Ecosystem Simulation Project Report

Layman, Brett

Montana State University

Introduction

The intention of Eco-Simulator is to create an open-ended, customizable framework for modeling a wide variety of artificial life (AL) systems. The field of AL attempts to understand life by creating artificial systems that exhibit life's properties through bottom-up mechanisms (Bedau, 2003). This is similar to how AI attempts to re-create and explore the properties of intelligent behavior through artificial systems. As an AL simulation, Eco-Simulator aims to model evolutionary and ecological processes through the cumulative behavior of neural network-based agents. Simulating these processes could help us to understand how life is capable of producing remarkably stable systems which support the coexistence of organisms from a variety of species for prolonged periods of time. The better we understand the emergent stability of these systems, the easier it will be to foster stability in the systems that we depend on.

The flexibility of the proposed system is due in part to its use of neural networks to guide agent behavior. Multi-layer feed-forward neural networks have the advantage of being able to create linear, non-linear, and logical mappings of inputs to outputs. Different problems may require different types of mappings, and it's helpful to be able to switch from one mapping to another when the need arises. For example, one can simply add activation functions to an agent's network if they believe that the agent shouldn't be limited to linear mappings. 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. Currently, it's possible to set activation functions and other aspects of a creature's network by passing arguments to high-level function calls.

Another open-ended characteristic of Eco-Simulator is it's use of user-defined resources. The user can create however many resources they would like the system to have, and specify which species of agents require which resources. They can also specify health bonuses or health costs when agent resource levels pass certain thresholds. In addition to collecting resources from the environment, agents can deposit resources, and even convert one set of resources into another. This allows for the creation of systems with intricate food webs and inter-dependencies. Resource settings and creation can be specified in a similar way to the creature's networks: through calls to a high-level API. Resource levels across the landscape can be visualized as the simulation runs.

Eco-Simulator uses an evolutionary process that is similar to natural selection. An agent's genetic material is the weights of its neural network. As some agents die and others reproduce, the weights that are best suited for survival and reproduction tend to be passed on. Mutation is used to explore new values for genes. As this process occurs, the level of genetic variability for each weight and the average variability across all weights are recorded and accessible through the user interface. This evolutionary process is similar to what is performed in a Genetic Algorithm, but with less top-down control. GAs have proven useful in 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). Eco-Simulator is capable of modeling many of systems mentioned above, especially if researchers are willing to develop custom inputs and actions for an agent's neural network.

The proposed project has the potential to be a convenient and useful tool for researchers in the field of AL, as well as an educational tool for students and others who are curious about ecology, evolution, sustainability, and neural networks. In order to make the program more user-

friendly, a point and click interface has been developed for running the simulation, visualizing the landscape, and observing the properties of the agents. The simulation can be run for different intervals of steps between each display, and different refresh rates. Additionally, it's possible to conduct experiments which involve multiple simulation runs and write results to a file. This is done through API calls.

Qualifications

Please see attached resume (last page).

Background

Eco-Simulator isn't the first multi-agent evolutionary simulation to model ecosystem dynamics. Holland's (1993) Echo model is perhaps the most similar to Eco-Simulator, in that it was designed to be general purpose, with a focus on how resources flow between agents. The Echo model allows agents to trade and fight, which can lead to both competitive and cooperative modes of resource exchange. In fact, their model demonstrated both competitive arms races, and cooperative symbiotic relationships. These are phenomena that I would also like to produce in Eco-Simulator. Despite commonalities in scope and intent, Echo does have some key differences with Eco-Simulator. 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 can be trading, attacking, or reproducing. This model results in simple reflex agents that are guided by the encoded appearance of other creatures. Eco-simulator is also capable of producing agents that base their decisions on the phenotypes of other creatures. However, the agents in Eco-Simulator are also

capable of incorporating other inputs into their decision making process such as: environmental inputs, internal state inputs, memory inputs (not yet implemented), and communication inputs (not yet implemented). 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 it's possible to get results even without ideally tuned networks.

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 has limitations in other regards that prevent it from being generalized to a wide variety of systems. 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 produce food. These specific aspects of the ERL model make it difficult to apply it to situations that involve multiple resources, or flexible agent-based predators. However, these limitations don't mean that ERL is incapable of producing meaningful results. 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 Eco-Simulator model won't incorporate reinforcement learning. However, I would eventually like to incorporate classical conditioning into the Eco-Simulator 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 that 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 could create an expectation. It would be interesting to see if the Baldwin effect would be present with this classical conditioning paradigm.

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. Eco-Simulator attempts to investigate similar concepts related to evolutionary progress, but utilizes different metrics. One metric is the standard deviation associated with a particular weight across the entire population. If the deviation is relatively small and the population appears to be well-adapted and stable, then it could be that deviations from that weight are being selected against. This is assuming a scenario in which mutation allows for exploration of new weight values. For example, the standard deviation for the weight associated with a bias node could be small if the agent requires constant levels of a behavior.

Despite its similarities with other frameworks, and the inspiration that it's received from past examples, Eco-Simulator is a unique system with a set of features that haven't been fully explored in past research. I believe that the flexibility and open-ended nature of Eco-Simulator with make it a beneficial tool for various lines of research.

Work Schedule

Lifecycle Approach

I have been using an iterative approach for the development of Eco-Simulator. Early iterations were focused on conceptualization, design, and prototyping, while later iterations have been focused more on the implementation of specific features and refining the user interface. I started this process by working on coarse representation of the overall software architecture of the program. This was followed by incrementally implementing components of the overall architecture and testing them with mock user inputs. More recently, I have been working on developing the user interface, implementing additional features for the neural networks, and creating a user-friendly API for ecosystem and creature creation. I have also done some recent work on the capacity to run experiments, and statistical analysis of genetic variability.

Progress Schedule

<u>September 2018</u>

- Developed framework for system.
- Created UML class diagram.
- Developed code for creating various aspects of the system.

October 2018

- Developed front end display for the system.
- Continued adding to code for system creation.

November 2018

- Tested environment, creature and network creation.
- Worked on software architecture for agent reproduction.
- Developed code for making copies of the creatures.

December 2018

- Tested running the system for multiple steps with creature movement.
- Tested creature reproduction.
- Fixed bugs related to copying creatures.

January 2019

- Worked on UI.

February 2019

- Transitioned the simulation to a separate thread.
- Worked on resource renewal.
- Fixed bug related to action queue.

March 2019

- Switched to displaying the environment with a texture.
- Tested a system with multiple species.
- Worked on allowing for resource conversion.
- Worked on phenotype networks
- Worked on user interface for creature info.

April 2019

- Worked on creating interesting scenarios.
- Worked on calculating averages and standard deviations for weights.
- Worked on writing statistics to a csv file.
- Continued working on creature info interface.
- Added layers of abstraction to interface for creating creatures

Proposal Statement

Functional and Non-Functional Requirements

Functional Requirements

- User can easily modify the system through function calls to a well-documented API.
- User can design an environment by distributing resources across a 2D grid.
- User can design an agent, including the agent's actions, resources, and neural networks.
- User can modify inputs, outputs, and hidden nodes of a creature's networks.
- User can run the simulation at different numbers of steps .
- User can pause the simulation and view it's state at that moment.
- User can save statistics and aspects of a simulation for further analysis.

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

An estimate for the time complexity of the program is: O(T*N*I*J*K^2 + S + T*N*Q) 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 (an approximation for the number of weights), S is the number of land spaces in the map, and Q is the average number of actions on a creature's action queue. The number of spaces on the map is only relevant when the state of the map needs to be displayed (for visualizing in the user interface). This formula doesn't include all of the steps involved making a copy of the system for the user interface. 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 = 9, J = 2, and K^2 = 100, S = 40,000, and Q < 10 . According to our time complexity, this simulation will require at least: 100 * 1000 * 4 * 3 * 16 ≈ 20,000,000 steps. Running this system on a mid-range laptop, I get an average performance time of 12.5 seconds. This performance is reasonable, but could be greatly improved through running aspects of the program in parallel.

One optimization that would have a profound effect on the performance of the application would be to run each creature's neural network on a separate thread or core. A creature's actions would still be carried out in a turn-based fashion by iterating through all of the creatures, but all

creatures would simultaneously decide on what actions to take based on the state of the ecosystem prior to any taking a turn. Further optimization can be achieved by running aspects of a creature's neural network in parallel, such as all nodes in a layer. Finally, 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. These optimizations could be important steps towards making this program into a more powerful scientific tool. The simulation aspect of the program is not closely tied to the Unity engine and could be run on other platforms that are more conducive to parallel programming.

Interface Requirements

- There is a point and click user interface with buttons and text boxes.
- The interface has different windows for visualizing a creature's basic statistics, resources, and neural networks.
- The user is able to navigate through the environment with zoom and drag controls.

Architectural Design Documents

The documentation for the program is incomplete, but here is a link: documentation. You will need to download the folder and open the index.html file to view it.

Tools Used

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

Methodology

Software Architecture

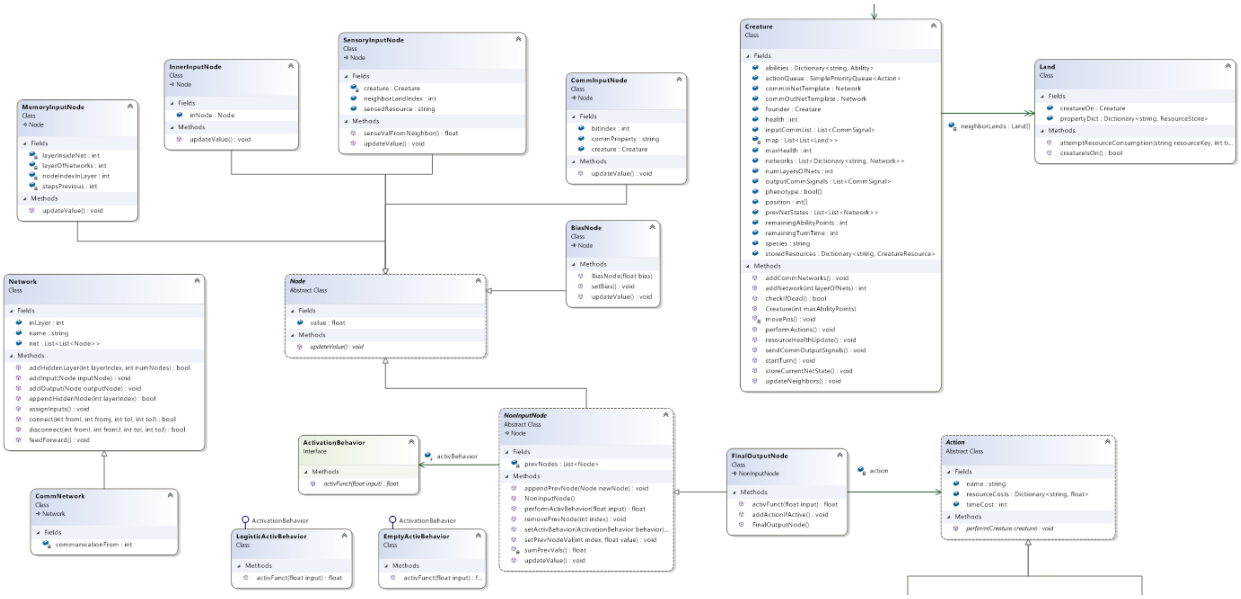

Figure 1. This partial class diagram shows several key classes including: Land for modeling the state of each grid space, Creature for modeling agents, Network for modeling a neural network in an agent, Node for modeling nodes in a network, and Action for modeling actions that a creature can take. Another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

Design Patterns

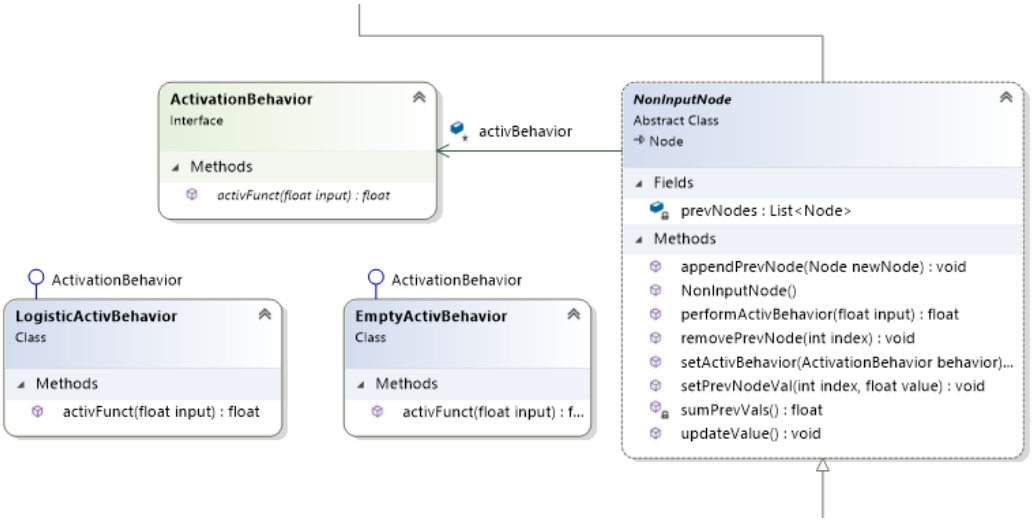

Figure 2. I am using the strategy pattern to allow nodes to switch their activation function at run time.

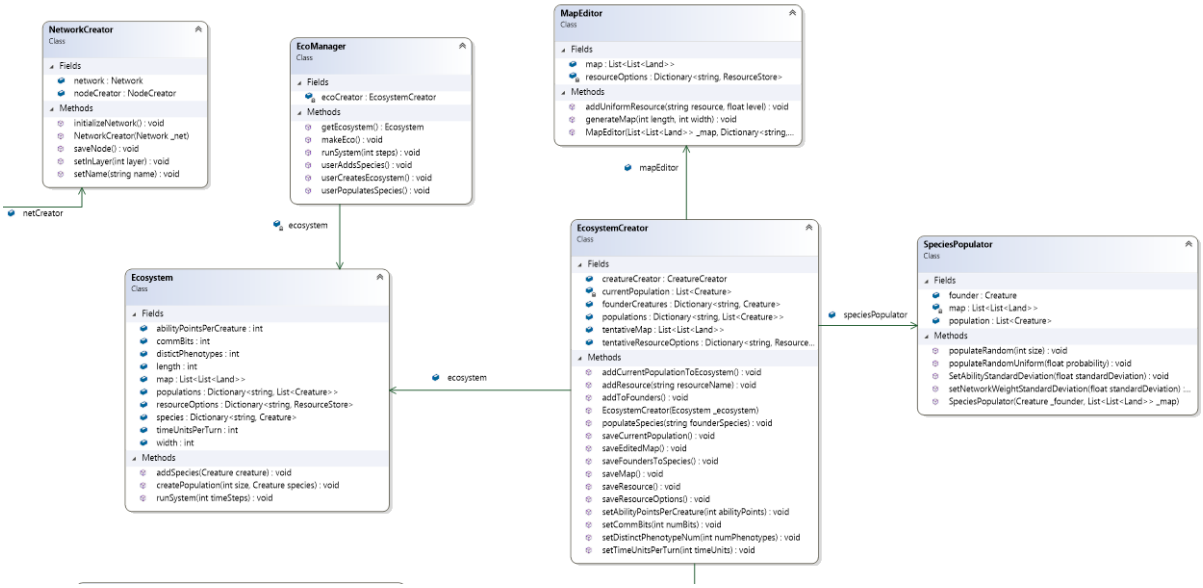

Figure 3. I am using the facade pattern to decouple the user interface from the back-end data structures. In the above example, EcosystemCreator, MapEditor, and SpeciesPopulator are all facades that modify the Ecosystem object. Each façade class typically corresponds to a separate user interface window.

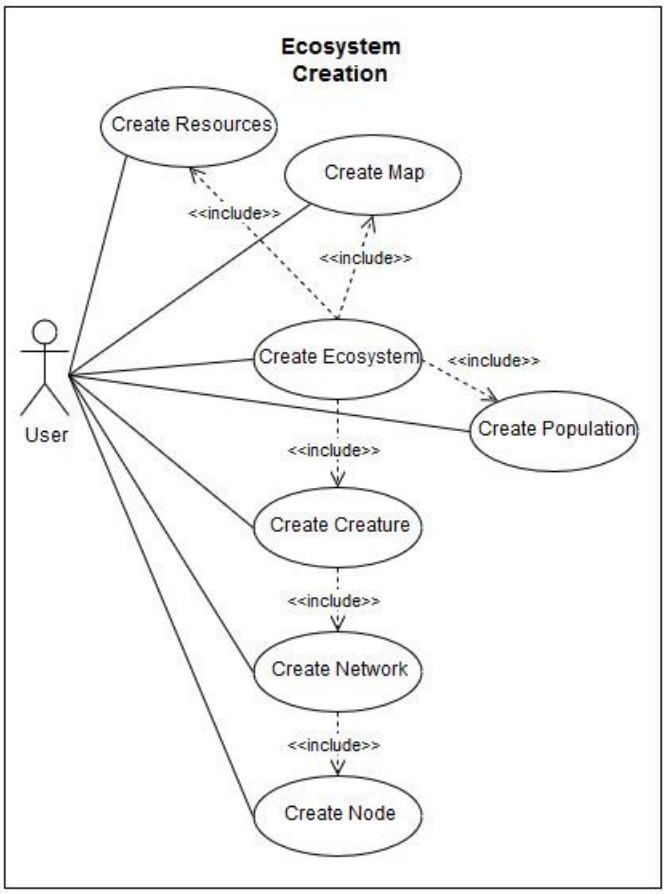

Figure 4. A use case diagram showing the different aspects of ecosystem creation. Note that creature (agent) creation is the only part of creating an ecosystem that involves further nested sub-tasks.

Trade-offs

One of the more difficult design decisions in building the software architecture for Eco-Simulator 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 the Ecosystem, Resources, Creatures (agents), Networks,

Nodes, and Actions. Creating a façade for each of these objects decouples the user API from the underlying data, which makes the program easier to maintain. It also helps to make the API more organized and intuitive.

Agents play a central role in this system, and 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. 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.

Results

One of the main goals of this program was to evolve agents that are adapted to survive in a particular environment. I have achieved this goal for simple environments where the creature is

required to find a balance between movement, reproduction, and resource consumption. Preliminary results show that the evolved agent is typically 10 - 100 times better than a randomly generated agent, although statistical analysis still needs to be performed. The evolved agent was created by allowing a population to evolve for a period of time (about 10,000 steps) and then taking the average value of each neural network weight across the whole population. Using these weights as the weights for our evolved creature, I was able to observe adaptive behavior. It should be noted that this approach is not ideal for a couple reasons. First, it's possible that there are several sub-populations in the original population that were using different strategies for survival and that taking an average of their behavior is less adaptive. Second, the environment changes substantially over time, so a creature that is well adapted to a later stage environment may not perform optimally in a new environment. Using a clustering algorithm to distinguish sub-populations, and training agents in well-designed environments could help with these downsides. Despite these limitations, the improvements In behavior have been substantial, and show that this program has the potential to find adaptive behaviors in a complex system.

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. Artificial life: organization, adaptation and complexity from the bottom up. TRENDS in Cognitive Science. Vol.7 No.11 November 2003.

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.

Appendix

Please follow this link to view my source code on GitHub:

https://github.com/BLayman/Artificial-Life-Simulator/tree/master/Assets/Scripts

BRETT LAYMAN

brettlayman7@gmail.com | 406-570-3916

EDUCATION

B. S. Computer Science 6/2016 – 5/2019
Montana State University
GPA 3.90

B. A. Behavioral Neuroscience 9/2006 – 6/2011
Western Washington University
GPA 3.31

SKILLS & PROJECTS

A.I. and Machine Learning

- Evolutionary Algorithm Comparison: compared a G.A., an evolutionary strategy, and differential evolution with backpropagation on a classification task.
- Swarm Based Algorithm Comparison: compared K-Means, DB-Scan, competitive learning N.N., P.S.O. and Ant Colony Optimization on a clustering task.
- I worked on various problems in my A.I. class including: search (A*), constraint satisfaction, and logic based agents.

App and Game Development

- Fractal Tree Visualization App: this app allows users the create their own fractal trees by setting various parameters. Published on the Google Play Store.
- Lunar Lemur Mechs: a 2D couch multiplayer physics game with an emphasis on cooperation. Supports up to 4 players with plug-and-play gamepad input.

Languages and Tools

- I'm proficient in: C, C++, C#, Java, JavaScript, and Python. I'm also experienced with Linux, Git, the Unity game engine, and OpenGL.

EXPERIENCE

TA for Introduction to C Programming 5/2017 – 5/2019
Montana State University (summers and current semester)

- Helped students during lab time. Graded labs

Web Application Developer 5/2017 – 8/2017
Montana State University

- Built a chat-style application for allowing teachers to share snippets of code with their students in real time.

Technical Supervisor 1/2016 –3/2018
Montana Sport TV

- Set up sound and video equipment. Managed the live broadcast: cameras, replays, commercials.