Mario AI using Neuro-Genetic Evolution

Victor Nguyen

Tommy Swenson

Colin Buck

August 15, 2019

1 Abstract

This papers focus is using neuroevolution and the NEAT process to create a neural network that is capable of playing the game Super Mario Bros though the FCEUX emulator. Manipulation of the bias mutation rate showed that the resulting populations could develop or destroy useful behaviors that would allow out neural network to get past obstacles quicker. We found that by increasing the bias mutation rate of our neural network it could more quickly learn these behaviors, however it also had the potential to mutate away from those behaviors resulting in the neural network going backwards in progress. In addition when changing the weight mutation rate the result was that once the neural network learned how to get passed an obstacle it continued to build on the learned behavior. Some obstacles within Super Mario Bros were difficult for our neural network to pass, this was associated with our fitness function. we believe modifying the fitness function to introduce a benefit to individuals who manage to overcome difficult obstacles would be an effective way to solve this issue.

2 Introduction

The problem that our group is focusing on is creating an AI that is able to complete levels within the game Super Mario Bros. The approach that our group will be taking towards completing this problem is through the use of neuro-genetic evolution. To create a functional genetic algorithm capable of completing a level we will have to identify an initial genetic representation (initiated randomly) and define a fitness function. The initial genetic representation will run simulations and will be scored based on the definition of the fitness function that we define. After running simulations for that specific generation, a select few species of the generation will be selected for breeding in the next generation. The pool of children composed of a majority of the highest scoring children and a few low scoring children to ensure genetic diversity. Genetic diversity is important within our problem such that our AI does not end up at a local maxima and halt progression. After getting a successful AI that reached our distance goal, our group has decided that it would be an interesting aspect to change the weight and bias values. By changing the weight and bias mutation rates we expect to see a variation in generations to complete the level. Specifically the bias mutation rates will be interesting to analyze as bias's often improve a population's generation when done in moderation, but can destroy good members if mutation occurs too quickly and too often. This allows the generation to potentially rapidly improve its fitness score, but it can also cause it to plummet with a series of bad mutations that carry over.

Video games are providing great test environments for neuro-genetic evolution. One reason they are such great test environments is that they can easily provide fitness functions. In Mario the distance traveled by each individual provides for a simple way to evaluate Mario's performance, and will be the basis of our fitness scores throughout our experiments. Creating an AI that performs well in Mario will not result in a revolutionary AI that changes the world. However, by testing concepts in video games, we can quickly learn how to pick optimal parameters which will lead to creating better AI in the future.

3 Literary Review

3.1 Why Neural Networks

Neural networks are a very useful concept in programming, as it allows computers to use observational data to solve problems more efficiently. [4] states neural networks are able to generalize information, leading to reasonable outputs for inputs that aren't necessarily specified. Neural networks are a way for computers to solve complex problems that take an insurmountable amount of computations, and therefore time, through decomposition and a distributed structure. The implementation of the learning algorithms used are an important concept to take note of truly understand neural networks. Learning algorithms are essential to the success of neural networks as they are what allow neural networks to process complex data sets.

3.2 Learning Algorithms

[6] presents a learning algorithm called extreme learning machine (ELM) which main benefits are "generalization performance at extremely fast learning speed." The authors of [6] state that feedforward neural networks are "far slower than required and have been major bottleneck in their applications for the past decades." Two key reasons behind may be: 1) the slow gradient-based learning algorithms are extensively used to train neural networks, and 2) all the parameters of the networks are tuned iteratively by using such learning algorithms". In their testing of ELM they concluded that ELM has a much faster learning speed and is able to find solutions in a more efficient manner compared to that of gradient-based learning algorithms. However, one of the drawbacks to using ELM is that it cannot be used for feedforward neural networks which have more than one hidden layer.

[2] creates a comparison of how different neural network training algorithms affect blood pressure estimates. The three algorithms that are analyzed in the paper [2] are: "steepest descent (with variable learning rate, with variable learning rate and momentum, resilient back-propagation), squas-Newton and conjugate gradient that are used to train two separate neural networks". The different training algorithms were compared based off of estimation error and training performance.

[11] introduces a new learning algorithm for multi-layer feedforward networks called RPROP. As described in [11] "RPROP performs a local adaption of the weight-updates according to the behaviour of the error function". The algorithm modifies the update-values for each weight according

to the partial derivatives. Through the testing that the authors of [11] conducted RPROP shows promising results such that the number of learning steps is significantly reduced in comparison to the generic gradient-descent.

[7] states that backpropagation algorithms are inefficient compared to parallelizable optimization techniques. The authors conducted a series of experiments based on small boolean learning problems and the noisy real-valued learning problem of hand-written character recognition. The results from [7] showed that the optimization technique, specifically the Polak-Ribiere algorithm, offers superior convergence properties and significant speedup over the backpropagation algorithm. The authors also state that this type of implementation works well in conjunction with parallelization.

3.3 Genetic Algorithms

Now that we understand the concept of Neural Networks, Learning Algorithms, and their use cases, we can talk about some of the possible specific Genetic Algorithms that turn these concepts into a reality. There are a few ways that a self-learning AI can learn to consistently complete an Infinite Mario Bros level. We will focus on methods revolving around concepts like Genetic Algorithms and Neuroevolution, but it should be noted that there are other techniques that can be used.

One of these non-GA techniques is using something called Dynamic Scripting, as noted in [9]. This methodology uses a Rule Base that contains a list of all possible rules that can be applied into a game. Each rule is given a weight that relates to how well that particular rule performed for the agent in previous runs. Rules are chosen by the agent based on a combination of these weights, and a roulette-wheel, and the weights are then further updated based on the performance of the agent using those rules.

[10] creates this AI with a Genetic Algorithm in its most pure form. The GA obtains its rule set from excel files, and these are the rules that determine how each generation will turn out. It uses an AI benchmark to determine how each action, and each set of evolution parameters, stand up against the other evolving children. It uses the "$[\mu + \lambda]$ Evolution Strategy" technique to do the cross over breeding. This way, the individuals that performed the best from the previous gener-

ation are kept, and are now considered the parents. Each of these parents produces λ/μ children. These children, combined with the parents, are now considered the next generation. It uses various fitness algorithms to assign probabilities of selection for each of the possible genes.

[12] creates a GA, and also utilizes what is called a "Cuckoo Search with Levy Flights". This search algorithm is used to find a way to the end of the level, and supposedly works quite well in this situation. This algorithm has two parts. The first part is creating a new "nest", which is generated after randomly walking in a direction or performing some other action. After the new nest is placed in this location, it is compared to the old nest. The better of the two nests is kept, and the other is no longer considered. With the second part of the algorithm, the nests that are considered the worst are thrown out and replaced by new random nests, and the cycle continues. The Levy Flights are an addition that allows for randomly making a large jump in a new direction, as a way to speed up this search process. Sometimes these long "flights" work out, and sometimes they don't. But overall they help explore the problem space at a quicker rate, which is largely beneficial to this kind of problem in regards to reaching the end of the level.

[5] utilizes a combination of a GA and a Finite State Machine. The finite state machine details the current state in which Mario is in, categorized as Running, Jumping, Speed Running, or Speed Jumping. In order for one of these states to be changed, there has to be some trigger that causes such a state change. Each of these connections between states is called a transition, as discussed in [8]. In our case, Mario is our actuator, who performs actions based on inputs received by sensors that are checking the environment. Some of these states include seeing an enemy, seeing an obstacle, seeing nothing, and many more. The genetic algorithm aspect is rather standard. A random population is initialized with various genes. The members of this population use a combination of the Finite State Machine, and their genes to attempt to complete the level. Then the members' fitness are evaluated based on the distance traveled in the level, and a new population is created based on mutations and how well certain members of the population performed.

3.4 Neuroevolution

Neuroevolution is a process that utilizes mutations and crossovers through generations to create populations of increasing levels of fitness. Neuroevolution is a good solution to solving Infinite Mario Bros because ”NE searches for a behavior instead of a value function, it is effective in problems with continuous and high-dimensional state spaces.”[15] It is important to have a benchmark for testing machine learning algorithms, [16] turned the game Infinite Mario Bros into a benchmark for testing machine learning and Neuroevolution. The authors of [14] state that neuroevolution is more efficient than the general process of neural networks implemented with deep learning which utilizes backpropagation and stochastic gradient descent algorithms to modify neural network weights. This is due to the fact that reinforcement learning techniques often show inconsistent results. This paper also discusses how to deal with the lack of diversity that can arise in neuroevolution. To prevent neuroevolution from reaching a local maximum and not making any improvements creative evolution techniques must be used. One neuroevolution algorithm that promotes diversity is called novelty search. This search rewards networks that behave simpler relative to the previous networks in the search, which results in a more diverse population. To create great AI one must promote diversity, ”A drive towards diversity is important when considering neuroevolution as a possible route to human-level AI.” [14]

NEAT Another important part of evolving neural networks through neural evolution comes from changing their topologies. NEAT (NeuroEvolution of Augmented Topologies)is a powerful method for artificially evolving neural networks [15]. The NEAT method offers an efficient solution to the problem of a population with diverse topologies, by using speciation based on their similarities diversity, and preventing a species from taking over an entire population is promoted. Using NEAT allows for meaningful crossover between individuals with different genetic length, which results in increasing complexity. [1] NEAT also has other potential opportunities specifically within video games. The rising popularity of video games has created a excellent test bed for artificial intelligence techniques. This is in part because video games ”carry the least risk to human life of any real-world application.” [13]. Real time NEAT, or rtNEAT is an attempt to create a non player character that can learn based on the actions of the human player within the game. NEAT was originally designed to not allow human interaction while the evolution is taking place. By using continuous replacement rtNEAT introduces a new agent into the population are repeated at regular time intervals.

Q-Learning Another learning algorithm that also utilizes Finite State Machines is called called Q-Learning, a concept discussed in [17]. It is considered to be a similar learning environment to the idea of a state machine. In this Q-Learning, each finite state is assigned a unique state number used to set values based on the environment's details. States and actions are combined to create what is called a state-action pair, and is the basis of Q-Learning. Each state-action pair gets put into a table, and a value is assigned to each of these pairs based on how well that state-action combination performed, i.e. based on the reward received with that state-action pair. There are many benefits to having every single state+action combination paired together with an associated value based on that combinations performance, but there are also significant downsides. The most important of which deals with the complexity of the problem at hand. As the environment for a problem becomes more and more complex, it can lead to a significant number of state-action pairs. This could make Q-Learning take a substantial amount of memory space, potentially affecting performance the overall quality of learning.

Neuroevolution Approach to General Game Playing Super Mario has little domain specific knowledge when compared with some modern 3D video games. However the 2D world of Mario is complex enough to create interesting dynamics with relatively few agents. A similar projects include Matthew Hausknect's paper in which he applied four neuroevolution algorithms (including NEAT, and HyperNEAT) to 61 Atari games. [3] Studying so many games at once provided a great challenge, dispute each individual game being relatively simple, presents a daunting task for a video game playing agent. However using neuroevolution is a promising approach. Comparing the different neuroevolution methods reveald that encoding algorithms like NEAT outperformed indirect encoding methods such as HyperNeat, on low-dimensional problems. However for learning larger networks, HyperNeat was the only successful algorthm of the four tested.

4 Neuro-Genetic Evolution Basic Algorithm/Implementation

Our group has decided to use a NEAT (Neuroevolution of Augmenting Topologies) algorithm to solve our Mario AI problem. Our group believes that using neuroevolution will be the best solution for our problem since each level in Super Mario Bros has a different set of similar obstacles obstacles. In doing so, we hope to optimize a set of neural networks that are able to complete levels within the game disregarding the placement of any obstacles. Our plan is to model a simple neuroevolutional process which consists of:

1. Creating a population of neural networks.
2. Run simulations on the neural networks within the population.
3. Evaluate fitness of neural networks.
4. Pick parents based on fitness scores.
5. Apply crossover/mutation.
    - The crossover and mutations be computed taking into account weights and biases, which we will change the value of so we can see how the alterations affect the learning process
6. Repeat steps 2-5.

5 Our Implementation

5.1 Initialization

Initialization of this experiment first requires cloning the "super-mario-neat" public GitHub repository, and building the programs based on the source code. The configuration file can be altered as desired to change various parameters in the Neural Network trainer. The parameters that we chose to alter when gathering our data were the bias mutation rate and the weight mutation rate. Then, using Python3, the main driver program is able to be ran. This Neural Network trainer can either start from generation 0, or can continue evolving from a previous genome's generation. The NEAT algorithm creates and evolves a neural network for the Infinite Mario Bros level, and the FCEUX emulator both plays the level and displays everything that happens to the screen. This allows us to actually watch the neural network evolve, and see how the mutations affect the way Mario interacts with the environment in an attempt to make it further down the level. As the Neural Network evolves, the program also prints to console the statistics about each species. It shows information such as: the ID member of the species, the age in terms of how many generations the species has existed, the number of members in the species, and the average distance that species travelled in that generation.

5.2 Fitness Function

Our fitness function was an integer directly based on the distance traveled for each individual within the population. The maximum fitness that could be obtained was 3252, which is the horizontal distance traveled when the level has been completed. When Mario first spawns into world 1-1 he starts at distance 40, so if Mario ended with a fitness of less than 40 it means that Mario started by running backwards, which speeds up the training process by eliminating individuals who decide to run left at the beginning of the level. Every 50 actions a distance was recorded after another 50 actions the distance is compared to the previous distance, if they are the same then no forward progress was made so terminate the execution, and run the simulation with a new member of the same or different species. Additionally, execution terminates through the other normal means of losing in Mario, such as getting hit by enemies or by falling into a hole.

5.3 Crossover/Mutation (Weights & Bias's)

The crossover and mutation rates for the weights and bias' are preset values that are set while initializing the population through the configuration file. The crossover and mutation rates are very important as it causes the population to not only make the AI learn, but also to introduce diversity into the population. Diversity is important within a population so that the AI controller does not continuously get stuck a certain point. This occurs when each of the species in a generation ends up in the same ending location, causing advancement to come to a halt. The mutations can

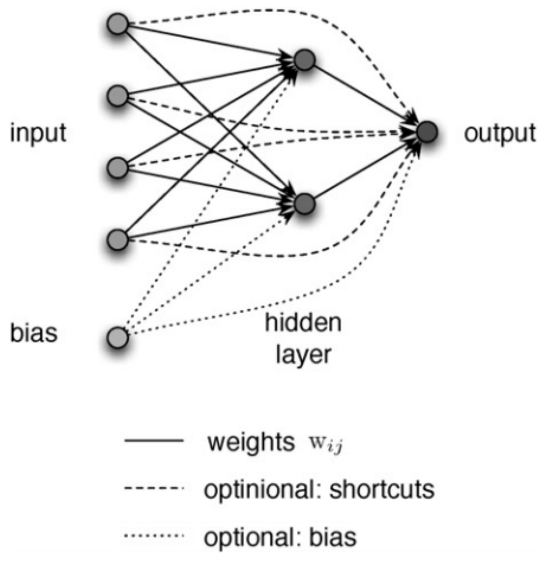


Figure 1: Neural Network through forward-feedback

force a species to evolve, or create an entire new species that may act differently near the spot the others got stuck in, potentially leading to a breakthrough in distance advancement. However, it is also possible that the probability of mutation could potentially make the next generation generate a population that does not perform as well as the previous population. An example of this is if the mutation rate is too high, then there is a higher chance that a well-performing member or species will get mutated, potentially making them worse than they were. Thus, this allows for an interesting topic of experimentation such that we will be able to see how the AI controller performs while changing the probability of mutation. Figure 1 demonstrates how the bias and weight is incorporated into the process of selecting a parent/species for the next generation.

6 Results

The results consist of the number of generations that took the neural network to reach 40% completion of level 1-1 in the game Super Mario Bros. We measured only up to the 40% mark to save time as we found that the 40% mark of the level consisted of the majority of obstacles that the neural network would have to learn to pass the level. While changing the weight mutation rate and bias mutation rate we kept all other variables consistent to ensure that the change in performance would be solely based on the given variable and a variable amount of probability.

The figures and tables of our results are in the Appendix, located at the end of this essay.

7 Analysis

7.1 Fitness Data with Default Weights and Bias

There were many interesting results that were printed to the terminal by the program while each generation was running. The aspects that were particularly interesting were in regards to each generation's overall average fitness score, and each species' fitness score in that generation. Generation 0 continuously ran into the first enemy every single time. At this point, it did not know how to jump. So it makes sense that the fitness score in the first generation was 312, which is the location of the first enemy, as seen in *Figure 6*. It is also interesting that for 14 generations straight, the AI made no notable change to this strategy other than running directly into the first enemy, as seen in *Figure 4*. It is hard to determine exactly what mutations were going on behind the scenes for these first 15 generations, but it was intriguing to see that it didn't make any noticeable changes. However, as seen in *Figure 8*, the NEAT algorithm realized it was not making any progress. So what it did was purged almost every species, since they had all stagnated to obtaining the same results every time. With all of these species purged, there became more room for mutated species and members to form. *Figure 9* shows that just 4 generations after this purge, the AI started making noticeable changes. It learned how to jump, and was able to finally make it past the first enemy, and even did so in multiple species within the generation, despite no noticeable changes occurring in the entire first 14 generations.

Another interesting part of the results is in regards to the data found in *Figure 4* and *Figure 5*. Generation 49 was doing very well, scoring over 600 on the fitness test after having a steady increase in their fitness from each of the prior generations for a while. However, from generations 50-52, we see the average fitness score steadily decline. This is quite interesting to see because it looked like the AI was performing so well, making it past more and more obstacles each generation. However, there are always random mutations that can come into play. In this case, there was a series of mutations on many of the species in each of the generations that caused their performance to decrease. Species ID 178 had one of the most unfortunate series of events, decreasing by around 100 each generation. This goes to show that the AI will not improve every single generation. Since there is randomness on top of how everything is chosen for breeding, there is a possibility that a species that performed well one generation could get a mutation that causes it to perform more poorly the next generation. Just like there is a probability that a generation that performs poorly one generation could get a mutation that causes it to perform much better in the following generation.

7.2 Relationship between Bias-Rate-of-Mutation/Weight-Rate-of-Mutation and Number of Generations to reach goal

Analyzing the *Figure 2* and *Figure 3* using an R^2 equation created by an exponential trend line we notice that by changing the bias mutation rate leads to a lot of variability and changing the weight mutation rate has a fairly linear result in respect to the number of generations that it takes to reach the goal. We believe that the reason that changing the bias mutation rate leads to a lot of variability is due to the fact that having a high bias allows the neural network to learn faster. We believe that the reason that the weight mutation rate has a fairly linear result in respect to the number of generations that it takes to reach the goal is due to the fact that it allows the previous species to have a greater influence on the mutations that it breeds. An example of this is when a neural network learns how to get past a certain point in the level it takes the species that was able to get past the point and mutates around that species. Thus, the next generation that is bred will also be able to get past the given point as well. Referring to *Table 1* and *Figure 2* Δx=7 between point 1 (Bias Mutation Rate = 0.0254) and point 2 (Bias Mutation Rate = 0.0509) compared to Δx=67 between point 2 (Bias Mutation Rate = 0.0509) and point 3 (Bias Mutation Rate = 0.1018. This relationship shows us that the general variability of generations needed to reach the goal increases the greater the difference between the mutation rates are. We suspect that increasing the bias mutation rate increases the probability of the neural network doing a various set of obstacles.

8 Conclusion

After observing areas where our neural network struggled it became evident that our fitness function could be changed to help encourage species to avoid some of the major problems our neural network faced when learning to play Super Mario Bros. One of the major hurls for Mario was the pipes. Pipes in Mario force the player to jump over them, however in order for Mario to learn this behavior he must randomly figure it out though a mutation, and has no incentive to avoid running face first into the wall until the simulation terminates. By changing our fitness function to include a break though fitness bonus this could have helped our AI learn from their mistakes. This could have been implemented in our fitness score by tracking long periods generations with no progress, then awarding a bonus fitness score to the individual who finally passes over the obstacle that has been causing all the species issues. Another possible change to our fitness function could be to include points for things like collecting coins and gaining power-ups. Gaining coins would lead to gaining extra lives in the game, and collecting power ups leads to more distance traveled they would be good measurements of fitness.

Our results indicate that increasing the bias mutation rate increases the probability of the neural network to learn how to get past an obstacle at a faster rate. However, it is very likely that the high rate of mutation also has the potential to make the neural network mutate in such a way that it loses the ability to get past an obstacle. Our results also indicate that increasing the weight mutation rate allows the the previous species in the neural network to have a higher influence on the future generations that are to be bred. This mechanic allows the neural network to consistently clear an obstacle once it as learned how to get past it. Based off of our results if we were looking for optimal configuration of the neural networks we would generally be careful with the bias mutation rate and have a modest value for the weight mutation rate. However, we would like to acknowledge that our results were based off of a fairly small sample size. If we would like to have a more concrete conclusion on the issue it would be beneficial to conduct this experiment on a much larger sample size to reduce the deviations due to probability.

9 Future Work

Since our Genetic algorithm was only trained on one level of Super Mario Bros the question arises of weather the neural network is leaning how to play Mario, or just how to beat one level. It would have been interesting to see how our neural network would have reacted when it reached situations it has not seen before after being trained on level 1-1. One way to do this would be to place Mario on level 1-2 and observe weather the Neural network is able to successfully avoid obstacles in this new environment. Another interesting experiment would be to try the same tests that we are applying, except on different levels to see how mutation affects with different obstacles. Although level 1-1 is supposed to serve as a tutorial for what is to come in Mario bro's perhaps a different level could provide a better environment to learn in. we suggest repeating the learning process thought multiple levels of Super Mario Bro's with the hypothesis that some would constitute a better environment for learning using neuroevolution. Some other aspects that would be interesting to expand on would be to modify the generation size, population size. Specifically Changing the population size at different points in the evolution process would be interesting to test. Using NEAT, The diversity of a generation would be preserved however having a large or small population could encourage learning in specific scenario's.

10 Contributions by Group Members

- Colin contributed writing on multiple sections including introduction, literary review, results, future work, analysis, our implementation, and conclusion.
- Colin revised the document.
- Victor contributed writing on multiple sections including introduction, literary review, conclusion, neuro-genetic evolution basic algorithm/implementation, our implementation results, analysis, and conclusion.
- Victor created graphs and applied them to the document.

- Thomas contributed writing on multiple sections including introduction, literary review, future work, fitness function, and conclusion.

- Everyone contributed to running tests, analyzing results, and gathering data.

References

[1] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[2] M. Forouzanfar, H. R. Dajani, V. Z. Groza, M. Bolic, and S. Rajan. Comparison of feed-forward neural network training algorithms for oscillometric blood pressure estimation. In *4th International Workshop on Soft Computing Applications*, pages 119–123, July 2010.

[3] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.

[4] S. Haykin. *Neural networks: a comprehensive foundation.* Prentice Hall PTR, 1994.

[5] N. C. Hou, N. S. Hong, C. K. On, and J. Teo. Infinite mario bross ai using genetic algorithm. In *2011 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (STUDENT)*, pages 85–89, 2011.

[6] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, et al. Extreme learning machine: a new learning scheme of feedforward neural networks. *Neural networks*, 2:985–990, 2004.

[7] A. H. Kramer and A. Sangiovanni-Vincentelli. Efficient parallel learning algorithms for neural networks. In *Advances in neural information processing systems*, pages 40–48, 1989.

[8] A. M. Mora, J. Merelo, P. García-Sánchez, P. A. Castillo, M. Rodríguez-Domingo, and R. Hidalgo-Bermúdez. Creating autonomous agents for playing super mario bros game by means of evolutionary finite state machines. *Evolutionary Intelligence*, 6(4):205–218, 2014.

[9] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 4(2):93–104, 2013.

[10] S. Pandian. An ai controller for infinite mario bros using evolution strategy. In *2013 International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 721–724. IEEE, 2013.

[11] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Proceedings of the IEEE international conference on neural networks*, volume 1993, pages 586–591. San Francisco, 1993.

[12] E. R. Speed. Evolving a mario agent using cuckoo search and softmax heuristics. In *2010 2nd International IEEE Consumer Electronics Society's Games Innovations Conference*, pages 1–7. IEEE, 2010.

[13] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE transactions on evolutionary computation*, 9(6):653–668, 2005.

[14] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[15] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[16] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber. Super mario evolution. In *2009 ieee symposium on computational intelligence and games*, pages 156–161. IEEE, 2009.

[17] J.-J. Tsay, C.-C. Chen, and J.-J. Hsu. Evolving intelligent mario controller by reinforcement learning. In *2011 International Conference on Technologies and Applications of Artificial Intelligence*, pages 266–272. IEEE, 2011.

11 Apendix

Table 1: Bias Mutation Rate and Generation

Bias Mutation Rate	Generations
0.0254	405
0.0509	412
0.1018	345
0.2036	373

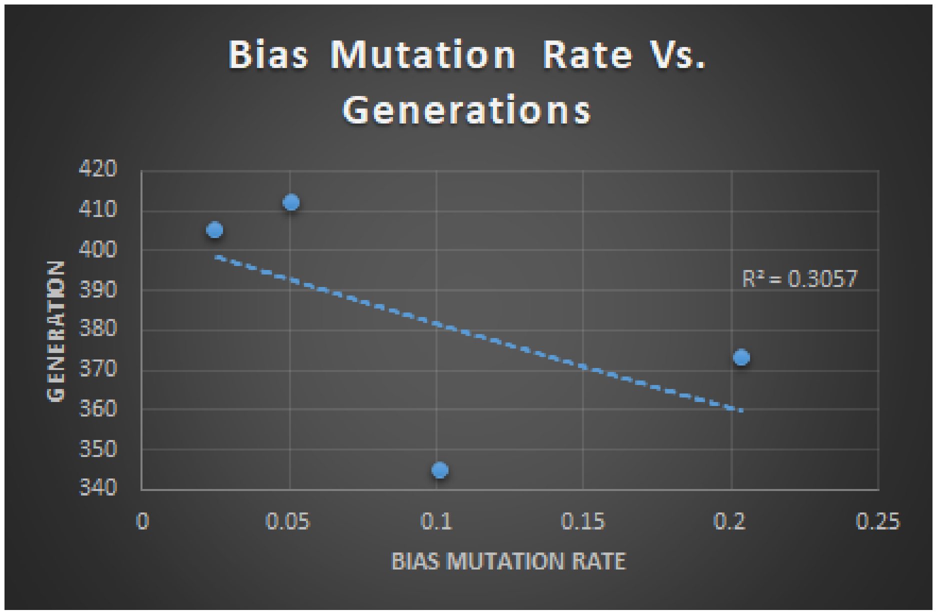


Figure 2: Graph showing the Bias Mutation Rate against the number of generations to reach the goal.

Table 2: Weight Mutation Rate and Generation

Weight Mutation Rate	Generations
0.46	412
0.92	402
1.24	399
1.84	395

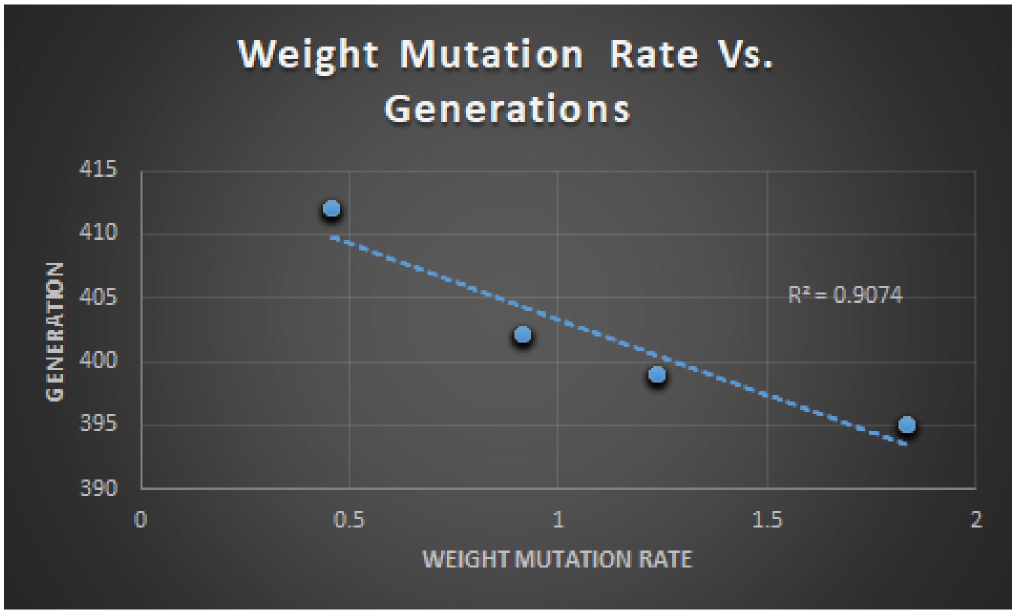
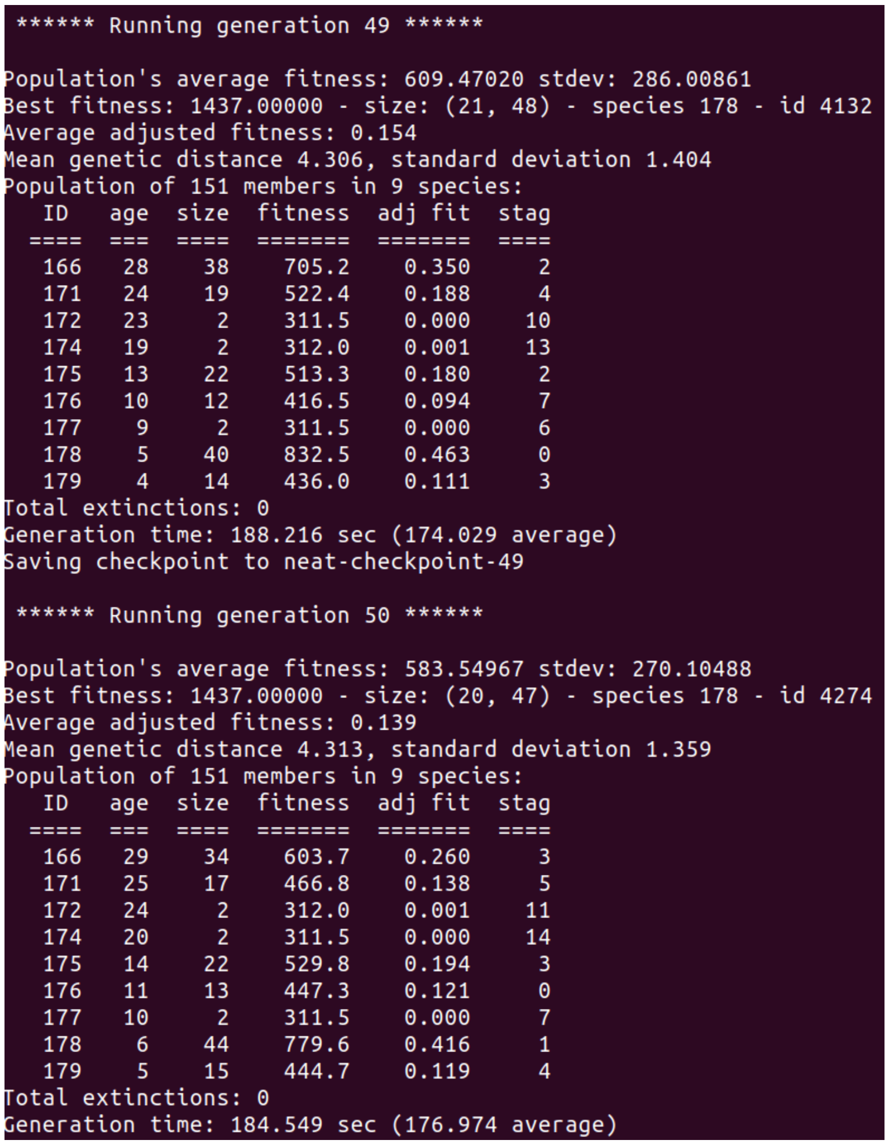
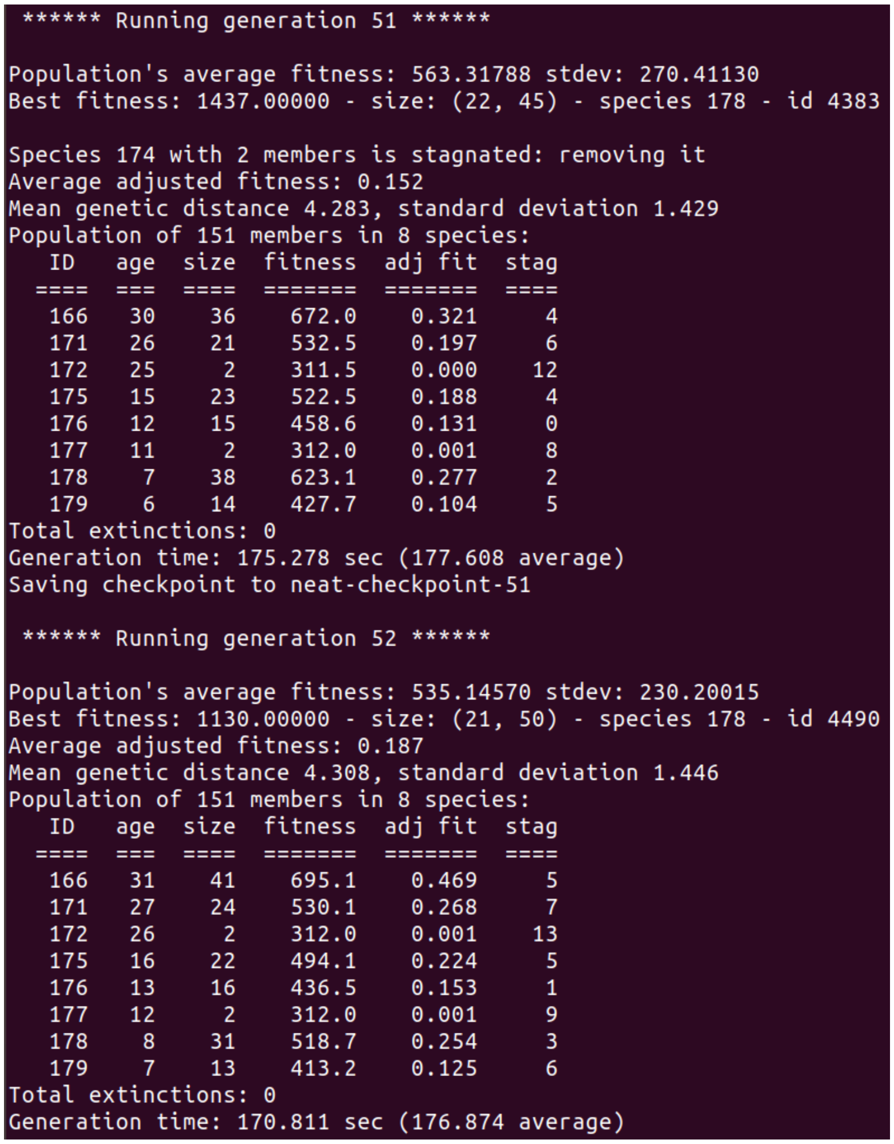


Figure 3: Graph showing the Weight Mutation Rate against the number of generations to reach the goal.



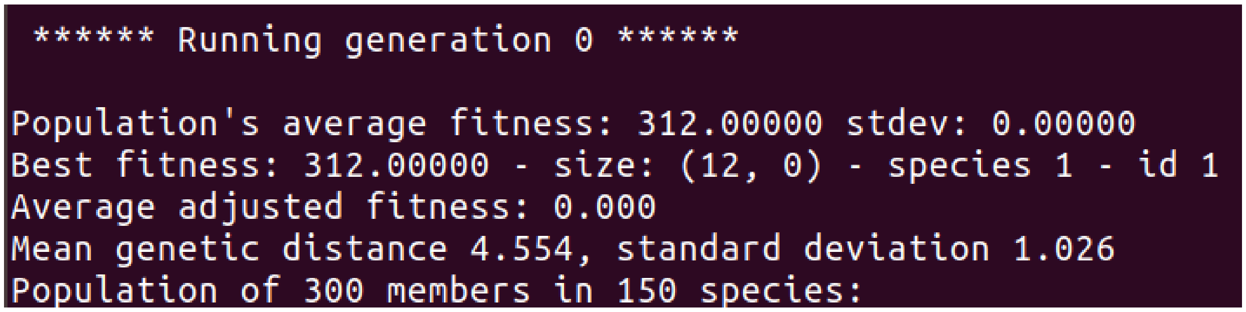
```
 ****** Running generation 49 ******

Population's average fitness: 609.47020 stdev: 286.00861
Best fitness: 1437.00000 - size: (21, 48) - species 178 - id 4132
Average adjusted fitness: 0.154
Mean genetic distance 4.306, standard deviation 1.404
Population of 151 members in 9 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
   166   28    38    705.2    0.350     2
   171   24    19    522.4    0.188     4
   172   23     2    311.5    0.000    10
   174   19     2    312.0    0.001    13
   175   13    22    513.3    0.180     2
   176   10    12    416.5    0.094     7
   177    9     2    311.5    0.000     6
   178    5    40    832.5    0.463     0
   179    4    14    436.0    0.111     3
Total extinctions: 0
Generation time: 188.216 sec (174.029 average)
Saving checkpoint to neat-checkpoint-49

 ****** Running generation 50 ******

Population's average fitness: 583.54967 stdev: 270.10488
Best fitness: 1437.00000 - size: (20, 47) - species 178 - id 4274
Average adjusted fitness: 0.139
Mean genetic distance 4.313, standard deviation 1.359
Population of 151 members in 9 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
   166   29    34    603.7    0.260     3
   171   25    17    466.8    0.138     5
   172   24     2    312.0    0.001    11
   174   20     2    311.5    0.000    14
   175   14    22    529.8    0.194     3
   176   11    13    447.3    0.121     0
   177   10     2    311.5    0.000     7
   178    6    44    779.6    0.416     1
   179    5    15    444.7    0.119     4
Total extinctions: 0
Generation time: 184.549 sec (176.974 average)
```

Figure 4: The 49th generation was performing very well with their fitness raising for many generations prior to it. Then starting with the 50th, it went through an impressive decline



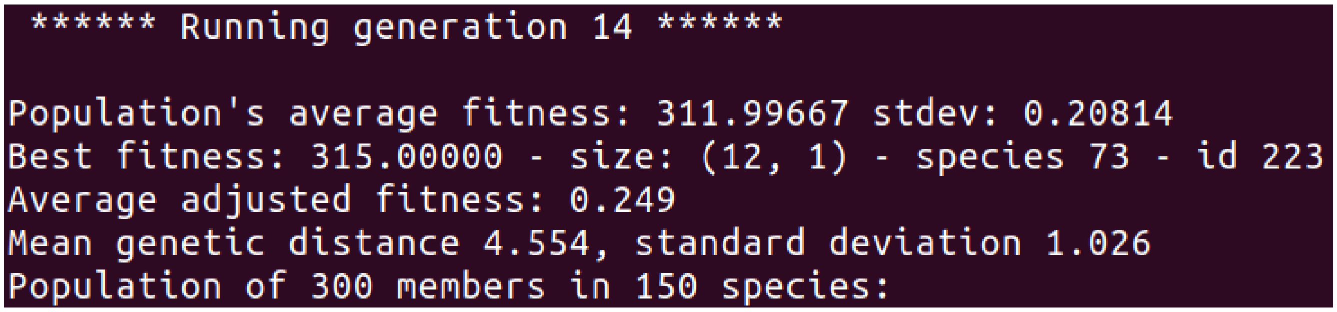
```
 ****** Running generation 51 ******

Population's average fitness: 563.31788 stdev: 270.41130
Best fitness: 1437.00000 - size: (22, 45) - species 178 - id 4383

Species 174 with 2 members is stagnated: removing it
Average adjusted fitness: 0.152
Mean genetic distance 4.283, standard deviation 1.429
Population of 151 members in 8 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
   166   30    36    672.0    0.321     4
   171   26    21    532.5    0.197     6
   172   25     2    311.5    0.000    12
   175   15    23    522.5    0.188     4
   176   12    15    458.6    0.131     0
   177   11     2    312.0    0.001     8
   178    7    38    623.1    0.277     2
   179    6    14    427.7    0.104     5
Total extinctions: 0
Generation time: 175.278 sec (177.608 average)
Saving checkpoint to neat-checkpoint-51

 ****** Running generation 52 ******

Population's average fitness: 535.14570 stdev: 230.20015
Best fitness: 1130.00000 - size: (21, 50) - species 178 - id 4490
Average adjusted fitness: 0.187
Mean genetic distance 4.308, standard deviation 1.446
Population of 151 members in 8 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
   166   31    41    695.1    0.469     5
   171   27    24    530.1    0.268     7
   172   26     2    312.0    0.001    13
   175   16    22    494.1    0.224     5
   176   13    16    436.5    0.153     1
   177   12     2    312.0    0.001     9
   178    8    31    518.7    0.254     3
   179    7    13    413.2    0.125     6
Total extinctions: 0
Generation time: 170.811 sec (176.874 average)
```

Figure 5: Generations 49-52 saw steady decreases in the average fitness of the generation. Surprising after so many generations before it had steadily increasing fitness levels instead



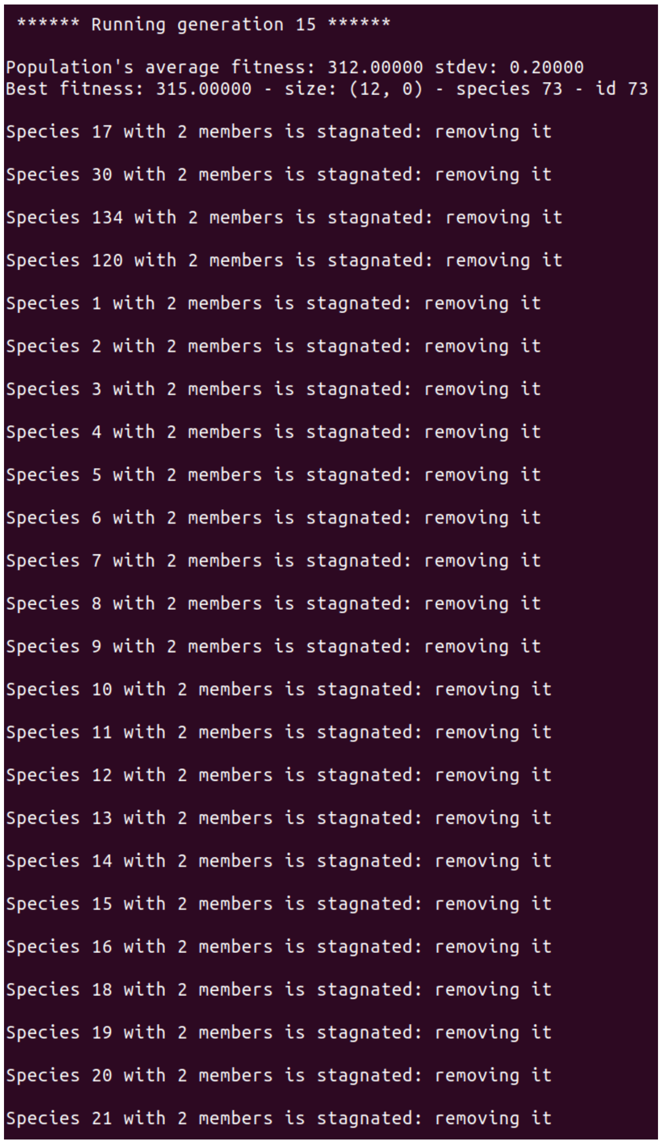
```
 ****** Running generation 0 ******

Population's average fitness: 312.00000 stdev: 0.00000
Best fitness: 312.00000 - size: (12, 0) - species 1 - id 1
Average adjusted fitness: 0.000
Mean genetic distance 4.554, standard deviation 1.026
Population of 300 members in 150 species:
```

Figure 6: Generation ran into the first enemy every single run, who was at location 312



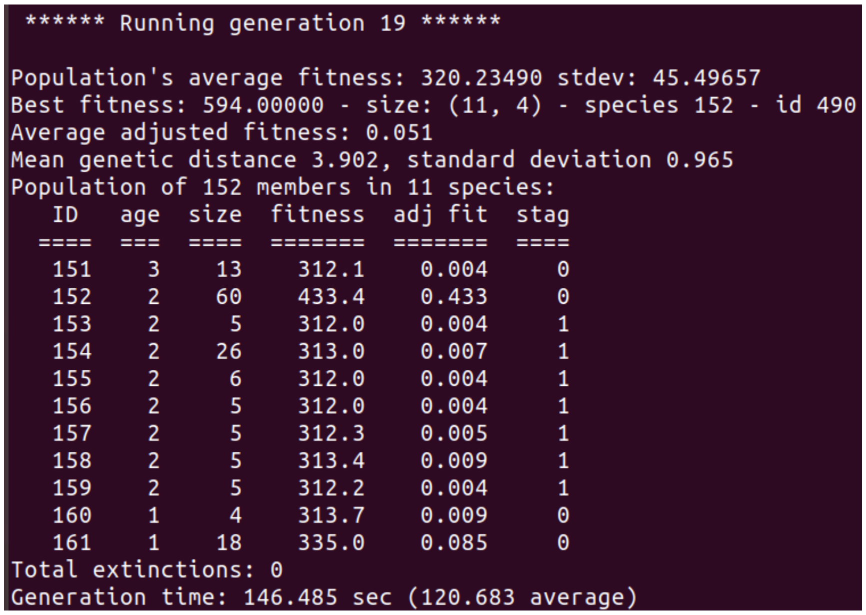
```
 ****** Running generation 14 ******

Population's average fitness: 311.99667 stdev: 0.20814
Best fitness: 315.00000 - size: (12, 1) - species 73 - id 223
Average adjusted fitness: 0.249
Mean genetic distance 4.554, standard deviation 1.026
Population of 300 members in 150 species:
```

Figure 7: All generations 0-14 also ran into the first enemy every single run, showing that no noticeable learning or mutations occurred in those first 15 generations



```
 ****** Running generation 15 ******

Population's average fitness: 312.00000 stdev: 0.20000
Best fitness: 315.00000 - size: (12, 0) - species 73 - id 73

Species 17 with 2 members is stagnated: removing it

Species 30 with 2 members is stagnated: removing it

Species 134 with 2 members is stagnated: removing it

Species 120 with 2 members is stagnated: removing it

Species 1 with 2 members is stagnated: removing it

Species 2 with 2 members is stagnated: removing it

Species 3 with 2 members is stagnated: removing it

Species 4 with 2 members is stagnated: removing it

Species 5 with 2 members is stagnated: removing it

Species 6 with 2 members is stagnated: removing it

Species 7 with 2 members is stagnated: removing it

Species 8 with 2 members is stagnated: removing it

Species 9 with 2 members is stagnated: removing it

Species 10 with 2 members is stagnated: removing it

Species 11 with 2 members is stagnated: removing it

Species 12 with 2 members is stagnated: removing it

Species 13 with 2 members is stagnated: removing it

Species 14 with 2 members is stagnated: removing it

Species 15 with 2 members is stagnated: removing it

Species 16 with 2 members is stagnated: removing it

Species 18 with 2 members is stagnated: removing it

Species 19 with 2 members is stagnated: removing it

Species 20 with 2 members is stagnated: removing it

Species 21 with 2 members is stagnated: removing it
```

Figure 8: Generation 15 noticed that generations 0-14 had made no advancement, and purged almost every species making room for additional mutations.



```
 ****** Running generation 19 ******

Population's average fitness: 320.23490 stdev: 45.49657
Best fitness: 594.00000 - size: (11, 4) - species 152 - id 490
Average adjusted fitness: 0.051
Mean genetic distance 3.902, standard deviation 0.965
Population of 152 members in 11 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
   151    3    13    312.1    0.004     0
   152    2    60    433.4    0.433     0
   153    2     5    312.0    0.004     1
   154    2    26    313.0    0.007     1
   155    2     6    312.0    0.004     1
   156    2     5    312.0    0.004     1
   157    2     5    312.3    0.005     1
   158    2     5    313.4    0.009     1
   159    2     5    312.2    0.004     1
   160    1     4    313.7    0.009     0
   161    1    18    335.0    0.085     0
Total extinctions: 0
Generation time: 146.485 sec (120.683 average)
```

Figure 9: In generation 19, just a few generations after the purge, species finally learned how to jump and began making it past the first enemy, despite not figuring it out for all of generations 0-14