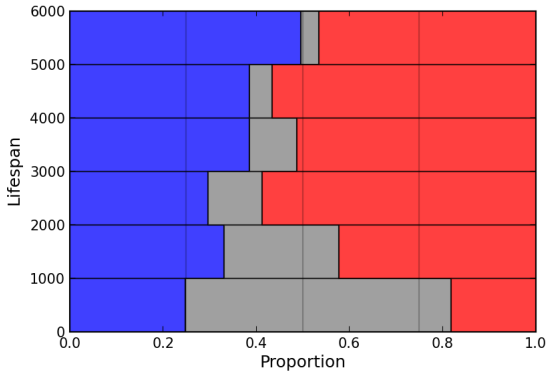


Figure 9: Lifespan histogram bin proportion for `rd-in-rs` experiment. Blue is the `rd` population (foreign), red is the `rs` population (native), and gray is the `c` population (control).

has brought up the question of possible data mishandling. To ensure that wasn't a factor, a new `rs` population may be evolved and the experiments rerun to confirm the result. If the result still holds then it is possible that these evolved behavior solutions are becoming trapped in local maxima.

## 8.1 Analysis

The analysis used in this report is relatively simplistic. Improved methods of distribution comparison could be utilized to get more quantitative results.

More data can also be collected. Experimental runs can be continually run in the background, accruing data. As these data sets become larger the random effects of random terrains on the distributions can be mitigated.

## 8.2 Interesting Observations

Although we expected subjectively smart behavior to arise, the animats managed to surprise. The behavior of the best population, evolved for 1500 generations in randomized and dynamic environments, exhibited rather keen behavior. This population had learned to swim back to shore when caught in the water. They also learned to swarm in circles around vegetation sources allowing them to minimize vulnerability by remaining near other animats and keeping consistent movement (which increases avoidance). Another population appeared to learn energy conservation by using the presence of nearby vegetation to climb uphill (when the extra cost is offset by a stable energy source) and sprinting downhill when the vegetation dissipated (taking advantage of the extra speed).

A peculiarity also shows up in regards to circling/swarming behavior. The majority, if not all, populations observed appeared to swarm in a counter-clockwise direction in all environments. Although a counter-case may have been missed during observation, the animats appear to prefer counter-clockwise motion. It is unclear why this is the case and how it arose. Further investigation may provide possible clues into the evolution of handedness preference.

## 8.3 Further Development

One glaring omission from the animat dynamics is thirst and the need for water for survival. This dynamic can be implemented with the system as is.

Selection pressure to explore for food and water sources is minimal in the current set up. Expanding the environment and adding truly limited resources could result in very interesting migratory behavior. For example, expanding the system to allow for local bodies of water that become consumed by animats could result is oases that become depleted. This would put selection pressure on the animats to explore, cooperate, and make decisions on whether they should stay or leave. The vegetation growth rates could be reduced and environmental events such as rainfall included.

The introduction of multiple animat types, including actual predators, can be done in the system. More complicate interactions between animats and different types of animats, such as a market, could possibly result in role specialization.

With the layer abstraction, adding complexity and in-

teraction to the environment can be relatively straightforward. Potential additions include different vegetation types (short, tall, poisonous, etc.) and wind effects on diffusion.

To alleviate the need for fine tuning environment and animat dynamics, the study of ecologies could be brought to bear on the problem. A good integration of these fields could possibly result in models sufficient for ecological study.

## 9 Conclusion

Beginning our study we expected dynamic environments to evolve more robust behaviors and for static environments to evolve more specialized behaviors. In our experiments we found these assumptions and hypotheses to be fragile. Our data showed populations evolved in randomized, dynamic environments performing better than specialized populations in their non-random, static environments. It also showed populations evolved in randomized, static environments performing better than populations evolved in randomized, dynamic environments in both static and dynamic environments. This finding was particularly surprising and warrants further investigation.

Perhaps most important is that a system has been developed robust enough to tackle these questions and be extended for further investigation. It is perhaps appropriate then that the initial studies utilizing the system have presented us with additional investigations for us to explore.

## 10 Notes

This report was written for the Computer Science 263C class taught by Professor Dyer during the Winter 2013 quarter at UCLA.

## References


[1] Bicubic interpolation. `http://en.wikipedia.org/wiki/Bicubic_interpolation`.

[2] Bilinear interpolation. `http://en.wikipedia.org/wiki/Bilinear_interpolation`.

[3] BOSTOCK, M. Data driven documents. `https://github.com/mbostock/d3`, September 2010.

[4] DAHL, R. Node.js. `http://nodejs.org/`, 2009.

[5] L., D. Genetic algorithms and simulated annealing.

[6] RISI, S., VANDERBLEEK, S. D., HUGHES, C. E., AND STANLEY, K. O. How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (2009), ACM, pp. 153–160.

[7] XU, X. Random fractal terrain generator. `https://github.com/qiao/fractal-terrain-generator`, October 2011.