Ecosystem Simulation Project Proposal

Layman, Brett

Montana State University

Introduction

The intention of the proposed project, tentatively called AL Agents, is to create a general and flexible framework for modeling a wide variety of artificial life (AL) systems. Unlike most models in the field of artificial life, this system is intended to be capable of modeling a wide variety of different systems, with highly customizable agents and environments.

The flexibility of the proposed system is due in part to its use of neural networks to guide agent behavior. Some problems in AL require logical agents, while others require relatively simple linear agents, and others yet require agents capable of complex non-linear decision making. Multi-layer feed-forward neural networks are capable of modeling all of these agent types, although logical agents can be a challenge to implement. Another useful feature of neural networks is their capacity to produce probabilistic or stepwise outputs, depending on the activation function used. This means that the behavior of the creatures can be either deterministic or stochastic, depending on the type of system being modeled. The user will be able to specify the activation function, as well as the weights of an agent's neural networks: which could be randomized or hard coded.

Another characteristic of AL Agents that broadens its scope, is it's use of generic, user defined resources. The user can create however many resources they would like the system to have, and to specify which types of agents require which resources. In addition to collecting resources from the environment, agents can deposit resources, or even take them from other creatures. This allows for the creation of systems with intricate food webs and inter-dependencies.

AL Agents uses a genetic algorithm (GA) to guide agent behavior. GAs are useful for many questions in the field of AL, because they share certain characteristics with natural evolution, and they can be used generally as a tool for learning. GAs have already been used to study a variety of subfields in AL including: immune system models, ecological models, population genetics, social systems, and interactions between evolution and learning (Mitchell and Forrest, 1995). AL Agents is flexible enough to be applied to many of these subfields.

The proposed project has the potential to be a convenient and useful tool for researchers in the field of AL, as well as an entertaining and educational program for students and others who are curious about what can be created with the AL Agents framework. In order to make the program more appealing, a user-friendly point and click interface will be developed for customizing the environment and agents, as well as for running and visualizing the simulation. Optional presets will be available for simplifying and expediting the creation process.

Qualifications

Please see attached resume (back pages).

Background

AL Agents isn't the first Agent-Based GA with an endogenous fitness function to model ecosystem dynamics. Holland's (1993) Echo model is perhaps the most similar to AL Agents, in that it was designed to be general purpose, and focused on how resources flow between agents. The Echo model allows agents to trade and fight. This allows for both competitive and cooperative modes of resource exchange, and in fact their model demonstrated both competitive arms races, and cooperative symbiotic relationships. These are phenomena that I would also like to produce in AL Agents. Despite commonalities in scope and intent, Echo does have some key differences with AL Agents. For instance, the system for guiding creature behavior is relatively simple in the Echo model. A creature's behavior is determined by a table of string to action rules, where the string is the phenotype of the other creature, and the action could be trading, attacking, or reproducing. This model results in simple reflex agents that are guided by the encoded appearance of other creatures. In contrast, the agents in AL Agents are capable of incorporating other inputs into their decision making process, such as: environmental inputs, internal state inputs, memory inputs, and communication inputs. Of course, this added complexity requires more from the user in terms of creating an effective neural network structure for guiding behavior, but that challenge is a concern for any neural network based learning task, and I don't see it as an unsurmountable obstacle.

Unlike Holland's Echo model, Ackley and Littman's (1992) Evolutionary Reinforcement Learning (ERL) model did make use of neural networks to guide agent behavior. However, the model had limitations in other regards that prevent it from being used for a wide variety of experiments. In the ERL model, agent behavior consists solely of movement. Agents automatically eat or engage with predators if they land on a spot with food or a predator. The only resource in the environment is energy (food). The model involves non-agent predators that are guided by hard coded finite state automata, and plants that provide produce food. These specific aspects of the ERL model make it difficult to generalize to situations that require multiple resources, or flexible, agent-based predators. One of the main purposes of the ERL model was to study the interaction between reinforcement learning during an agent's lifetime, and GA scale learning at the population level. They found that combining reinforcement learning with GA learning produced the most successful populations, which they attributed to the Baldwin effect. The AL Agents model won't incorporate reinforcement learning, and so it has that limitation. Time permitting, I would like to incorporate classical conditioning into the AL Agents model as a simple form of learning. This could be accomplished by following the principal often cited in neuroscience that, "neurons that fire together wire together". One could create links between memory nodes and current input nodes, where the weight of the link is increased whenever the two nodes fire in unison. Then, when only the memory node is activated, it will partially activate the input node, even though the input isn't actually present at this time. This could potentially be maladaptive in cases where it's important for the creature to distinguish between the two associated inputs, but it could also be adaptive when one of the inputs is a predictor of the other, and the creature should act when the predictor is present. In effect, the activation of the memory node is creating an expectation. This could be an interesting direction for future research.

Bedau and Packard's (1992) "Strategic Bugs" model uses relatively simple agents guided by a lookup table of rules. The agents move throughout the system looking for food, which determines survival and reproduction. Despite the simplicity of this model, Bedau and Packard use it to create a compelling measurement for evolutionary activity. They measure evolutionary progress by looking at how often particular rules are used in a creature's lookup table. If a creature starts to use a rule on a frequent basis, they say that this is a sign that the creature has made an evolutionary innovation. AL Agents could perform a similar measurement by looking at how often a particular sequence of nodes activates a particular output. Alternatively, one could perform time series analysis on edge weights in order to distinguish random variation in a population from meaningful evolutionary trends.

Despite AL Agents similarities with other frameworks, It is far from identical to any previous project, and I believe that its unique set of characteristics offer something distinct and beneficial to the field of AL.

Work Schedule

Lifecycle Approach

I plan to use an iterative approach for the development of AL Agents. Early iterations will be more focused on conceptualization, design, and prototyping, while later iterations will focus more on the implementation of specific features and refining the user interface. I have started on this process by working on coarse-grained representation of the overall software architecture of the program. This will be followed by incrementally implementing components of the overall architecture and testing them with mock user inputs. After implementing several parts of the architecture, I plan to start testing a simple user interface to replace the mock inputs.

Progress Schedule

<u>November 2018</u>

- Develop front end prototype for environment creation.
- Test environment, creature and network creation.
- Work on designing software architecture for agent reproduction.

<u>December 2018</u>

- Test running the system for multiple steps with creature movement.
- Represent the creatures and resources visually on a grid.

January 2019

- Transition the simulation to a separate thread.
- Experiment with setting neural network weights to guide agent behavior.
- Work on UI for creature and network creation.

February 2019

- Work on teaching agents to move along resource gradients.
- Work on UI for running simulation.

March 2019

- Work on teaching agents to trade resources.
- Work on improving visualizations for creatures and resources.
- Work on allowing user to save populations of creatures.

April 2019

- Work on teaching agents to use their memory to guide behavior.
- Work on writing population statistics to a csv file.

Proposal Statement

Functional and Non-Functional Requirements

Functional Requirements

- User can design an environment by distributing resources across a 2D grid.
- User can design an agent, including the agent's attributes and neural networks.
- User can modify inputs, outputs, and link weights of a creature's networks.
- User can run the simulation at different intervals or for a set number of steps.
- User can pause the simulation and modify model before resuming it.

- User can save populations of creatures to use later.
- The program writes statistics to a csv file.

Non-functional Requirements

- User interface is easy to navigate, and not overwhelming.
- It is easy to visualize the resource levels in the environment, and the distribution of creatures in the environment.
- The software architecture is extensible and can be easily modified to support additional requirements in terms of creature behavior.
- The architecture supports different levels of granularity in terms of designing the agents and environment.

Performance Requirements

Further testing is required to determine a realistic goal in terms of performance requirements. A preliminary estimate for the time complexity of the program is: O(T*N*I*J*K^2) where T is time, N is the number of agents, I is the number of networks per agents (usually around 2 – 8), J is the average number of layers in a network (usually 2-4), and K^2 is the average number of nodes per layer squared (could be large if there are a large number of hidden nodes). Let's explore this formula with an example. Let's say that the user would like to simulate 100 creatures for 1000 time steps, where I = 4, J = 3, and K^2 = 100. According to our time complexity, this simulation will require at least: 100 * 1000 * 4 * 3 * 100 = 120,000,000 steps. This is a significant number of steps for a relatively small simulation and could take on the order of hours to finish. However, there are several ways in which the performance could be improved.

One potential optimization would be to partition the agents into separate clusters that can't interact (based on their location) for a particular number of steps, and to run the simulation on each cluster using a separate thread. The partitioning algorithm would likely take O(N^2) time, and be run every x steps, where the agents in a partition are at least 2*x distance from agents in another partition. Another optimization would be to prune the neural networks according to which weights appear to have a functional significance. This could be especially important in situations where there are many hidden nodes. This approach would clearly affect the results of the simulation, but not necessarily in an unrealistic way. Neural networks in the human brain are pruned significantly in the developmental process, and useless but slightly detrimental traits are often gradually lost in the process of evolution.

Interface Requirements

- There will be a point and click user interface with buttons and text boxes.
- The interface will have different windows for environment creation, resource creation, creature creation, and neural network creation.

Architectural Design Documents

I will be producing html design documents. The documentation is currently incomplete.

Tools Used

I will be using Unity to display the user interface, and to visualize the state of the system. I will be running the simulation on a C# thread that is attached to a Unity C# MonoBehavior script.

Methodology

Software Architecture

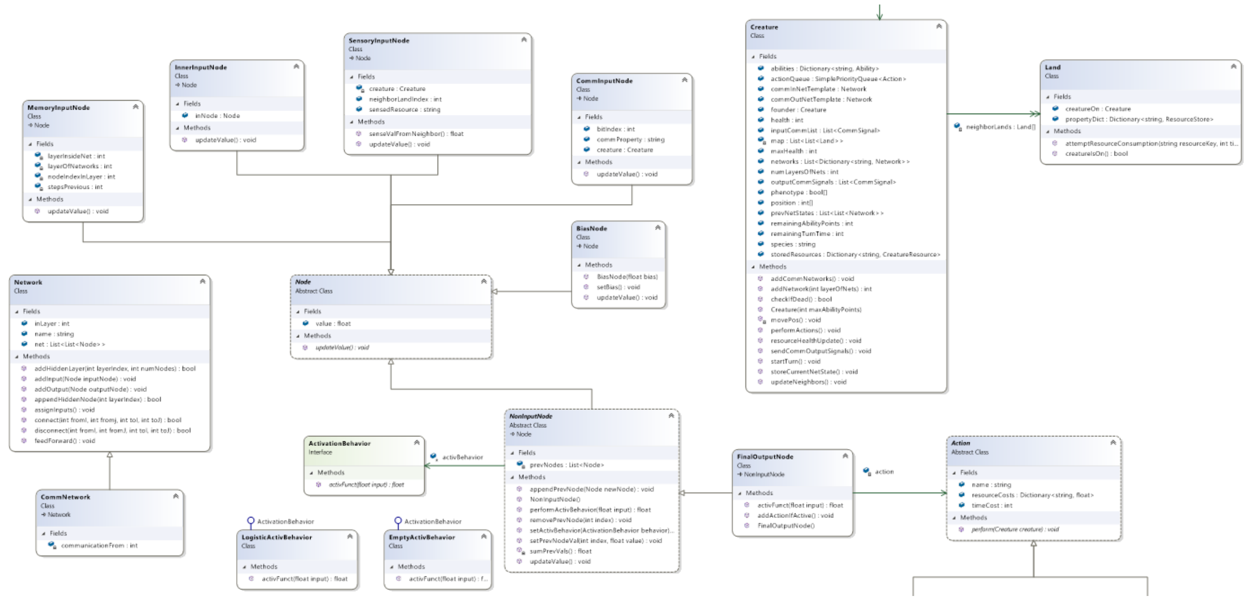

Figure 1. This partial class diagram shows several key classes including: Land for modeling the state of each grid space, Creature for modeling agents, Network for modeling a neural network in an agent, Node for modeling nodes in a network, and Action for modeling actions that a creature can take. Another important class, Ecosystem, can be seen in Figure 3.

Design Patterns

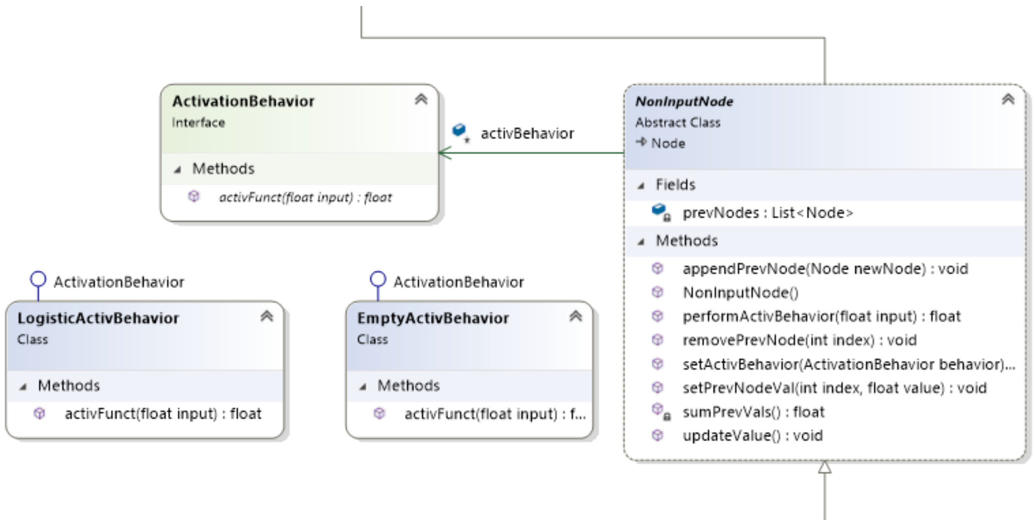
ActivationBehavior
Interface
Methods
activFunct(float input) : float
activBehavior
NonInputNode
Abstract Class
Node
Fields
prevNodes : List<Node>
Methods
appendPrevNode(Node newNode) : void
NonInputNode()
performActivBehavior(float input) : float
removePrevNode(int index) : void
setActivBehavior(ActivationBehavior behavior)...
setPrevNodeVal(int index, float value) : void
sumPrevVals() : float
updateValue() : void
ActivationBehavior
LogisticActivBehavior
Class
Methods
activFunct(float input) : float
ActivationBehavior
EmptyActivBehavior
Class
Methods
activFunct(float input) : f...

Figure 2. I will be using the strategy pattern to allow nodes to switch their activation function at run time.

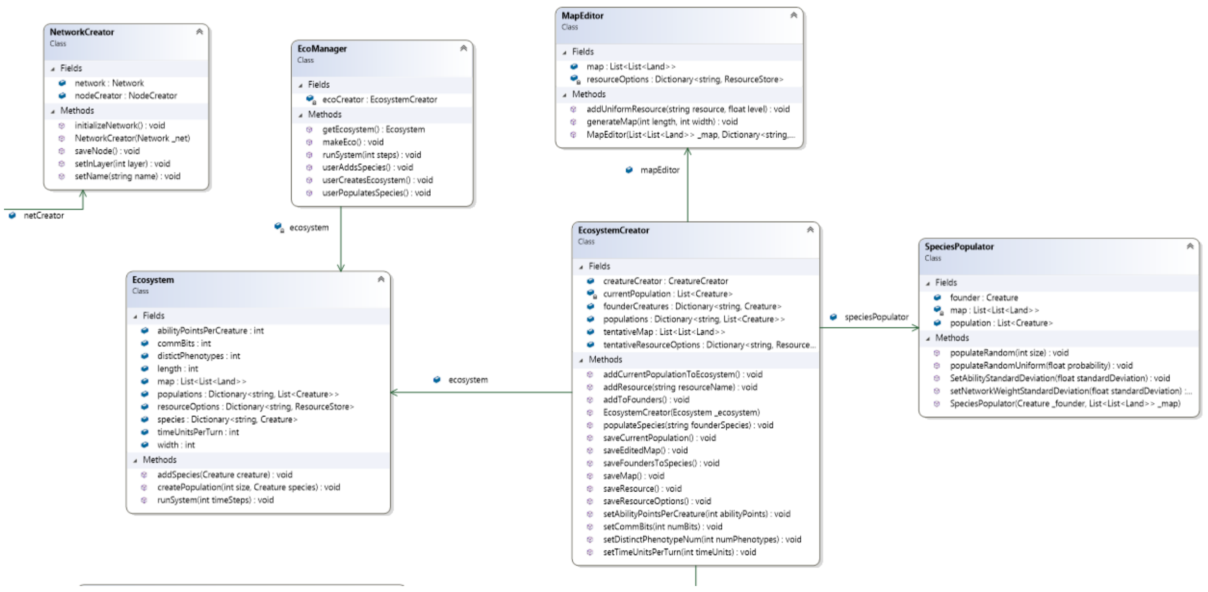
NetworkCreator
Class
Fields
network : Network
nodeCreator : NodeCreator
Methods
initializeNetwork() : void
NetworkCreator(Network _net)
saveNode() : void
setInLayer(int layer) : void
setName(string name) : void
netCreator
EcoManager
Class
Fields
ecoCreator : EcosystemCreator
Methods
getEcosystem() : Ecosystem
makeEco() : void
runSystem(int steps) : void
userAddsSpecies() : void
userCreatesEcosystem() : void
userPopulatesSpecies() : void
ecosystem
Ecosystem
Class
Fields
abilityPointsPerCreature : int
commBits : int
distictPhenotypes : int
length : int
map : List<List<Land>>
populations : Dictionary<string, List<Creature>>
resourceOptions : Dictionary<string, ResourceStore>
species : Dictionary<string, Creature>
timeUnitsPerTurn : int
width : int
Methods
addSpecies(Creature creature) : void
createPopulation(int size, Creature species) : void
runSystem(int timeSteps) : void
ecosystem
MapEditor
Class
Fields
map : List<List<Land>>
resourceOptions : Dictionary<string, ResourceStore>
Methods
addUniformResource(string resource, float level) : void
generateMap(int length, int width) : void
MapEditor(List<List<Land>> _map, Dictionary<string,...
mapEditor
EcosystemCreator
Class
Fields
creatureCreator : CreatureCreator
currentPopulation : List<Creature>
founderCreatures : Dictionary<string, Creature>
populations : Dictionary<string, List<Creature>>
tentativeMap : List<List<Land>>
tentativeResourceOptions : Dictionary<string, Resource...
Methods
addCurrentPopulationToEcosystem() : void
addResource(string resourceName) : void
addToFounders() : void
EcosystemCreator(Ecosystem _ecosystem)
populateSpecies(string founderSpecies) : void
saveCurrentPopulation() : void
saveEditedMap() : void
saveFoundersToSpecies() : void
saveMap() : void
saveResource() : void
saveResourceOptions() : void
setAbilityPointsPerCreature(int abilityPoints) : void
setCommBits(int numBits) : void
setDistinctPhenotypeNum(int numPhenotypes) : void
setTimeUnitsPerTurn(int timeUnits) : void
speciesPopulator
SpeciesPopulator
Class
Fields
founder : Creature
map : List<List<Land>>
population : List<Creature>
Methods
populateRandom(int size) : void
populateRandomUniform(float probability) : void
SetAbilityStandardDeviation(float standardDeviation) : void
setNetworkWeightStandardDeviation(float standardDeviation) :...
SpeciesPopulator(Creature _founder, List<List<Land>> _map)

Figure 3. I will use the facade pattern to decouple the user interface from the back-end data structures. In the above example, EcosystemCreator, MapEditor, and SpeciesPopulator are all facades that modify the Ecosystem object. Each façade class typically corresponds to a separate user interface window.

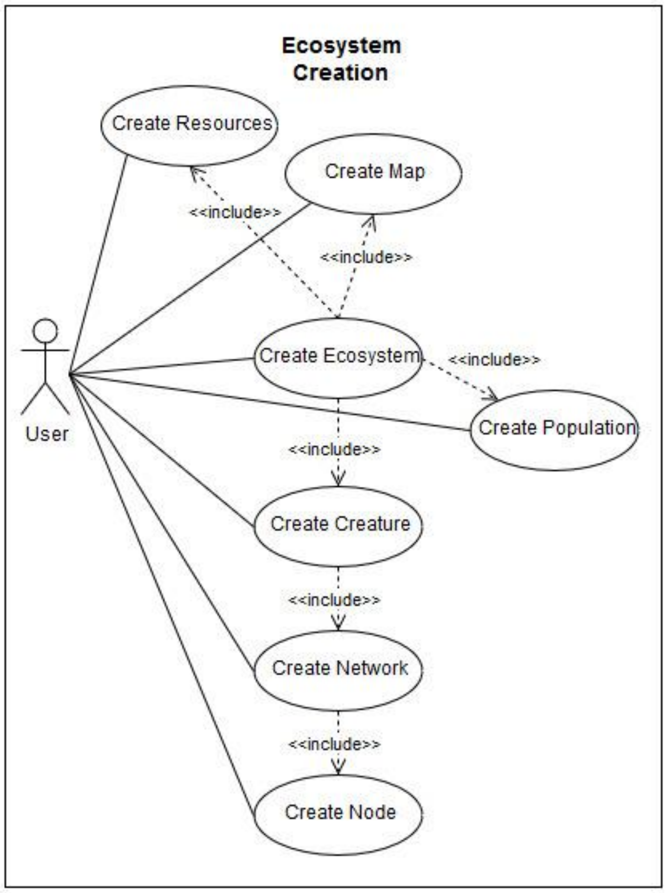

Figure 4. A use case diagram showing the different aspects of ecosystem creation. Note that creature (agent) creation is the only part of creating an ecosystem that involves further nested sub-tasks.

Trade-offs

One of the more difficult design decisions in building the software architecture for AL Agents was deciding how to organize an interface between the user and the underlying data. I knew that a façade would be necessary, but I wasn't sure how many classes would be involved, and what the scope of each class would be. On the one hand, having fewer classes would keep the façade more compact, but it could also make it more convoluted. Eventually I decided to make a separate facade object for every distinct window that the user interacts with. For example, the user interacts with a number of separate windows in the process of creating an agent: one for setting agent attributes, another for network creation, and another for node creation. Having a separate façade for each window makes it easy to save user entered data in organized bundles, based on what was entered in that window. There is usually a one to one mapping of interface windows to objects being created, so the façade acts as a wrapper for the object being modified by that window.

Agents play a central role in this system, and so a number of important decisions had to be made in regards to agent design and functionality. Initially, I wasn't sure if agent behavior should be guided by a look-up table or by a neural network. Look up tables are relatively easy to work with, and are generally much faster than neural networks. Additionally, the mapping of inputs to actions is straightforward, and easier to analyze in the context of a genetic algorithm. On the other hand, neural networks are more flexible than look-up tables. Look-up tables can become unmanageable when the number of potential inputs becomes large, because the number of possible combinations of inputs grows exponentially. This is not the case for neural networks, because mappings for each combination of inputs to outputs are encoded into the weights of the network. Additionally, a neural network can take in numerical inputs, rather than simply looking at whether a series of conditions are met. This means that the output of the network could be similar when similar sets of conditions are met. In theory, this characteristic of neural networks could making learning easier by smoothing out the fitness landscape. The performance downside of neural networks can be mediated by only making a network as complex as it needs to be for a given problem. For instance, some problems may only require a linear mapping of inputs to outputs, in which case only one layer is needed in the network. In the end, I decided that the benefits of neural networks, especially in terms of their flexibility, outweighed their downsides, and made them a good fit for this system.

Expected Results

I expect that AL Agents will demonstrate improvements in agent behavior over time as a result of the genetic algorithm. I expect the agents to become better at following resource gradients and picking up resources when their resource levels are low. I also hope to find that agents will learn to cooperate by communicating and exchanging resources. I expect that the agents will be able to learn multiple skills if trained in one skill after another. For example, they could learn to follow one resource gradient, and then another one, or potentially both simultaneously. I also expect that AL Agents will be able to statistically identify which weights in a creature's neural network are likely relevant to the creature's survival and reproduction, based on how those weights change over time in a population.

References

D. H. Ackley and M. L. Littman. Interactions between learning and evolution. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artifcial Life II*, pages 487{507, Reading, MA, 1992. Addison-Wesley.

M. A. Bedau and N. H. Packard. Measurement of evolutionary activity, teleology, and life. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artifcial Life II*, pages 431{461, Reading, MA, 1992. Addison-Wesley.

J. H. Holland. Echoing emergence: Objectives, rough defnitions, and speculations for Echo-class models. Technical Report 93-04-023, Santa Fe Institute, 1993. To appear in Integrative Themes, G. Cowan, D. Pines and D. Melzner, Reading, MA: Addison-Wesley.

M. Mitchell and S. Forrest, "Genetic algorithms and artificial life," *Artificial Life,* vol. 1, no. 3, pp. 267–289, 1995.

Brett Layman

BrettLayman7@gmail.com
406-570-3916
Bozeman, MT

Interests:

Artificial intelligence, biologically inspired algorithms, artificial life, simulating natural systems.

Skills

Programming Languages: Java, Python, C#, C++, C, JavaScript
Other Tools: Git, Unity, Visual Studio

Education

Montana State University
2016-2019
BS in Computer Science
GPA: 3.89

Western Washington University
2006-2011
BA in Behavioral Neuroscience
GPA: 3.31

Projects

2D Physics Game
Creator/Developer
Dec 2017 – current
Tools used: Unity, C#
Details:

- A multiplayer side-scroller game with race, jump, and battle game modes.
- 2D physics have a large impact on the gameplay.

Fractal Visualization App
Creator/Developer
May 2018 - June 2019
Tools used: Unity, C#
Details:

- App for visualizing fractal tree patterns by specifying various parameters including angle, depth, and random variation.
- Published as “Fractal Generator” on the Google Play Store.

<u>Procedural Texture Generation Program</u>

Creator/Developer
April 2018 – May 2018
Tools used: C++
Details:

- Created a ray tracing program to render 3D objects with Phong shading
- Textures generated using linear interpolation and noise.

<u>Evolutionary Algorithm Project</u>

Team member
Oct 2017 - Nov 2017
Tools used: Python
Details:

- Compared 3 different Evolutionary Algorithms with Backpropagation on a neural net classification problem.
- Performed a statistical analysis of the results.

<u>Neural Network Project</u>

Team member
Sept 2017 - Oct 2017
Tools used: Python
Details:

- Compared a radial basis function neural network to a feed forward neural network that used backtracking and gradient descent.
- Performed a statistical analysis of the results and wrote a report.

Work Experience

<u>TA for Introduction to C Programming</u>

May 2017 - June 2017
May 2018 - June 2018
Tools used: Linux, C
Tasks:

Helped students during lab time. Graded Labs.

<u>Web App Developer</u>

May 2017 – August 2017
Tools Used: JavaScript, html, css, Angular, socket.io, Node, Express, Postgres
Tasks:

Build a web app that allows students to share their code with the teacher and the rest of the class. Built using Express, Angular, and SocketIO. Saved data to a Postgres database.