2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering
Education for 130 Years
Paper ID #39664

Self-learning Sandbox to Emulate Biological Systems

Mr. Benjamin Lubina, Gannon University

I am currently an undergrad in Cybersecurity at Gannon University, I run the school Cyber Defense Club, represented Gannon in challenges and competitions, and published a prior paper in the field of machine learning. I have 3 years of experience with software development, cyber risk assurance, and data analysis.

Dr. Ramakrishnan Sundaram, Gannon University

Dr. Sundaram is a Professor in the Electrical and Cyber Engineering Department at Gannon University. His areas of research include computational architectures for signal and image processing as well as novel methods to improve/enhance engineering educa

Educational Self-learning Sandbox to Emulate Biological Systems

Introduction

In nature, organisms evolve into their own niches in an environment over time, despite harsh changes in both biology and nature itself. This paper describes the development and observations from a self-learning sandbox intended to mirror and emulate the biological systems around natural selection and environmental pressures. Unique “pixel creatures” with random behaviors are generated in a pre-made environment and compete against others to survive and pass their genetic information to the next generation. This iteration of self-learning differs from standard neural networks by the method of which fitness is achieved. As opposed to the model of backpropagation, which applies changes to errors after the learning action has been done on the same model, this instead uses a generalized fitness approach where only the top performers of each generation may be given the chance to move on. Random changes called “mutations” will give a varied approach, act partially in place of a learning rate, and prevent a form of the local minima problem, as well as provide resilience to environmental change or change from possible competitors. We develop a simulator based on an emulation modular framework with a customizable environment and varying levels of complexity. This sandbox encodes genetic data and abstracts the concepts of behavior and genotypes using machine learning concepts. Besides inputs and outputs, organisms’ internal networks are completely dependent on its encoded “genes”, a bit string, which includes connections between neurons and the properties of the neurons themselves. Such a sandbox was developed, conclusions and comparisons to nature were made, as well as insights to possible expansion and application to education. In addition, there are evaluations of configuration changes and its effects are recorded for each unique trial within the simulator. Analysis on similar projects are provided and how they may proceed against some problems in design and theory. These applications are then put into the lens of educational advancement and the effects it may have on student development and its growth. A primary goal of this paper is to outline the results of this software for education and to this end, transparency was a primary focus in mind.

Emulating biology and its emulation in computing is a topic with various development and disjointed educational material over the years. Although there has been attempts to unify or catalog the results of this literature into decided outcomes, those that have not been outdated still have difficulty encompassing the depth of the field and keeping up with its advancements [1]-[2]. There have been a variety of analyses on the effects of different algorithms and variables to judge the effectiveness of certain designs. In addition, literature exists using these algorithms and environments to judge their effect on the evolution processes [3]. Teaching with these applications in mind has a positive effect on a target body of students and it can be safely assumed that interactions with such concepts would work well as an educational medium [4]. To this end, the following project, in addition to its current goals, will attempt to act as educational material made by students for students entering the field.

Primarily, there will be 2 cooperating processes that define the scope of the relevant simulation; Neural Networking and Genetic Algorithm Emulation. Neural Networking, or ANN, is a form of deep learning that makes decisions from inputted data in a way that is inspired by the brain. This subtype of machine learning process, uses layers of nodes called neurons to dictate its action from the input, with the input being the first layer of neurons received and the output being the last layer [5]. In a setting where the goal is to emulate a creature, this would simply act as the creature's brain; dictating what it should do in each moment and making its correlating action. The agent, being this single creature that perceives its environment through a set of inputs and acts, would not be able to change its ANN within its lifetime on its own. To optimize an ANN to fit a goal, there must be a mechanism from which the agent's actions can be evaluated and another to enact the results of this evaluation. In traditional single agent circumstances, each positive action will give the agent positive feedback, reinforcing the connections it had made to get to that conclusion. Likewise, for each negative action, or one that did not perform positively enough, the agent gets negative feedback which is normally interpreted to mean that it should weaken the signals that lead it there. This model of rewards and punishments, backpropagation, for their actions to achieve fitness is well suited to some tasks, such as recognition and classification, but for some tasks such as simulating biological life, there is a form of fitness evaluation that is deemed more appropriate [6][7][14].

Genetic Algorithms are a different form of fitness evaluation in that they use an iterative process to achieve fitness for an optimization task as opposed to backpropagation. In this context, these algorithms typically involve the use of many independent agents working towards the same task over the course of generations, with each new generation using a slightly randomized form of the most fit of the last generation [7]. This ensures that hopefully that the local minimal problem would be avoided or minimized, as there would be a large enough "mutation rate" to allow for ample exploration of possibilities [8][9]. To identify the agents from one another, each has its own unique "genome" which changes the action of its agent [7]. In a biological simulation, these agents' genome decides the structure of the ANN, which in turn will decide its actions within its lifetime. Instead of measuring fitness and deciding the best of each generation, it can be more advisable to instead set criteria that each agent must fulfill in order to survive, and surviving agents will have their genomes used as the starting point from which to deviate upon in the subsequent generation. In between generations, these algorithms would eliminate the underperforming "genomes" of some agents and promote those of others so that the following generations can further develop this evolutionary path. Overtime, this process can generate multiple distinct solutions to solve a problem, and in the context of evolution, has the potential to exhibit emergent properties, which is particularly desirable in a situation where a computer is emulating biological systems [10].

Section 2 goes over requirements of the project, including hardware and software used and its environment. Section 3 outlines the software's capabilities and its workings. Section 4 encapsulates noticed results, trends and takeaways. Lastly, Section 5 pertains to conclusions, which include its applications in other fields and appeal to education coming from both the software itself and the conclusions drawn from it.

Section 2: Project Requirements / Background

Machine learning in general normally takes a large amount of computer resources. Extending this to a large number of agents each with their own ANN would necessitate a similarly large resource commitment. Often, it is expected of machine learning applications to use or need dedicated workstations with nominally large amounts of processing power in the form of GPUs. However, in the case of application to education, the ease of running code and nominally low resource usage can prove beneficial for student interaction: as it proves more beneficial for classroom experience and usage if each student can experience results live and to create connections in the learning environment [11]. Although minimal resource usage was a secondary goal, the final version of software ended up meeting the expectation for running nominally well on a single GPU system, allowing for runtime on personal computing devices or GPU-equipped workstations, For both application and development of this project, software requirements are shown in Table 1 with the configuration that has been known to work. Version numbers are given, however unless explicitly stated, these are not necessarily required. Hardware specifications used on workstations will be listed further on:

Table 1: Prerequisite Software Requirements

#	Name	Version	Pre-req	Comments
1	Python	3.9.7	N.A.	Future versions (notably 4+) might prove incompatible with libraries used.
2	Nvidia CUDA Dev Kit	11.6	N.A.	11.6 is required for usage of present GPUs, however version 10.2 is a possibility with certain versions of PyTorch
3	Conda	22.9	1	Optional, but recommended for package management
4	PyTorch	1.13.1	1, 2	Site settings selected: Stable, Windows, Conda, Python, Cuda11.6 https://pytorch.org/get-started/locally/
5	matplotlib	3.6.3	1	Library for graph creation and data analysis
6	IPython	8.9.0	1	Library for interactive analysis
7	Microsoft VS Code	1.75.0	N.A.	Optional: IDE used, causes problems integrating with Conda

As stated before, much of this project, in its current form, has been done in a manner so that it can be run on desktop computers containing a GPU. As training, or “evolving”, completes, there will not be the need to run the software for days or weeks, computations on both tested platforms have been performed within minutes and lesser single GPU platforms are predicted to have similar results. This provides a boon to educational standards as previously mentioned as anyone may have and run their own version of the software. All software used is free and openly available under free licensing.

As for coding libraries and software choice, all software listed in Table 1 is extremely common in these fields, with PyTorch being the library that specializes in Machine Learning and is considered the standard being ubiquitous and among the few true python options in its field. It also proves relatively easy to learn to the uninitiated and it can be entirely contained in the Python programming language, which itself shares the same properties [12]. The Nvidia CUDA Dev Kit is a package that allows GPU usage in coded software and is used by PyTorch as a diver and pre-requisite package in order to give GPU support for machine learning, which is used extensively in this project and by extension, any other reinforcement learning applications. However, CUDA is theoretically optional as low cost student workstations/ integrated platforms might be able to run similar applications without use of the GPU, but however, this method of computation was not tested and would predictably increase the computational time to unacceptable levels for this use case [13]. Computational times generally take 10-15 minutes on a typical single GPU system for a 1000 population environment with 5000 generations. With only CPU usage, it can be safely assumed that these times would triple at minimum. Such computational times would limit the efficacy of classroom demonstrations and limit the educational effectiveness of the project, so therefore keeping runtimes to a minimum is an imperative goal if educational use is desired.

Both Matplotlib and IPython are used as libraries for the sake of convenience. Matplotlib helps in plotting complexities and easing development time for the Python language and its acts as a numerical mathematics extension NumPy. IPython however offers debugging possibilities, rich media, and interactive shell features that shorten the development time additionally. It also adds support for data visualization in unique ways from Matplotlib and methods for distributing and parallelizing computational resources [15].

Conda acts as a package manager with Python and allows the creation of unique Python environments, each with its own libraries and versions of codebases. Each of these environment acts without overlapping with other envs on the same system, which allows for differing projects with different requirements that would otherwise be out of date or unnecessary.

Figure 1: Workstation 1 (Lambda Vector)

For thoroughness and completeness, hardware specifications of both workstations used in this project is listed below. Workstation 1, a prebuilt shown above, is a high power environment that was used to test remote execution and applications. Workstation 2 is a more traditional desktop computer that contains a GPU. Note that although both workstation are optimized and used for AI applications, this software is believed to not require the computation strength that either station would provide, as surmised from resource usage data gathered.

Workstation 1

- 4x NVIDIA RTX A4000
- AMD Threadripper Pro 3955WX: 16 cores, 3.90 GHz
- 128Gb RAM
- 2x Monitor, 1x Keyboard, Mouse
- TrippLite UPS 1500 LCD
- Linux Mint 21.1 “Vera”

Workstation 2

- 4x NVIDIA GeForce RTX 2080

- Intel i7-8700 CPU: 6 cores, 3.20GHz
- 16Gb RAM
- 2x Monitor, 1x Keyboard, Mouse
- Windows 10 Pro N

Section 3: Software

Upon starting of the software agents are generated in accordance with preset arguments. These arguments will also set the size of the environment, which takes the form of a square checkerboard grid with optional "walls". Agents are seen as a single squares in this grid pattern. Each agent has its own object with attributes and an unchanging ANN which will determine its actions in the current generation. Each generation has a preset number of steps, and when those steps have taken course, a new generation is created with new agents populated with the properties of the successful last generation. This process will be discussed later. To visibly show the actions and results of these emulated biological agents, the software will take snapshots at pre-defined steps in chosen generations, and when the queued computation is completed, these snapshots are dumped as images in a folder. One of such snapshots of generation 0 is below, along with some of its meta settings. Each of these settings will be elaborated on and are all user inputted on startup:

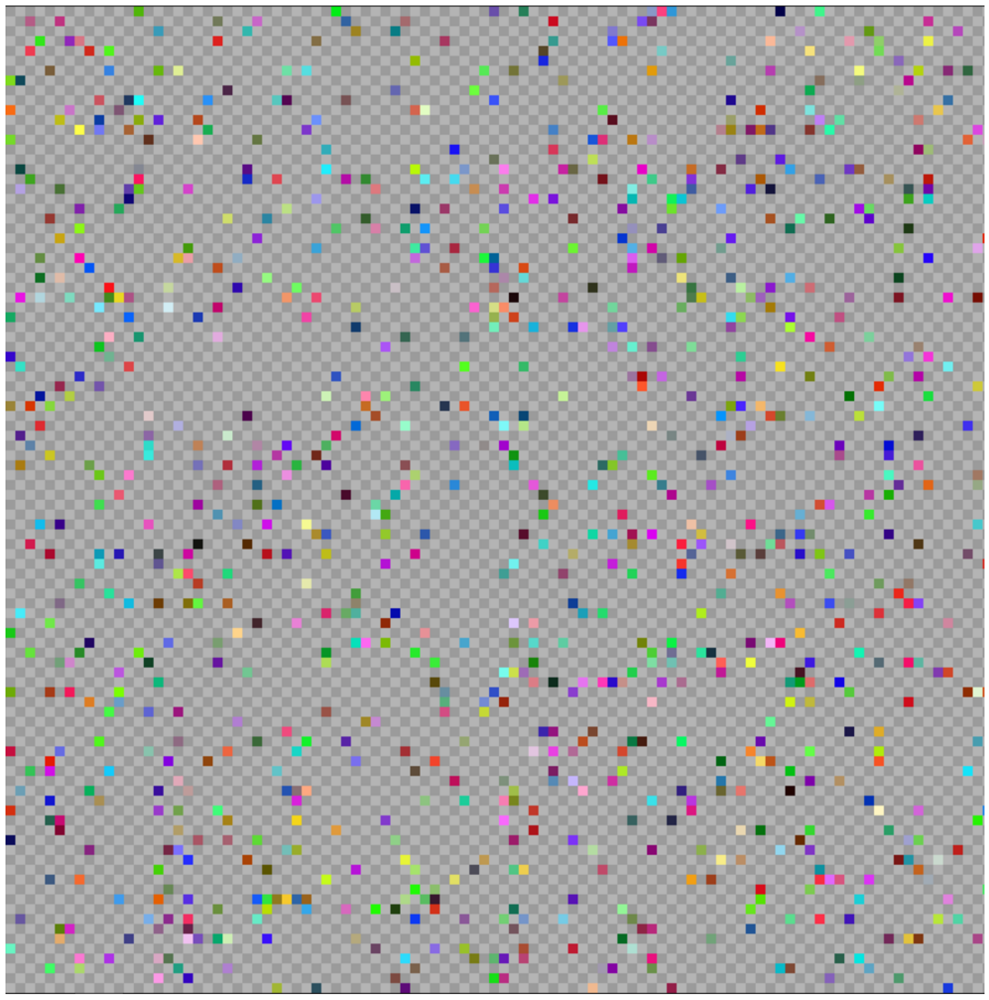

Figure 2: Base Environment

EnvSize:	100
Population:	1000
GenerationLength:	1000
AgentComplexity:	5
MutationRate:	.01
Current Generation/Step	0/0

Environment size and population are the two of the most visible variables to change before the begin of a simulation. EnvSize is simply the length of one of the sides of the square world in number of squares. These squares act as the smallest metric of movement and intractability for the agents. Each square contains data about whether an agent is on it, and contains data about whether an agent is in an adjacent square to it. Movement between squares is done in the 4 cardinal directions, and no diagonal movement is permitted. Population is the amount of agents created per generation and is the determining factor of performance. However, this does not necessarily represent the number of agents on the field, as in the event that an agent generates on the same square as another in the population phase of each generation, then the new agent overwrites the one that was on the square. Once agents are generated, then no more can be created or destroyed in that generation. These 2 variables are persistent throughout the entire simulation, and every time a generation ends, the world is remade using the same attributes. Runtime of these environments varies between seconds for populations below 50, to an hour at a pop of 5000. Below in Figure 3 are samples of worlds with varying values of these attributes, all of which were taken at first generation.

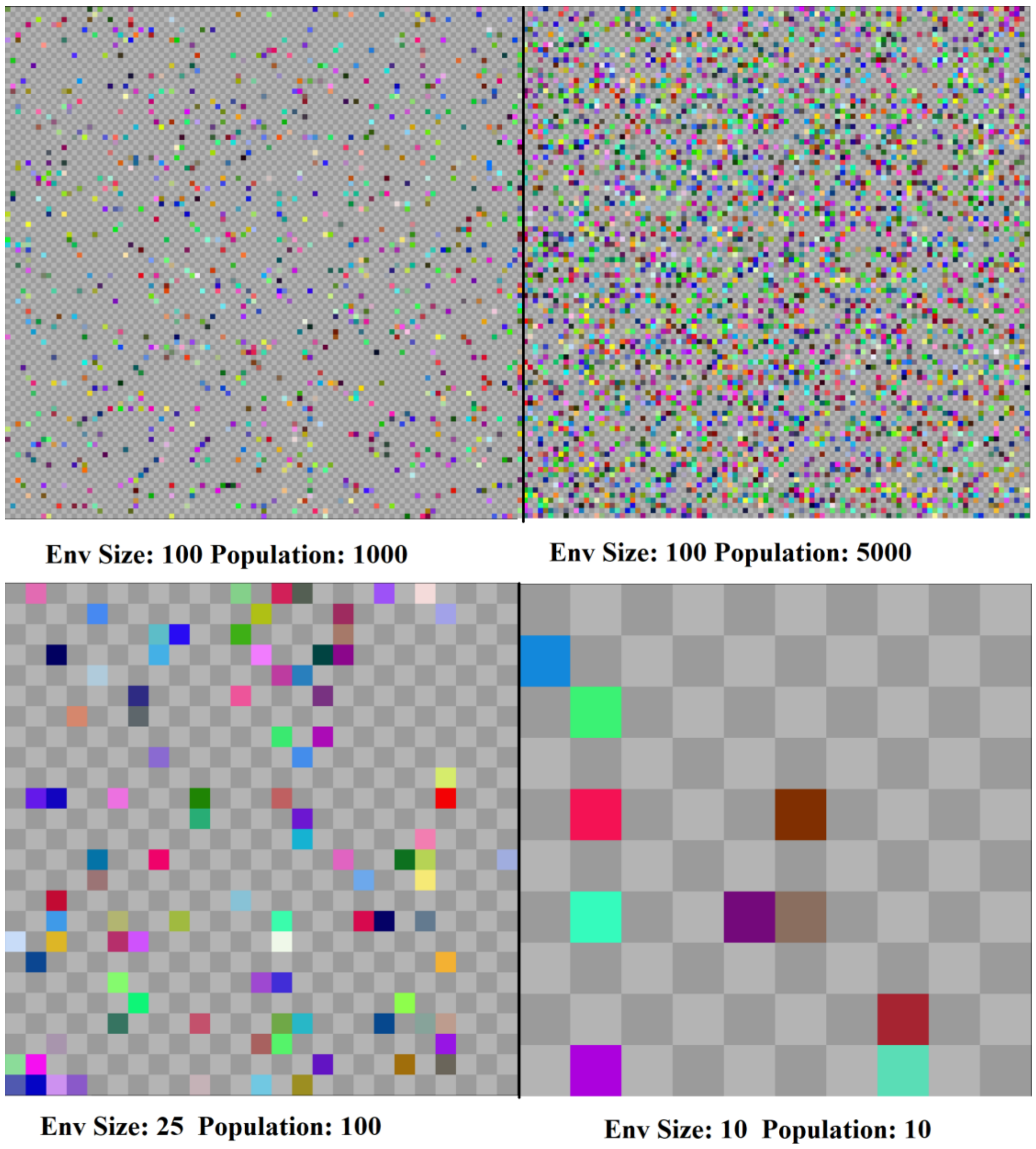
Env Size: 100 Population: 1000
Env Size: 100 Population: 5000
Env Size: 25 Population: 100
Env Size: 10 Population: 10

Figure 3: Environment Seeding Example

Generational length is simply how long a generation will last. The longer a generation lasts, the more time agents will have to achieve their survival requirements in order to pass genetic information onto the next generation. The integer value directly counts for the amount of steps that the agents will be able to take. The process for each step is as follows:

1. Receive Inputs – Each input neuron will be given a value gotten from the environment.
2. Generate Output – This is the computation stage, where each agent will figure out its output in accordance with its neural network. There can be only one output, which will always be the one with the highest **absolute** value.

3. Act – Agents on the field each take their turn to move depending on the order of the agents' generation. This stage also resolves conflicts and exceptions, such as an agent trying to move onto another, in which case it will not move.

When the steps are concluded, the generation ends. Longer generations have the hypothesized effect of allowing for more varied styles of play, while shorter generations force a more optimized approach of beating out the other competitors.

AgentComplexity is a metric that changes the depth of each agent's ANN, and therefore is the most computationally expensive variable to change along with population. Each agent shares the same input and output neurons, as well as the number of intermediary neurons, known as the hidden layer number, which is set by the AgentComplexity. All of these intermediary neurons are labeled as in the "Hidden Layer". Figure 4 has a flow chart made of one such ANN for a randomly chosen agent, and a key for inputs and outputs will be below explaining these neurons.

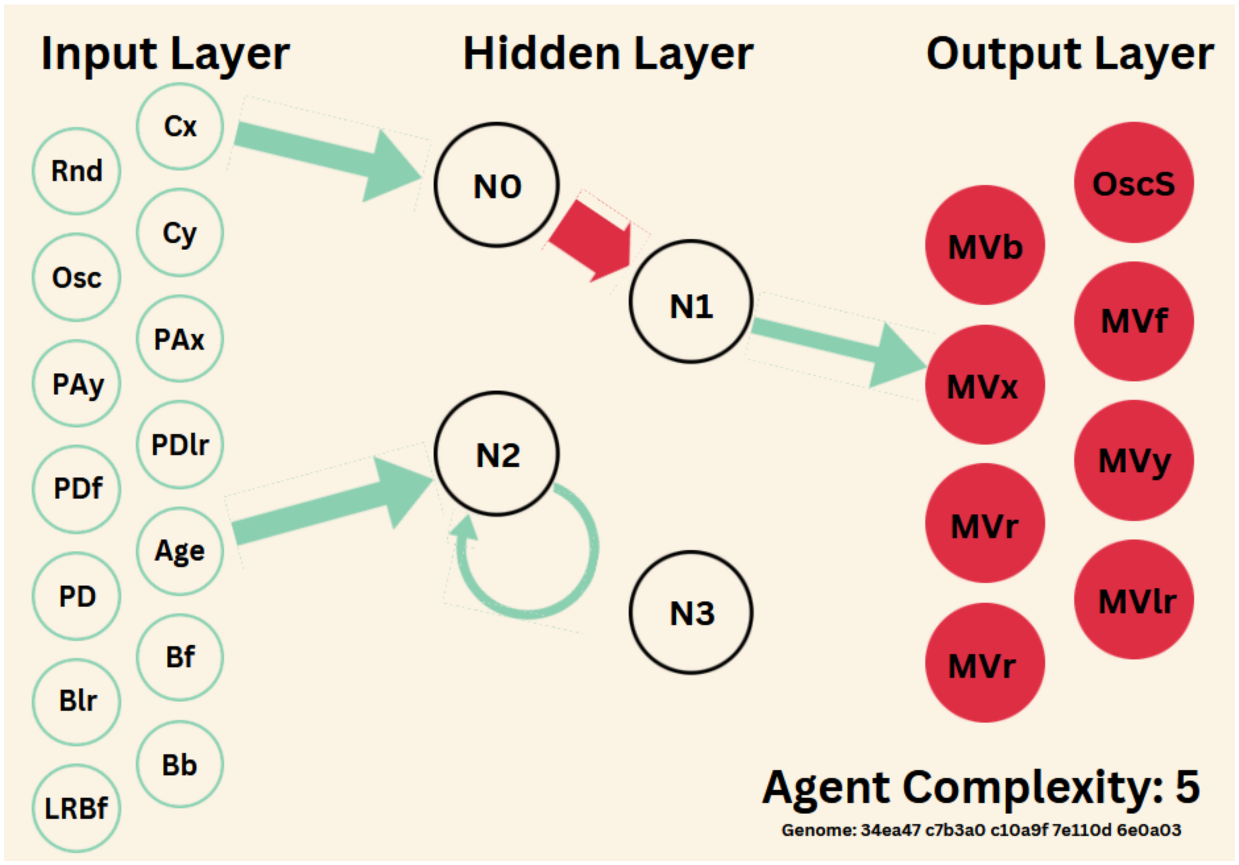

Figure 4: Example Neural Network of a Created Agent

Table 2: Input Neurons

Cx = Current position on the x axis in the world. -1 being on the left border and 1 on the right, 0 centered.	**Rnd** = Random number from –1 to 1
Cy = Current position on the y axis in the world -1 being on the top border and 1 on the bottom, 0 centered.	**Osc** = Oscillator, will go from 1 to –1 over the course of 10 steps as a sin wave function. Length of these waves can be changes from an output neuron: **OscS**
PAx = Previous action on the x axis, if agent moved last step left or right, make –1 or 1.	**PDlr** = Density of Agents in 5x5 areas on left/right side as a decimal -1 to 1, 0 if equal
PAy = Previous action on the y axis, if agent moved last step left or right, make –1 or 1.	**PDf** = Density of Agents in a 5x5 area in front of agent as a decimal 0 to 1
Age = 0 to 1 percentage of steps done in current generation	**PD** = Density of Agents in a 5x5 area around as a decimal 0 to 1
Bf = if an agent is in front of it, set to 1	**Bb** = if an agent is behind it, set to 1
Blr = if an agent is to the left or right of it, set to –1 or 1 respectively	**LRBf** = takes the percentage of agents in front of it to the border of the world as a decimal 0 to 1

Table 3: Output Neurons

OscS = Sets the wavelength of the **Osc** input neuron (Absolute –1 to 1)	**MVb** = Move backward (in relation to "front", defined at creation)
MVx = Move in the x direction (positive or negative)	**MVy** = Move in the y direction (positive or negative)
MVf = Move forward ("front" is defined at creation)	**MVlr** = Move left or right (+ or -, in relation to "front", also turns the front face)
MVr = Move randomly	**MVn =** Do not move at all

The number of neurons in the hidden layer of any given agent will be the AgentComplexity number minus 1. As seen in Figure 4, nodes of the hidden layer may connect to each other and often times do by random chance. Connections can take information from an input or hidden node and transfer it to either an output or hidden node. AgentComplexity also defines the number of these connections made in a sub-argument. These connections are defined by the agent's genome.

Genomes are gotten from the "winners" of the last generation that fit the prespecified criteria. Usually this criterion will be to reach a certain part of the map. Winners will have their genomes put into a pot of sorts and whenever the next generation is being made, each new agent will get the genome of a random winner of the last. This new agent will then undergo a mutation check which has the possibility of modifying these inherited genes before it is placed in the environment to start the next generation.

Genomes themselves are hexadecimal values that correspond to a bit string. Each gene in a genome is 6 hexadecimal digits, a 24 bit string, that defines a connection. A connection has an input, output, and an associated weight to go along with it that modifies the carried value. The first 5 bits state the ID of the node that acts as the input (which is either a sensory or intermediate neuron), the next 5 denote the ID of the output of the connection (intermediate neuron or action neuron), and the last 14 bits make a signed integer denoting weight which then gets compressed into a decimal value between -3 and 3. Large weights in the figure are defined with thicker arrows, negative weights are red, and vice versa for thin and green arrows. Figure 4 demonstrates a genome with 5 genes, and the utility of these genes can give hints as to how many generations have been undergone. For example, Figure 4 shows that this agent has drawn a simple negative correlation between its x position and its movement pattern. From this one can infer that this agent will always move to the y-axis in the center of the field, and therefore that the selection criterium is likely in the center of the board. The 2 unexpressed connections shown in Figure 4 likely mean that there hasn’t been enough time passed to allow for a beneficial mutation.

Mutation rate is the chance for a given bit to change from 0 to 1 and vice versa. Without mutation, there is no introduction of new connections, and those randomly moving (or stationary) agents from generation 0 will never change from their first random gen. The rate of mutation acts as a sort of learning rate parallel to other forms of deep learning and often runs into the same pitfalls when over or under-tuning this value [16].

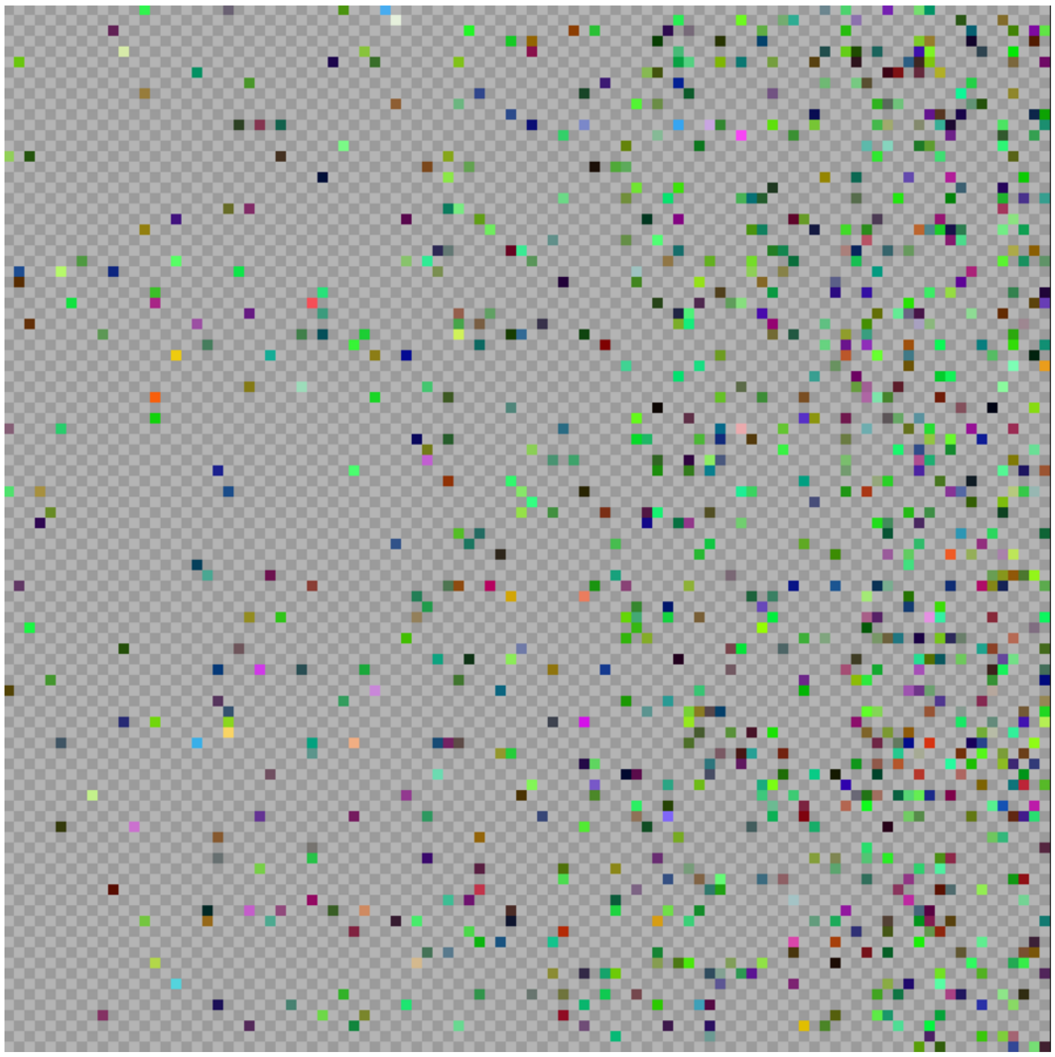

Figure 5: Genome Color Demonstration (Gen/Step: 50/100)

For visual purposes, genomes are also responsible for the color of each agent. This works by compressing and mapping the bits of each full genome into a hex color value, resulting in similar genomes exhibiting similar colors. Figure 5 is an example of this at work. The selection criteria was set to be the right half of the map, and it’s evident that not only has a concentration of agents formed at this edge of the map, but also a concentration of green shaded agents are prominent as well. This suggests that these agents share a similar genome, and therefore, similar behavior that has been successful for achieving the selection criteria. A video made of collaged snapshots showing the movement of this generation’s agents has supported the claim that their patterns behavior are similar; which has been to move right in a slow, sporadic, wandering fashion. Similar colors between 2 agents therefore represent similar genomes, which implies that they will have similar behavior. This allows one to inference the behavior of a group or archetype of behavior through color, without having to manually inspect the ANN of each agent.

Now that the workings of the environment have been shown, the following section will focus on observations and emergent properties of this system.

Section 4: Test Observation

Now that the workings of the environment have been shown, the following section will focus on observations and emergent properties of this system. A simple trial will be shown with a preset selection criteria zone. This trial below will have 3 figures showing the early, middle, and late development of generations with each snapshot taken in the middle of the generation. All staring points will share the following settings: EnvSize: 100, Population: 1000, GenerationLength: 250, MutationRate: .01

Figure 6 below shows the first trial. The selection criteria is shown as a red rectangle in the left-most image and encourages agents to be in the right hand side of the field. AgentComplexity is set to 7.

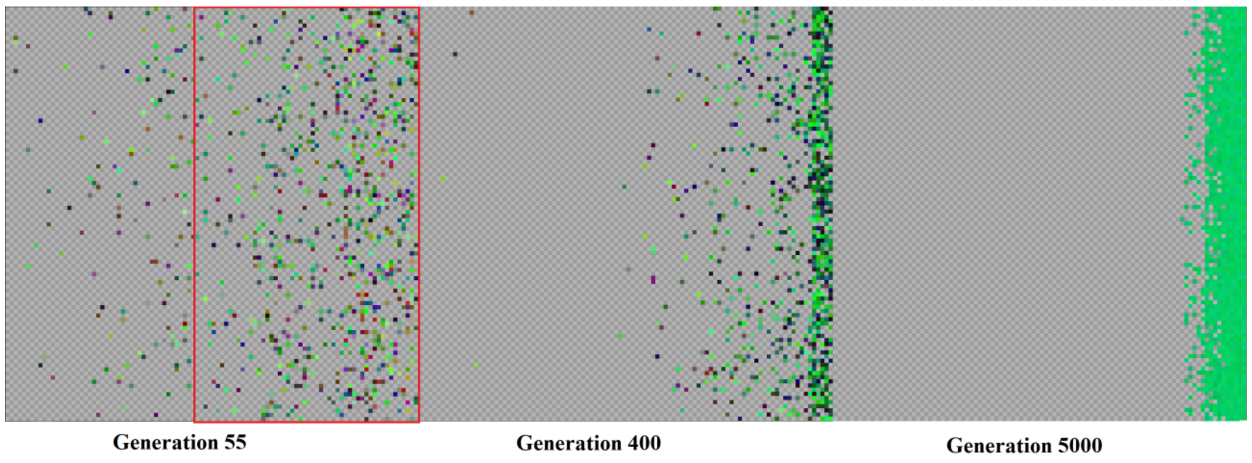

Figure 6: Eastern Block Fitness Trial

Early on, there already was a developed tendency to move in the positive x direction, as seen in Gen 55. Generation 400 exhibited an odd behavior where a certain demographic of genetically similar cousins crowded the last several columns, preventing others from coming in and showing an odd tendency to avoid wandering past a certain x value, shown by these certain shade of green. By generation 5000, genetic diversity is diminished and this behavior already shown in Gen 400 have been expanded to the entire population, crowding the right side and preventing movement.

Section 5: Conclusions and Future Work

The applications of such a project have cross-disciplinary effects, and it is clear that such ideas would be especially important to biology education. Biology in the modern era is dependent on bioinformatics, and integrating technology deeper into the classroom is ideal for the highest rate of effective and successful graduates [17]. This project can serve as a staging point for expansion by universities into bio computing for student education and more advanced projects to be built on. Section 4 has demonstrated emergent properties typical of biological systems that are the focus of any related course in the field, and more advanced and atypical environments will show equally more advanced and atypical behaviors whose analysis is beyond the scope of computer science and software engineering. These relationships are further reinforced upon repeated trails. Although we lack expertise in biology teaching pedagogy, we believe that there is firm interest and possible applications for such software in teaching of these fields, where it is most urgently needed.

Section 4 has also demonstrated the patterns and results coming from our chosen method of programming genetics and evolutionary concepts. Explaining the cause and effect of such processes has the effect of adding to the expanding knowledge base of these topics, which in an of itself is an educational goal. Papers found relating to similar topics tend to have a focus on either the evolutionary/biological aspects of this technological application, or the computing/mathematical applications of the algorithms behind them. By giving a look at both in an intertwined form, aspects of cross-disciplinary cooperation get enforced, and the uses of such information can be of use to both fields for further work, reference, or study, which further enhances the educational aim.

Computer Science and Software Engineering principles are inherent in the design of this software. There was never any mechanism that explicitly asked for a solution for a given problem. Solutions for selection emerged over time by random chance and were positively selected for. The properties of these interactions would be a positive point of focus for educational purposes and the process of controlling and managing these connecting systems is one focused on education.

Future work would likely follow one of two possibilities:

1. The first being one of software development: where this evolution simulator would be enhanced with modular and more varied functionality. Starting points for such would likely include making the AgentComplexity value set per agent rather than as a global variable as well as adding environmental variations in shape and possible forms of selection criteria in the form of food or a “kill” mechanism. Adding accessibility options like colorblind adaptations or easier to read text would allow greater educational flexibility.
2. The second possibility would be to test the application of this software in an educational environment for curriculum development and teaching pedagogy. There has been expressed interest by both computing and biology departments for cross-disciplinary interaction and this software proposes a method for this involvement. Analysis of both the results and the application process would be made to judge its educational efficacy in the classroom.

Bibliography

1. J. D. Schaffer, D. Whitley and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: a survey of the state of the art," *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, Baltimore, MD, USA, 1992, pp. 1-37, doi: 10.1109/COGANN.1992.273950.
2. Xin He, Kaiyong Zhao, Xiaowen Chu, AutoML: A survey of the state-of-the-art, Knowledge-Based Systems, Volume 212, 2021, 106622, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2020.106622.
3. LANDGUTH, E.L., CUSHMAN, S.A. and JOHNSON, N.A. (2012), Simulating natural selection in landscape genetics. Molecular Ecology Resources, 12: 363-368. https://doi.org/10.1111/j.1755-0998.2011.03075.x
4. Burgsteiner, H., Kandlhofer, M., & Steinbauer, G. (2016). IRobot: Teaching the Basics of Artificial Intelligence in High Schools. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1). https://doi.org/10.1609/aaai.v30i1.9864
5. Maind, Sonali B., and Priyanka Wankar. "Research paper on basic of artificial neural network." International Journal on Recent and Innovation Trends in Computing and Communication 2.1 (2014): 96-100.
6. Cheng, Bing, and D. M. Titterington. "Neural Networks: A Review from a Statistical Perspective." Statistical Science, vol. 9, no. 1, 1994, pp. 2–30. JSTOR, http://www.jstor.org/stable/2246275. Accessed 8 Feb. 2023.
7. Rajpaul, Vinesh. "Genetic algorithms in astronomy and astrophysics." arXiv preprint arXiv:1202.1643 (2012).
8. Ge, Rong, Chi Jin, and Yi Zheng. "No spurious local minima in nonconvex low rank problems: A unified geometric analysis." International Conference on Machine Learning. PMLR, 2017.
9. Atakulreka, A., Sutivong, D. (2007). Avoiding Local Minima in Feedforward Neural Networks by Simultaneous Learning. In: Orgun, M.A., Thornton, J. (eds) AI 2007: Advances in Artificial Intelligence. AI 2007. Lecture Notes in Computer Science(), vol 4830. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-76928-6_12
10. Kotrajaras, Vishnu, and Tanawat Kumnoonsate. "Fine-tuning parameters for emergent environments in games using artificial intelligence." International Journal of Computer Games Technology 2009 (2009)
11. Chen, Chunlei, et al. "Deep learning on computational-resource-limited platforms: a survey." Mobile Information Systems 2020 (2020): 1-19.
12. Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." Advances in neural information processing systems 32 (2019).
13. K. Afsari and M. Saadeh, "Artificial Intelligence Platform for Low-Cost Robotics," 2020 3rd International Conference on Signal Processing and Information Security (ICSPIS), 2020, pp. 1-4, doi: 10.1109/ICSPIS51252.2020.9340156.
14. Bitters, N. C., & Sundaram, R. (2021, March), Integrated Project Platform for Student Research and Curriculum Development Paper presented at 2021 ASEE North Central Section Conference, University of Toledo, Ohio. https://peer.asee.org/36343
15. Pérez, Fernando, and Brian E. Granger. "IPython: a system for interactive scientific computing." Computing in science & engineering 9.3 (2007): 21-29.
16. Burger, Klara Elisabeth, Peter Pfaffelhuber, and Franz Baumdicker. "Neural networks for self-adjusting mutation rate estimation when the recombination rate is unknown." PLOS Computational Biology 18.8 (2022): e1010407.
17. Pevzner, Pavel, and Ron Shamir. "Computing has changed biology—biology education must catch up." Science 325.5940 (2009): 541-542.