Mario Evolved

Training Evolutionary Algorithms to Beat Levels of Mario

Jonathan Clark
University of Tennessee
Knoxville, TN 37996
Email: jclar166@vols.utk.edu

Nolan Coffey
University of Tennessee
Knoxville, TN 37996
Email: ncoffey3@vols.utk.edu

Faithful Odoi
University of Tennessee
Knoxville, TN 37996
Email: cvy221@vols.utk.edu

***Abstract*—The study and application of evolutionary algorithms is an ever growing field with still more room to go. One particularly interesting application of evolutionary algorithms is in the world of video games. From the potential use of evolutionary algorithms to train characters to adapt to the player, to the training of algorithms to play and beat games on their own, a considerable body of work already exists investigating the use of evolutionary algorithms in video games. Our work seeks to further the study of such applications by seeing if we can train two different evolutionary algorithm approaches to beat a level of Mario. The first is a Fixed Action Sequence approach, wherein the algorithm evolves an array of actions Mario can take in the game until it beats the level. The second is a Neuroevolutionary approach using NEAT, wherein the topology of a neural network is evolved to beat the level. We compared the number of times each approach was able to beat the levels, as well as each approach's average completion percentage of a level over each generation. We found that the Fixed Action Sequence completed more levels than the Neuroevolutionary approach, and that it completed these levels more times. We also found that the Fixed Action Sequence approach achieved a higher average completion percentage in fewer generations.**

***Index Terms*—Evolutionary algorithms, Neural networks, NEAT, Neuroevolution, MarioAI Framework**

I. Introduction

The goal of our project was to see if an evolutionary algorithm could be trained to complete a level of Mario. Specifically, we were interested in comparing the performance of different evolutionary approaches and seeing if we could train a generalizable evolutionary algorithm - one that could be trained on a smaller set of levels and then still perform well on levels outside of its training set. We chose this topic because we were interested in the applications of evolutionary algorithms outside those that were discussed in class, specifically we were interested to see how an evolutionary algorithm could be applied to video games. We chose evolutionary algorithms in particular because it seemed to have the greatest potential for use in a video game application, and also because we all found it to be one of the more interesting topics covered in the course.

II. Related Work

While our project might seem like a niche application at first glance, we are far from the first to implement evolutionary algorithms in game environments. Our project differs from others by specifically training the model to beat levels of Mario, while also comparing the results of two different approaches to train evolutionary algorithms. Each of the works we will discuss falls into one of the following three categories:

- Fundamental Evolutionary Algorithm Methods
- Neuroevolution in Platforming Games
- Evolutionary Approaches in Other Game Types

The Fundamental Evolutionary Algorithm Methods category focuses on papers that specifically highlight the architecture of evolutionary algorithms and provide deeper insight into the terminology and structure of NEAT. The Neuroevolution in Platforming Games section examines how evolutionary algorithms have been implemented in other platforming games, whether to have a model beat levels or generate levels. This section demonstrates the variety of ways evolutionary algorithms can be applied to platformers. The final section, Evolutionary Approaches in Other Game Types, explores how other projects have implemented evolutionary algorithms in non-platforming games, showing the broader applicability of these methods.

A. Fundamental Evolutionary Algorithm Methods

One of the foundational pillars of our project is NeuroEvolution of Augmenting Topologies (NEAT), which is expounded on in greater detail by Stanley and Miikkulainen [1]. NEAT improves upon traditional neuroevolution by starting with minimal networks and gradually making them more complex through mutation, allowing both topology and weights to evolve. This methodology enables the discovery of optimal network architectures in parallel with weights. This paper covers the specific mechanisms of NEAT and will provide further insights to specific questions about NEAT architecture that are not covered in our report.

Papavasileiou et al. [2] provides a comprehensive review of NEAT's impact by analyzing NEAT's successors and categorizing them into structural improvements, hybrid approaches, and application-specific adaptations. Their work helped contextualize our implementation within the broader scope of neuroevolution research, showing that our NEAT usage is aligned with common applications in real-time, agent-based decision making. It is also provides potential next steps, with exploring more hybrid approaches to potentially decrease the learning time our current implementation of NEAT.

"An introduction to genetic algorithms and evolution strategies" is a survey by Dianati et al. [3], which breaks down the

mechanisms behind genetic algorithms and evolution strategies. Their explanation of genotype representation, selection pressure, and convergence issues shows the logic behind how we structured our Fixed Action Sequence method and tuned parameters, like mutation and crossover rates. Additionally, their report covered strategies to avoid early convergence in genome sequences which is a much more pressing issue when dealing with less complex evolutionary algorithms like the Fixed Action Sequence method.

Lastly, Jangid et al. [4] provides a critical review of evolutionary algorithms and their future prospects. They highlighted performance bottlenecks such as premature convergence and stagnation—issues we encountered in early iterations of our NEAT experiments. Their work underscored the importance of balancing exploration and exploitation during evolutionary training. This insight led us to adjust our mutation rates and selection mechanisms to promote greater diversity in early generations. Additionally, we monitored fitness variance across the population to detect and address convergence stalls more effectively.

B. Neuroevolution in Platforming Games

A growing number of experiments are applying neuroevolutionary algorithms to platforming games like Super Mario Bros. Ortega et al. [5] presented one of the earlier examples of learning human-like playing styles in Mario using neuroevolution, including behavior cloning and NEAT-based agents. Their work influenced our input feature selection for NEAT, particularly in capturing local level geometry and enemy positions. Their approach also greatly differs from ours in focus. As mentioned earlier they focus on making the model play in a way that is as human as possible and as such train their model with human gameplay. We focus purely on the model finishing a level, therefore our model regularly does tasks like frame and pixel perfect jumps that a human could not replicate. On the other hand, due to their approach their more human like model is able to backtrack and solve more complicated level design, something our version struggles to do.

In a more recent work, Paul and Selvan [6] examined hybrid approaches that combine NEAT with reinforcement learning to play 2D platformers indefinitely. Their findings reinforced our hypothesis that NEAT-based agents, while slower to converge than fixed strategies, are more robust and adaptive to changing environments. Their work also supported our idea that implementing a hybrid approach to NEAT training—such as first using a Fixed Action Sequence and then refining the results with NEAT, or employing any other method that integrates NEAT—is more efficient for developing a model than training using NEAT alone.

The concept of using evolutionary methods to generate or complete levels is also present in the work by Ortega et al. [5] and the level-learning approach by Cotta et al. [7], which utilized Genetic Algorithms to evolve levels tailored to agent behavior. Although our work focuses on agent performance rather than level design, these papers show the flexibility of evolutionary methods in interacting with game environments.

C. Evolutionary Approaches in Other Game Types

Outside of platformers, evolutionary algorithms have been applied to games like Flappy Bird, as seen in Cordeiro et al. [8]. Their minimal training approach using NEAT demonstrated that even very simple environments benefit from neuroevolution, especially in terms of adaptability to variable timing challenges—a principle we observed in our own work with levels requiring tight platforming precision.

Telikani et al. [9] provides a broader overview in their survey of evolutionary machine learning. They highlight the trade-offs between genetic programming, evolution strategies, and hybrid models. Their work was helpful in contextualizing our comparison of NEAT with simpler, rule-based methods like Fixed Action Sequences, as it reinforced the idea that more complex evolutionary methods can outperform simpler ones given enough time and well-structured environments. They also emphasized that hybrid models often strike an effective balance between exploration and exploitation, which inspired our interest in combining Fixed Action Sequences with NEAT for improved training efficiency. Furthermore, their analysis of scalability challenges in evolutionary algorithms helped guide our decisions regarding population size and generational depth. By incorporating these lessons, we were better equipped to design experiments that maintained performance while avoiding unnecessary overhead.

Additional insights came from the work of Ambuehl [10], who investigated genetic algorithm optimization techniques for video game immersion. Their paper emphasized how genetic approaches could create more lifelike AI agents compared to traditional decision-tree models, aligning with the broader goals of creating adaptive, human-like behavior. Our model ultimately prioritized level completion and optimization over behavioral realism. We focused on ensuring that the agent could consistently reach the end of each level rather than mimicking human-like hesitation or exploratory behavior. Nevertheless, Ambuehl's emphasis on immersion helped us critically evaluate instances where our model's behavior, though effective, appeared overly mechanical or repetitive. This informed our reflection on the trade-off between efficiency and naturalness in AI behavior within game environments.

III. Methods and Approach

For this project we used the *MarioAI Framework* from Ahmed Khalifa on GitHub [11]. The framework contained 15 premade levels and game information such as level completion percentage, number of coins collected, number of mushrooms collected, and more. We used this game information to construct our fitness function, and ran our algorithms on the 15 premade levels. We tested each level 20 times, performing evolution over 200 generations. Each generation consisted of 300 individuals, leading to a total of 1.2 million tested individuals for each level.

We chose a population size of 300 to ensure diversity among the individuals in the population. We found that a population greater than 300 did not have a noticeable impact on the agents' ability to complete the levels, and a lower population size unsurprisingly limited the agents' ability to learn and complete the levels. We chose to evolve the agents over 200 generations to ensure the agents had enough chances to iterate and learn the levels. Finally, we chose to run each level 20 times to ensure we were getting generalized results and that the agents were able to consistently beat the levels.

The framework also visualized the results of our algorithms, which allowed us to see what our algorithms were actually doing in the level and make adjustments to improve performance. We also used this visualization to create videos of the agents running through the levels.

A. Fitness Function

Input: Completion Percentage C, Time Left T, Win Status W, Kills K, Mushrooms M, Coins P, Hurts H

$completionScore \leftarrow 4000 \times C$
$timeScore \leftarrow \frac{T}{1000} \times 10 \times W$
$killScore \leftarrow 1.5 \times K$
$mushroomScore \leftarrow 1.5 \times M$
$coinScore \leftarrow 1.25 \times P$
$hurtPenalty \leftarrow 3 \times H$

$fitnessScore \leftarrow completionScore + timeScore + killScore + mushroomScore + coinScore - hurtPenalty$

Fig. 1: Finalized Fitness Function

The criteria evaluated for the fitness function included: Completion Percentage which evaluated how far the agent traversed in the level. This metric was weighted the highest in order to incentivize the agent to complete the level as its first priority. Secondary to this was time remaining left in the level in seconds. The agent was only positively rewarded for time remaining if it successfully completed the level. This was done to avoid agents terminating runs early in order to maximize the time remaining score. The remaining metrics used for calculating the total fitness score include a score for killing enemies, collecting mushrooms, and collecting coins. These components constitute a simplified version of the score calculation used in the original Super Mario Bros. This fitness function was used instead of the original scoring system for both ease of implementation with the simulator as well as incentivizing behavior of a "good player". We decided upon this fitness function after several iterations closer in representation to the original game's scoring system, however this led to agent behavior that wasted time in favor of collecting coins or killing enemies. We further introduced a penalty for agents that are hurt by enemies in the level. The goal of this was to discourage agents from being damaged by enemies and thus potentially extract patterns specific to each enemy type and placement on how to avoid this.

B. Fixed Action Sequence

For our first approach, we used a Fixed Action Sequence. In this approach, our population of 300 individuals consists of arrays of size 1000 where each element of the array is a random action or set of actions that Mario can take in the level - move right, move left, jump, duck, and sprint. Each index represents a frame in the simulator, and at each frame the actions in the index are performed. These individuals then run through the level and see how far they can get, with the end goal of completing the level. After all individuals in the generation have finished, we populate a new generation of individuals using tournament selection. When a new generation is populated, we also perform mutation and uniform crossover.

Frame 1	Frame 2	Frame 3	Frame 4	...
Jump + Right	Right + Sprint	Right	Jump	...

Fig. 2: Fixed Action Sequence Example

We decided to use uniform crossover for our algorithms with a 30% chance of crossover. This crossover approach as well as the percentage chance were determined based on the parameter values of Lab 2 that resulted in the best fitness. We used the same idea for our original mutation chance, set at 10%. We quickly noticed, however, that the agent relied heavily on mutation to reach the end of certain levels. To account for this, we increased the mutation chance to 30% to match crossover chance. While this did improve the agent's ability to complete the mutation reliant levels, it also decreased the agent's consistency in completing levels that it originally had no trouble with. We then reduced the mutation chance to 20%. This balanced approach allowed for the creative mutation needed to complete mutation dependent levels while avoiding destructive mutation that hurt the agent's ability to consistently beat other levels.

We decided to use tournament selection to populate new generations because of its ability to maintain the diversity of a population and because it is generally a fair selection method. It also allowed us to quickly and easily alter our level of selection pressure simply by changing the tournament size. We once again relied on the results of the Lab 2 parameter values when determining our tournament size. We briefly debated whether a size of 4 or 5 would be better, given how similar their performance was in lab 2, but ultimately settled on a size of 4. This was largely a result of the observed larger standard deviation for performance with a higher selection pressure, meaning that a size of 4 would likely produce more consistent results than a size of 5.

We also attempted to create a "jump aware" version of our mutation and crossover functions. We noticed that the most common result of harmful mutations was the removal of jump sequences, preventing the agent from clearing certain gaps or platforming challenges. This version of the function would skip over any sections that started with a jump action. We would detect if the index contained a jump action, and then would maintain the sequence 5 to 7 indexes from where the

jump was detected. The idea was that jumping was almost always beneficial to the agent's ability to complete a level, and so maintaining as many jump sections as possible would result in quicker and more consistent level completion. In reality, this version of the crossover and mutation functions impeded the agent's ability to complete levels. Indeed, the agent was incapable of completing any levels. Maintaining jump sequences in this manner all but prevented the agent from attempting crossover or mutation. Looking at the visualization of the agent's actions, we saw that it would jump repeatedly and would not make substantial progress towards completing the level. With all this in mind, we scrapped the jump aware versions of our functions.

C. Neuroevolutionary Agent - NEAT

For our second approach, we utilize the Encog Machine Learning Framework's implementation of NEAT created by Jeff Heaton [12]. This implementation was used due to its ease of incorporation with our simulator MarioAI which was also written in Java.

In order to maintain consistency between both approaches, a population of 300 individuals was utilized running for a total of 200 generations. The same fitness function was utilized for evaluation of the population as well. However, in contrast to our Fixed Action Sequence Approach, NEAT agents were not "blind" to their environment. NEAT evolves Neural Network agents over time to maximize the fitness score, and thus we pass input features into these networks so that they can potentially learn and extract patterns from their environment. We selected the following features as input to our NEAT agent networks:

- A flattened grid representation of the space around Mario that indicates nearby enemies, coins, and blocks.
- Mario's current x and y velocity.
- If Mario is touching the ground and whether Mario may jump.

These features were selected so that the agents could learn to extract patterns about the environment including enemy type, enemy location, gap and block placement in order to help time jumps, and so that the agent might learn to harness speed and momentum.

Each frame these features were extracted from the MarioAI simulator and fed as input into the agent's neural network. The agent then processes these features and determines which actions to take. Each output neuron is passed through a Sigmoid Activation Function used for binary classification with a threshold of 0.5 resulting in 5 possible actions (Left, Right, Jump, Duck, Sprint) being tried alone or in any combination.

IV. Results

For the figures in this section, we sampled the best individual from each generation. After 20 runs of each level, we had a sample of 4000 total recorded individuals.

A. Fixed Action Sequence

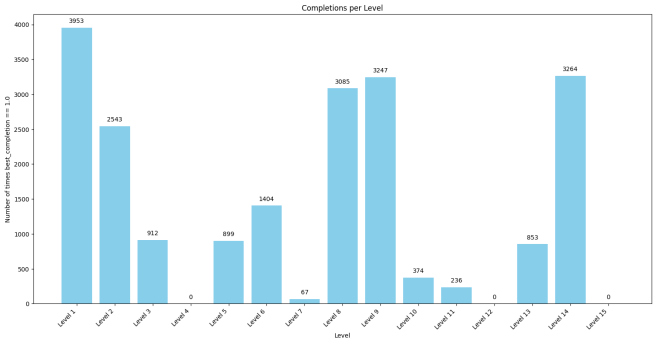

Fig. 3: Number of Completions per level vs Level

The Fixed Action Sequence approach performed well overall. The agent was able to complete 12 of the 15 levels that it was trained and tested on. Of those 12 levels, 4 featured over 3000 individuals that achieved completion. Six of those levels featured fewer than 1000 individuals that achieved completion. These lower performing levels often featured a more varied construction that required more complex actions to complete.

For the first level the agent failed to beat, level 4, the final obstacle right before the end of the level is a large wall that prevents forward movement. The intended course of action is to backtrack and platform over the wall. We tried to implement backtracking as a set of actions the agent could take, moving left for 5 to 10 frames if forward moment had paused. Implementing this did not help us complete the level. The agent would either backtrack to a point where it could theoretically complete the platforming challenge but fail to properly jump over the wall, or would backtrack only a few steps before running forwards into the wall again. Additionally, the agent was now incapable of completing other levels - reaching only ˜ 20% completion on most levels. Much like our jump aware functions, we had to remove this implementation and settle for the agent failing to complete level 4.

For the other two levels the agent failed to complete, the issue was the agent's inability to safely perform one block jumps. In level 12, the agent fails to perform the jump because the closest landing point is a pipe with an emerging piranha plant. The agent hits the piranha plant and dies. In level 15, the agent sprints through a completely flat section before falling in a one block gap at the end of the section. The agent seems to struggle to complete both levels in a timely manner, having only 12 seconds left at 75% at level 12 and only 2 seconds left at 50% completion in level 15. Although we cannot definitively say what caused both of these errors, it seems that the agent may have evolved to prioritize getting as far as possible, even at the expense of completing the level, because it "realized" it was running out of time.

Level	Best Completion
1	1.0000
2	1.0000
3	1.0000
4	0.9561
5	1.0000
6	1.0000
7	1.0000
8	1.0000
9	1.0000
10	1.0000
11	1.0000
12	0.7539
13	1.0000
14	1.0000
15	0.5512

Fig. 4: Best Fitness and Best Completion per Level

Figure 4 shows that our agent still achieved good progress even on the levels it did not complete. The agent all but completes level 4, and is over three quarters of the way to completing level 12. Our agent performs the worst on level 15, but still completes over half the level.

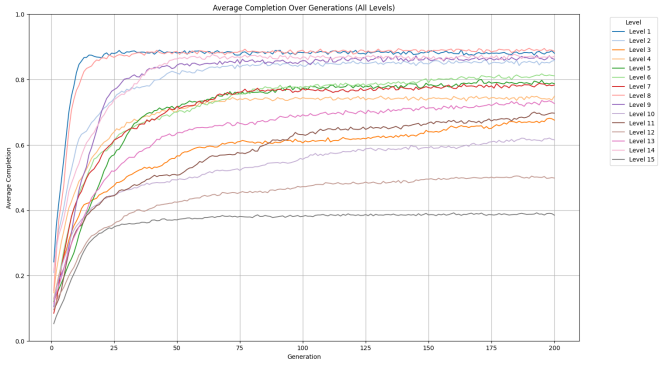

Fig. 5: Average Completion Percentage per level vs Generation

Our Fixed Action Sequence approach achieved high average completion percentage on all but levels 12 and 15. Even in our under performing levels, we see an immediate jump in average completion percentage before plateauing fairly quickly. This demonstrates that a Fixed Action Sequence approach performs well even with fewer generations, likely as a result of how varied the initial population is. The approach can almost immediately try multiple different methods of completing the level and evolve based on those that perform the best.

For the levels with fewer total completions, as seen in Figure 3, we see that their trend lines have a less severe spike at the beginning and that they take longer to reach convergence. This suggests that the lower number of completions is a result of the agents taking a longer time to reach a good solution. Levels like level 3 and level 11 never converge, demonstrating that the speed at which the agent learns directly correlates with how many times it is able to complete the level. These levels are also more complex and difficult, with harder and more varied platforming sections, explaining why the agent takes more time to converge and complete them.

B. Neuroevolutionary Agent - NEAT

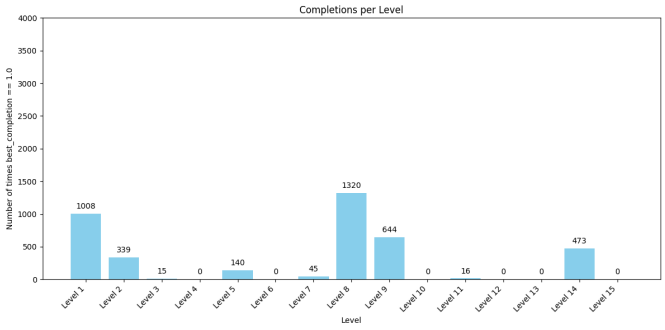

Fig. 6: Number of Completions per level vs Level

The Neuroevolutionary approach utilizing NEAT had mixed results, successfully completing 9 of the 15 levels it was trained and tested on. However, this approach had significantly fewer completions per level than the Fixed Action Sequence approach. In contrast to the Fixed Action Sequence there were no levels on which 3000 individuals achieved completion, instead there were only two in which over 1000 individuals achieved completion.

Level	Best Completion
1	1.0000
2	1.0000
3	1.0000
4	0.9468
5	1.0000
6	0.9139
7	1.0000
8	1.0000
9	1.0000
10	0.9165
11	1.0000
12	0.8051
13	0.8768
14	1.0000
15	0.6608

Fig. 7: Best Completion per Level

When examining the best recorded completion rate for all levels NEAT achieved similar performance to the Fixed Action Sequence on most levels. While the Fixed Action Sequence Agent completed 3 more levels than the NEAT agent did the difference in completion percentage was not a large gap, demonstrating that NEAT did not lag far behind on its best trained agent.

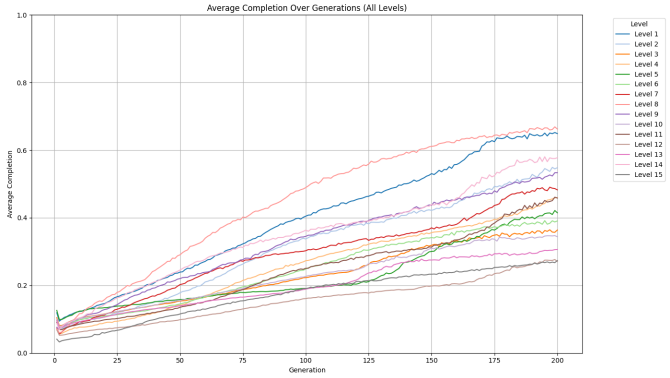

Fig. 8: Average Completion Percentage per level vs Generation

Figure 8 provides valuable insight for understanding the learning of the NEAT agent, especially when juxtaposed with the corresponding Fixed Action Sequence graph. Figure 5 demonstrates that agents exhibit rapid initial learning and then plateau quickly. In contrast Figure 8 demonstrates that NEAT agents exhibit a slower but steady learning rate continuing up until the last generation is reached. This growth does not appear to plateau and thus training for more generations might yield better agent performance. These trend lines suggest that NEAT Agents take significantly more generations to learn each level.

V. Conclusion

We saw that both approaches were able to finish multiple levels, with the Fixed Action Sequence performing slightly better than the Neuroevolutionary approach. Both approaches perform better on less complex levels like 1, 8, and 14 where major sections of the level are mostly flat with few platforming challenges. While the Fixed Action Sequence approach shows an immediate jump in completion percentage, the Neuroevolutionary approach demonstrates a more gradual growth in completion percentage. Both trend lines show higher average completion percentage on levels with higher total completions, demonstrating that the agents' ability to achieve convergence early results in greater performance overall.

We believe the disparity in performance between the two approaches is a result of the number of tested generations. The immediate spike and then plateau of average completion percentage in the Fixed Action Sequence approach suggests that the agent is able to quickly learn the levels and then complete them in a short amount of time. The more gradual slope of average completion percentage for the Neuroevolutionary approach suggests that the agent takes a longer time to learn the levels before completing them. Furthermore, the trend lines for each level are still increasing when we reach our final generation, showing that the agent had not finished learning the levels by the time we cut them off. This idea is further supported by more testing we performed after our initial comparison study was finished.

Additional testing was conducted utilizing varying generation counts in order to evaluate whether increasing the training time of NEAT would yield overall performance similar to the Fixed Action Sequence approach. While requiring further validation we found that increasing generation count up to 500 led to performance improvements and additional level completions in some cases. However, even with higher generation counts the NEAT agents were unable to complete some levels that the Fixed Action Sequence agent were (i.e. levels 6 and 12). This suggests that further investigation for improving the performance of our Neuroevolutionary agent approach is warranted.

We hypothesize that while NEAT may require more generations to train for comparable performance it is potentially more generalizable. The Fixed Action Sequence is a static array of actions that is initially randomized and evolves over time in accordance with feedback from the fitness function, however it receives no other input from the environment. Thus, Fixed Action Sequence agents could be said to act "blindly" and a trained individual agent will always perform the same sequence of actions regardless of which level it is evaluated on. These factors would likely severely hinder the potential for Fixed Action Sequence agents to generalize to other levels. In contrast, the implemented Neuroevolutionary agent receives constant information from its environment. Furthermore, NEAT results in a final trained neural network that will process environmental input and act upon it. This architecture seemingly lends itself better to generalizability and the potential for a single trained network to complete several levels. While not the main focus of our research this was partially validated by further testing of NEAT agents. During this training in order to emphasize generalizability each individual in each generation was evaluated on the first 10 of the 15 levels. The resulting fitness scores were averaged for a single final fitness score for each individual. The goal of this was to increase generalizability by increasing an overall average fitness score on all levels. Initial testing found this approach to be promising, but with major challenges for viability. Testing this approach with 1500 generations and no changes to other parameters already established led to the completion of 2 of the 15 levels (levels 1 and 8). Upping the generation count to 5000 led to the completion of 3 of the 15 levels (levels 1, 3, and 8). Increasing the generation count to 7000 led to no levels being completed. Further testing is required to better understand the generalizability of NEAT agents, however these preliminary results suggest that NEAT agents possess the capability of generalizing to multiple levels. There is clear room for improvement as the best agent only completed 3 of the 15 levels.

VI. Future Work

Potential future work for this project could investigate optimizing implementations of NEAT for better generalizable performance. These optimizations might be in the form of alterations to the fitness function, generation size, or input feature selection. In our testing we found that multi-level training posed a significant training time bottleneck due to a need for many evaluations and generations of training. Thus,

we could investigate utilizing a novel approach that combines a Fixed Action Sequence and NEAT agent. This would likely take the form of training a Fixed Action Sequence agent for a specified number of generations (i.e. 10) to take advantage of the rapid learning ability of these agents. This resulting sequence could then be passed as an input feature to a NEAT agent to act as a reference for the agent to more quickly learn viable paths. This approach could lead to reductions in training time, and a combined approach could potentially leverage the rapid learning of a Fixed Action Sequence agent and the pattern extracting ability of NEAT in order to solve levels that neither approach could solve alone.

REFERENCES

[1] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[2] E. Papavasileiou, J. Cornelis, and B. Jansen, "A systematic literature review of the successors of NeuroEvolution of Augmenting Topologies (NEAT)," *Evolutionary Computation*, vol. 29, no. 1, pp. 1–38, 2021.

[3] M. Dianati, I. Song, and M. Treiber, "An introduction to genetic algorithms and evolution strategies," University of Cincinnati, 2018. [Online].

[4] S. Jangid, R. Puri, and T. Kumar, "Evolutionary algorithms: A critical review and its future prospects," *International Journal of Trend in Research and Development*, vol. 7, no. 6, pp. 1–5, 2020.

[5] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, "Imitating human playing styles in Super Mario Bros," in *Proc. 9th Int. Conf. on Computational Intelligence and Games (CIG)*, 2013, pp. 1–8.

[6] J. P. Selvan and P. S. Game, "Playing a 2D game indefinitely using NEAT and reinforcement learning," *arXiv preprint arXiv:2207.14140*, 2022. [Online].

[7] A. M. Mora and C. Cotta, "Learning levels of Mario AI using genetic algorithms," in *Applications of Evolutionary Computation*, Springer, 2016, pp. 288–300.

[8] M. Cordeiro, P. Serafim, Y. Nogueira, C. Vidal, and J. Neto, "A minimal training strategy to play Flappy Bird indefinitely with NEAT," in *Proc. Brazilian Symp. Games and Digital Entertainment (SBGames)*, 2019. [Online].

[9] A. Telikani, A. H. Tahmassebi, W. Banzhaf, and A. H. Gandomi, "Evolutionary machine learning: A survey," *ACM Comput. Surv.*, vol. 55, no. 1, pp. 1–38, 2021. DOI: 10.1145/3467477.

[10] Nathan Ambuehl, "Investigating genetic algorithm optimization techniques in video games," East Tennessee State University, 2023. [Online].

[11] A. Khalifa, "Mario AI Framework: 10th Anniversary Edition," GitHub repository, https://github.com/amidos2006/Mario-AI-Framework, accessed May 11, 2025.

[12] J. Heaton, "Encog Machine Learning Framework for Java," GitHub repository, https://github.com/jeffheaton/encog-java-core, accessed May 11, 2025.