Evolving Mario to Maximize Coin Score Using Neat and Novelty

Amy Han and Jeremy Han

Abstract

Genetic algorithms can be used to evolve agents that will complete game tasks in a given game environment. In this paper, we discuss our experimental results using NEAT and Novelty to evolve Mario, from the popular game Super Mario Bros, to maximize his coin score. To conduct our experiments, we developed our own Mario simulator, creating a small world and a big world. Each world has an easy and hard version; the addition of coin boxes is what defines a world to be difficult. Just like in the original SMB, we added hidden rooms that contain many coins. However, finding the hidden room is the most difficult part. We hypothesized that Novelty would outperform NEAT because of its exploratory nature, while NEAT would focus more on getting the coins that are easy to obtain. Our results indicated that without an input that tells Mario what type of coin is nearest to him, Novelty significantly outperforms NEAT. However, with the input, NEAT can perform on par, or even better than Novelty at times. We argue that the reason for his discrepancy is that the addition of the input makes the task less deceptive for NEAT, which closes the performance gap between NEAT and Novelty.

1 Introduction

Video and computer games often use artificial intelligence (AI) code to give life to non-playable characters (NPCs) in the game. For example, enemy NPCs are programmed to attack your in-game character while ally NPCs are programmed to help. In our experiments, we used artificial intelligence for an entirely different purpose: to evolve an agent to excel at a certain task. Inspired by Seth Bling's "MarI/O Machine Learning for Video Games" video [1], we decided to conduct experiments on Mario. In Super Mario Bros. (SMB), a popular two-dimensional platform game, the human player controls a character named Mario and tries to clear each level of the game by exploring and finding the finish flag. Along the way, the player can collect coins, kill enemies, be killed off by enemies, and more. Most levels are predictable so that the player can expect exactly what will happen, such as when and where an enemy will appear or where the coins are. Players often memorize key features in each level to optimize their score.

In Seth Bling's video, Mario evolved to excel at finishing the level. Our experiments focus on evolving Mario to obtain as many coins as possible. To evolve Mario, we used NEAT and Novelty Search. While NEAT alone can be used to solve this coin-collecting task, it is not ideal for solving deceptive tasks - tasks that force the agent to explore areas with lower fitnesses during evolution. Instead, another approach is to use Novelty Search in the hopes that the search for novel behaviors will result in a more promising solution. In this paper, we conducted NEAT and Novelty experiments in a custom Super Mario Bros. simulation environment. Our goal was to find Mario behaviors that could collect many coins. Deception was added to the task by creating hidden coin bonus rooms in coin-sparse areas to observe the strengths and weaknesses between the NEAT and Novelty solutions.

NEAT is a genetic algorithm that evolves neural networks using genetic encodings with historical markings to create a population of individuals that are judged on how they perform based on a user-defined measure of fitness [5]. For NEAT, the goal is to find high fitness scoring individuals (global maxima in the behavior space). Thus, during its search for high scoring individuals, NEAT often falls victim to local maxima and is unable to find better solutions. Novelty Search uses NEAT but focuses on finding novel behaviors, as opposed to finding global maximum, in the behavior space [4]. To accomplish this, Novelty keeps an archive of unique behaviors and compares the most recent behavior with those in the archive. If the new behavior is novel, it gets a higher uniqueness score. This uniqueness score is then fed into the underlying NEAT network so that it can find more novel behaviors.

1.1 Previous Research

Of course, our study is not the first to incorporate evolutionary algorithms in video games. One well-cited paper, co-authored by one of the creators of the NEAT algorithm, Kenneth O. Stanley, used neural networks to evolve NPCs and weapons in a game that they created called NERO [3]. In NERO, the human player acts as a trainer for NPCs equipped with rtNEAT, a real-time version of NEAT. The human player sets up a training scenario so that the NPCs can learn combat tactics and strategy in real time. Then, the NPCs fight against a team that the human player controls using what they learned during the training session. This evolutionary algorithm shows promise because it can add more flavor to NPCs' predictable behavior, making gameplay more interesting. However, since rtNEAT is run in real-time, it is not useful for quickly optimizing the ability to excel in one task.

There are two other studies that are similar to the NERO study in the way that the genetic algorithms used also react to their environment in real time. In the first study, Mario uses reinforcement learning to react to his surroundings [2].While the main goal is for Mario to reach the end of the level, there are auxiliary goals such as maximizing the coin count, game score, enemies killed, and so on. Information on the surroundings is constantly fed as an input. Mario's input, however, does not encompass the entire screen, but either a 3x3, 5x5, or 7x7 grid in which he is at the center. This grid keeps track of the objects or enemies in Mario's vicinity. While this paper did use a genetic algorithm in Super Mario Bros., it did not address how the algorithm fared when faced with deceptive choices during the course of evolution in the SMB environment. The second study uses deep reinforcement learning to create a generalized learning algorithm that is tested on 49 different video games [6]. However, it uses the entire screen as input and uses past experiences to predict the best move in a situation it has never seen. Interestingly, this algorithm does better than average human ability in around half of the games. Again, this paper did not go into any details of how the evolved agents behaved when put in deceptive situations.

These papers focus more on the real-time aspect of video games and how the in-game character behaves in the moment while our paper focuses more on excelling at one task. While these papers make excellent cases for why their algorithms are suited to the game at hand, there aren't any papers that compare the performances of different algorithms in one game environment. Specifically, we use two different methods of evolution on a deceptive task while the other three papers only use one, making it is difficult to see how those other algorithms compare relatively. Also, this paper explores the merits of both NEAT and Novelty search, showing experimentally the benefits and drawbacks of each evolutionary algorithm.

2 Hypotheses

Two hypotheses were tested. The first hypothesis was that Novelty would find the hidden rooms more often and in fewer generations than NEAT. Since Novelty does not get ensnared by deceptive tasks, its exploratory nature should help it discover the hidden rooms in fewer generations. On the other hand, NEAT is unlikely to find the hidden room as often and as quickly because it must be willing to explore an area with fewer coins. Additionally, finding the hidden room requires Mario to execute a duck move, a move that does virtually nothing elsewhere. We predict that NEAT will evolve to avoid duck movements because when executed, it rarely results in an increase in fitness.

The second hypothesis was that NEAT will earn a higher fitness than Novelty on average. NEAT will focus on collecting all the possible coins in the non-hidden level area and will eventually learn to perform actions such as successfully killing enemies and getting coins from the coin box. Novelty, however, will be more focused on finding new behaviors independent of how many coins the new behaviors collect. Thus, Novelty will not perform as well as NEAT in terms of coins collected unless Novelty finds the hidden room.

3 Simulation Environment

Though SMB simulators exist, the levels and gameplay details are hard coded so that users cannot access and modify how game levels are structured. Thus, we designed and developed a custom Mario simulation environment using Python and the Python Zelle graphics library. This environment allowed us to create game levels used solely for experimentation purposes. Although our simulation environment still retained most of the key features, such as the inclusion of Goombas, coins and coin boxes, the levels were much smaller and the in-game physics were not identical to that of SMB. The simulation environment is based on a two dimensional cartesian grid system whose origin is at the top left corner. The size of the environment, as well as how the game objects were arranged, were determined by parameters that the user set.

3.1 The World

The eight game objects in our Mario simulation world included ground tiles, platforms, coins, coin boxes, hidden rooms, the finish flag, Goombas, and Mario. The ground tiles (green squares) and platform objects (blue lines) gave structure to the world and defined where Mario (red circle) could travel. Mario could collect coins (yellow circles) by walking or jumping through them, which removed them from the world and incremented Mario's coin score. Coin boxes (brown box) worked differently. Mario could earn a coin by jumping into a coin box while standing underneath it. Afterwards, the box became white and contained no further coin value. The hidden rooms (red crosses) were accessible by ducking at a secret entrance and gave Mario a coin bonus of fifteen. Once a hidden room was visited, it was closed off. If Mario reached the finish flag (blue cross), he was rewarded with 5 coins and the run was terminated. Goombas (brown ovals) were hard coded to patrol back and forth in a predictable manner. When a Goomba walked into Mario, he died. However, if Mario jumped on top of a Goomba, it was removed from the game and Mario was given a coin.

In our simulator, we created a small world (27 x 7 grids) and a large world (49 x 9), each with an easy and hard version. In the easy version, all of the coin boxes were taken out. The hard, small world (Fig 1) consisted of 3 goombas, 9 coins, 4 coin boxes, and 2 hidden rooms. The easy, small world (Fig 2) was similar, except that the coin boxes were removed. The hard world had a total of 46 coins and the easy world had 42 coins.

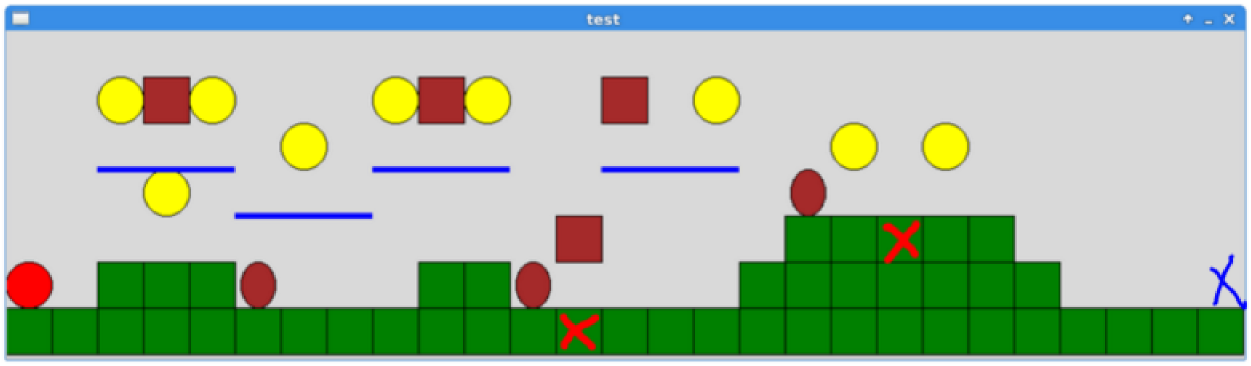

Figure 1: Small, Hard World

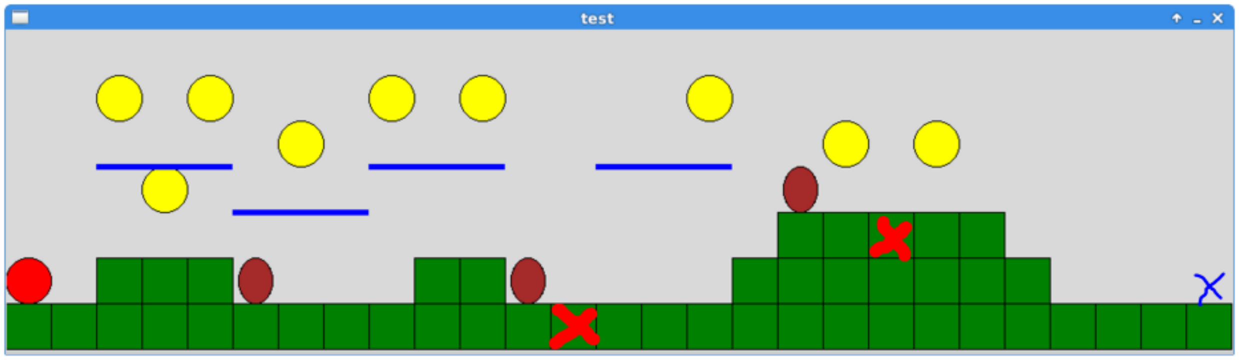

Figure 2: Small, Easy World

For the hard, large world (Fig 3), there were 4 Goombas, 25 coins, 13 coin boxes and 2 hidden rooms. The easy, large world (Fig 4) was the same except that all coin boxes were removed. The total coin score possible for the hard world was 78 and that for the easy world was 65.

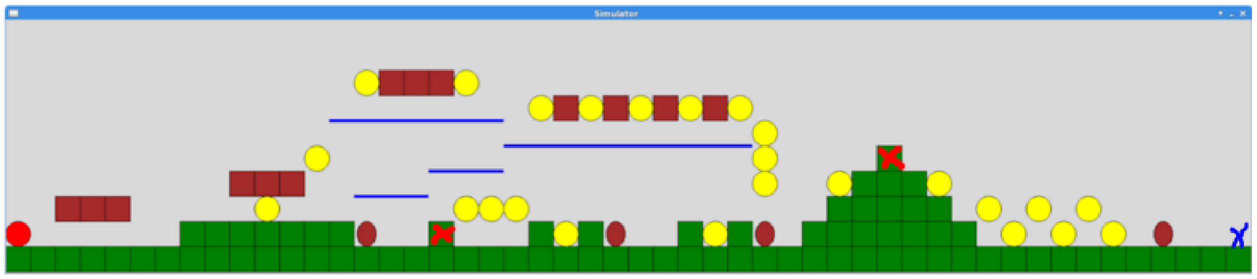

Figure 3: Large, Hard World

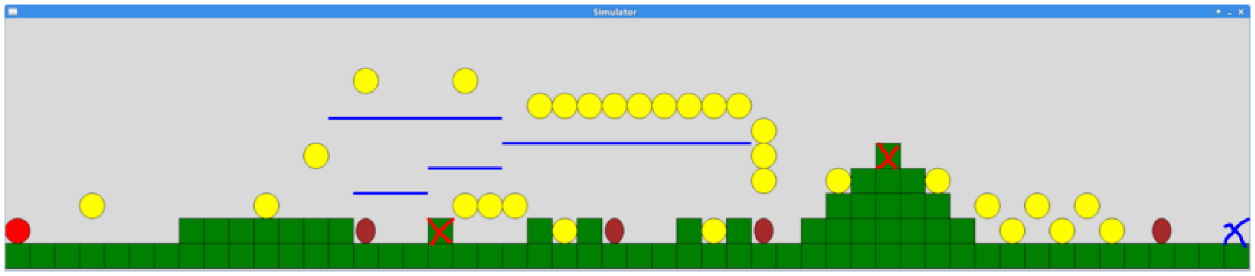

Figure 4: Large, Easy World

We included experiments in the large world for three reasons. First, we wanted to see if the results from NEAT and Novelty would be similar for both worlds. Second, we wanted to see if the size of the world had influence on Mario's evolved behavior. Third, by creating a larger world, we had more space to work on the design. For example, in the middle of the big world, Mario can take one of two paths: the path going up the platforms and the path below. By traveling the top path, Mario can earn 12 coins, but by traveling the bottom path, Mario can earn 23 coins. However, the bottom path also contains a hidden entrance, which is more difficult for Mario to exploit. This diverging path is somewhat deceptive. We were testing to see if Mario could learn to travel backwards to collect the coins he missed. An ideal behavior is Mario taking the top path, collecting all the coins, falling off and collect the three coins in the air, and then collecting all the coins that can only be obtained by taking the other path.

3.2 Mario

Mario was able to execute five different actions: moveLeft, moveRight, jumpLeft, jumpRight, and duck. MoveLeft and moveRight each consisted of 1 grid box translation in their respective directions. The jump moves required at least 2 timesteps. If Mario executed a jumpRight movement, he translated one grid space up, then one grid space to the right, then fell until landing on a valid standing space. The duck move kept Mario in place and was used as a special movement that allowed Mario to visit the hidden rooms. If Mario executed a duck movement on top of the entrance to a hidden room, he stood in place and gained a fifteen coin bonus for visiting the room. Finding the hidden room was the most difficult way to obtain coins.

4 Methods

4.1 NEAT

In this paper, we used standard NEAT with *tanh* activation. The fitness function was the normalized coin score. It is important to note that the NEAT fitness function represents the percentage of coins obtained. When comparing fitness scores between easy and hard worlds, keep in mind that the easy world has fewer coins than the hard world. It is meaningless to compare the fitness values between the small, easy world and the small, hard world. However, the number of generations required to find hidden rooms can be directly compared because the location of the hidden room was constant regardless of the difficulty.

4.2 Novelty

For the Novelty component, we also used *tanh* activation. Each behavior consisted of a list of unique (x, y) grid spots that Mario visited in the order that Mario visited them. The behaviors were padded with (0, 0) tuples so the behavior list lengths were equal to the max number of visitable squares in the world. The sparseness of a new behavior was calculated by averaging the Cartesian distances between the new behavior and the existing archived behaviors. The Cartesian distance between two behaviors is the sum of all the distances between corresponding behavior points in each list. The novelty fitness was calculated as the normalized average distance between the newest behavior and the kth nearest behaviors in the archive. We used a k of 15, a limit of 100 and a threshold of 0.25. For each generation, we also saved the best performing chromosome in terms of the objective fitness function that was described in the NEAT section above.

4.3 Experiments

To observe NEAT and Novelty, we conducted six experiments total using the four different worlds aforementioned, shown in Fig. 1-4, respectively. For the small world, we ran 4 experiments: two experiments with the Coin Type input and two experiments without. Each of those two experiments consisted of an easy and hard version.

The coin type, which will be explained a little bit later in this paper, was included as an experimental input to observe how impactful knowing the type of coin was on performance. For the large world, we ran 2 experiments. For both the easy and hard versions, we ran an experiment in which the coin type was included as an input. For each experiment, out of 10 runs, we recorded the frequency of finding one hidden room, the average generation at which the first room was found, the frequency of finding both hidden rooms, the generation at which the second hidden room was found and the average best fitness.

Experiment Number	Used Coin Type Input?	Size of World	Difficulty Setting
1	No	Small	Easy
2	No	Small	Hard
3	Yes	Small	Easy
4	Yes	Small	Hard
5	Yes	Large	Easy
6	Yes	Large	Hard

Table 1: Experiments Summary

4.4 Experiment Setup

For both NEAT and Novelty, each experiment was run ten times. Each run was evolved using 30 generations of populations of 100 individuals. We let each simulation run for 1500 time steps. However, the simulation was terminated early if Mario was either killed off by a Goomba or if he reached the finish flag. In addition, since human players often memorize key features in levels, where enemies are located, where power ups can be obtained, etc., we allowed the learning algorithm to evolve in a structured game environment that did not change in between runs so that the algorithms could take advantage of the predictability.

The 4 inputs into the neural networks were Mario's current coin score, the signed x-direction Cartesian distance to the nearest coin giving object, a stall sensor, and a coin type sensor. By *coin giving object*, we mean that Mario was given the distance to the nearest object that could give a coin. This could be a Goomba, coin box, finish flag, hidden entrance, or coin. The Coin Type input tells Mario what type of coin giving object is nearest to him. This sensor was implemented by assigning an integer value to each of the coin types. The stall sensor was implemented such that it outputted True if Mario's X position had not changed for 3 or more time steps. The stall sensor was included to help NEAT learn how to navigate coin boxes. Without the stall input, Mario would get stuck jumping into a coin box from the side. Only the coin type input returned values between -1 and 1; the rest of the inputs were not normalized.

5 Evolution Results

In this section, the results of the six experiments are presented. The behaviors are analyzed in more detail in Sections VI. For simplicity, we will refer to NEAT-Mario and Novelty-Mario as NEAT and Novelty respectively.

5.1 Small World

The experiments in this section tested NEAT and Novelty in the small world environment. Table 1 shows that in the easy world (no coin boxes) without the coin type input, NEAT and Novelty both had the same average best fitness. However, while NEAT and Novelty were both able to find the first and second room with the same frequencies, Novelty was able to find the first and second rooms in fewer generations than NEAT: 0.3 vs. 1.7 generations for the first room and 12.3 vs 18.5 generations for the second room. For the hard world, NEAT and Novelty were both only able to find one of the hidden rooms twice out of the ten runs. The average fitness always decreased drastically, from 0.83 to around 0.2. Evidently, the addition of coin boxes complicated the task. While both algorithms were able to find the first hidden room at the same frequency, Novelty was also able to find it in fewer generations than NEAT (3.33 vs 18 generations).

In Table 2, both NEAT and Novelty were given the coin type as an input. For the easy world, NEAT and Novelty were able to find both rooms at similar frequencies and had average best fitnesses that were higher than their counterparts that didn't have the coin type input. While Novelty found the first room slightly faster, NEAT was able to find the second room faster. For the hard world, Novelty was once again faster at finding the first room but slower at finding the second room. In terms of average best fitness, NEAT outperformed Novelty in the easy world but Novelty outperformed NEAT in the hard world. However, comparing the values between Table 1 and Table 2, clearly both algorithms performed better with the Coin Type input.

The results in this section show that Novelty sometimes performs better in situations in which less information is passed in as input data. When the coin type was not provided, Novelty was consistently able to find the hidden rooms at earlier generations than NEAT and generally had a higher average best fitness. However, when more information was provided as input, NEAT's performance was on par with that of Novelty's and NEAT even outperformed Novelty in the easy world. NEAT was also able to find the second hidden room at an earlier generation than Novelty. Thus, when faced with the easier task of navigating a world with no coin boxes and given enough sensor input, NEAT and Novelty both performed similarly. However, for the harder task of navigating a world with coin boxes, even with more input information, NEAT was not able to keep up with Novelty's performance in terms of average best fitness.

	Freq. 1st Found	**Freq. 2nd Found**	**Avg. Gen. 1st Found**	**Avg. Gen 2nd Found**	**Avg. Best Fitness**
NEAT (Easy)	10/10	8/10	1.7	18.5	0.83
Novelty (Easy)	10/10	8/10	0.3	12.3	0.83
NEAT (Hard)	2/10	0/10	18	n.a	0.192
Novelty (Hard)	2/10	0/10	3.33	n.a	0.235

Table 2: Small World Without Coin Type Input

	Freq. 1st Found	**Freq. 2nd Found**	**Avg. Gen. 1st Found**	**Avg. Gen 2nd Found**	**Avg. Best Fitness**
NEAT (Easy)	10/10	8/10	0.9	7.14	0.8812
Novelty (Easy)	10/10	8/10	0.7	12.3	0.8443
NEAT (Hard)	9/10	2/10	11.66	14	0.4823
Novelty (Hard)	9/10	3/10	5.44	15.33	0.5449

Table 3: Small World With Coin Type Input

5.2 Large World

This experiments in this section tested NEAT and Novelty in a large world with the coin type input. Table 5.1 shows that for the easy world, NEAT and Novelty were able to find the hidden rooms at the same frequencies and that Novelty was consistently able to find the rooms at earlier generations than NEAT. Novelty search resulted in a higher average best fitness. For the hard world, while NEAT was able to find the first room earlier than Novelty (13.57 vs. 20.57 generations), only Novelty was able to find both hidden rooms.

The results in this section further confirm what we observed in Section V. Since the world for these experiments already added difficulty to the task, it is no surprise that Novelty consistently outperformed NEAT in terms of when it found the hidden rooms and in terms of the average best fitness. The one outlier, however, is that in the hard world, NEAT was able to find the first hidden room faster than Novelty. This is offsetted though because while it took longer for Novelty to find the first room, Novelty was still the only algorithm to be able to find the second hidden room in the large, hard world.

	Freq. 1st Found	**Freq. 2nd Found**	**Avg. Gen. 1st Found**	**Avg. Gen 2nd Found**	**Avg. Best Fitness**
NEAT (Easy)	10/10	3/10	10	12.67	0.6130
Novelty (Easy)	10/10	4/10	2.86	10.29	0.6594
NEAT (Hard)	7/10	0/10	13.57	n.a.	0.4795
Novelty (Hard)	8/10	1/10	20.57	26	0.4807

Table 4: Large World With Coin Type Input

6 Analysis of Evolved Behaviors

In this section, the evolved NEAT and Novelty behaviors for the small and large world are analyzed in terms of evolved behaviors and performances.

6.1 Small World Behavior

In general, the coin type input did not significantly alter Mario's actions, it only sped up the evolution. In other words, the coin type input did not make Mario learn new behaviors, it only decreased the learning time. Adding the coin type input decreased the number of generations needed for NEAT to learn how to deal with coin boxes, obtain coins and kill Goombas. For Novelty, the coin type input decreased the number of generations before it found the hidden rooms. Additionally, for both the easy and hard worlds, the behaviors exhibited were generally similar. The addition of the coin boxes made the learning process longer for NEAT and increased the number of generations needed to find the hidden rooms.

For both the easy and hard small worlds, NEAT generally evolved to travel over the top platforms (top route) to find the right-most hidden room before exiting the level by reaching the finish flag. Often times, NEAT would find the left-most hidden room early on in evolution but would abandon the hidden room once it learned how to get coins from the top route. This shows that NEAT does indeed fall for the deception present in the task - even though NEAT could collect the maximum number of coins by going the bottom route and finding both hidden rooms, it chose to give up the bottom room and go for the top route in order to get a temporarily higher fitness.

Novelty's general behavior was split between either going through the top route and getting the right-most hidden room before reaching the finish flag or going the bottom route and getting both hidden rooms before reaching the finish flag.

6.2 Large World Behavior

NEAT exhibited three distinct behaviors: 1. take the bottom route, find the left-most room, walk past the right-most room and reach the finish flag, 2. take the top route, find the right-most room and reach the finish flag, and 3. take the bottom route, find both rooms and reach the finish flag. The third behavior rarely occurred. NEAT also evolved so that it's behavior reflected the structure of the world. In the beginning portion, Mario would just translate under the coin boxes until it reached the first goomba. After that, Mario would change strategies and just jump right until it either died or reached the finish flag. For the easy world, out of ten runs, NEAT exhibited behavior 1 three times, behavior 2 two times and behavior 3 two times. For the remaining two runs, the last generation of NEAT actually took the top route and didn't enter any of the bonus rooms even though it had found the first bonus room previously during evolution. For the hard world, NEAT exhibited behavior 2 seven times out of the ten runs. In three of the ten runs, NEAT found the first hidden room but then regressed and abandoned the hidden room in favor of the top route. The distribution of exhibited behaviors suggests that NEAT fell for the deception and evolved to take the top route over the bottom route.

The highest scoring Novelty behaviors were behaviors 1 and 2 from above and behavior 4 (take the bottom route, find the right-most room and reach the flag). For the easy world, the best scoring behaviors always took the bottom route. Out of ten runs, Novelty exhibited behavior 1 three times, behavior 2 two times, behavior 3 four times and behavior 4 one time. This shows that Novelty's exploratory nature allows it to roam more often along the more promising bottom route without any evolutionary pressure. For the hard world, the best behaviors also prefered to take the bottom route and out of ten runs, Novelty exhibited behavior 1 five times, behavior 2 three times, behavior 3 one time. For the remaining run, Novelty took the top route, didn't find any hidden rooms and reached the finish flag. Once again, the behavior distribution confirms that Novelty is more likely to find the hidden room along the bottom route. By frequently exploring along the bottom route, Novelty has greater potential for finding both rooms than NEAT has.

7 Discussion

7.1 Were our Hypotheses Supported?

Based on the number of runs we ran, we can see that, at least for the small world, giving NEAT and Novelty the Coin Type input made a significant difference. Before giving NEAT and Novelty the Coin Type input, there were clear differences in the number of generations required to find both the first and the hidden room. However, after giving them the input, that gap decreased noticeably. For example, without the input, it took NEAT roughly fifteen more generations on average in the small hard world to find the first hidden room than it took Novelty. After giving it the input, NEAT only required around six more generations. The number of generations required to find the second hidden room in the same setting was roughly equal for both NEAT and Novelty. By giving Mario the Coin Type input, we make the task less deceptive because Mario learns to map the coin type with the number of coins obtained. Since the task is less deceptive, NEAT is able to perform better than if it did not have the input, and is sometimes able to perform even better than Novelty search. Regardless, both NEAT and Novelty performed much better with the Coin Type input than without. Specifically for the small hard world with the Coin Type input, Mario was able to collect roughly two to three times more coins and the number of generations required to find both hidden rooms was smaller.

In the easy large world, Novelty finds the first room in roughly seven fewer generations than NEAT, and in the hard large world, NEAT finds the first room in roughly seven fewer generations than Novelty. We predicted that Novelty would find the hidden rooms in fewer generations than NEAT, so these results go against our hypotheses. One reason why the results seem to deviate from what we expected is that perhaps a bigger world requires more runs, or more timesteps for actual results to manifest. Randomness could be a factor, and if we were to conduct another ten runs, Novelty could perform better. Another potential reason is that the Coin Type input simply works greatly in NEAT's favor. That extra bit of information makes the task significantly less deceptive because NEAT knows exactly what type of coin is nearby.

7.2 Drawbacks

By simplifying the simulation environment and changing some of the in-game physics, we inadvertently introduced challenges into the simulation world. One such problem lies in the way that the jump commands were implemented.

By limiting the jump height to only one unit high, we caused the coin boxes to become major obstacles for Mario. Mario's only way then to get past a coin box is to translate. Thus, the coin boxes drastically slowed down evolution and it increased the number of generations needed before Mario was able to explore the world beyond the coin boxes. In the original game, Mario's jump was several units high and he was able to easily jump over the coin boxes.

For Novelty, another potential problem was the way that behaviors were defined. Since we defined a behavior as the set of unique locations that Mario visited in the order that Mario visited them, we didn't encode anything about the coin locations into the behavior description. Perhaps a better definition for the behavior could be the set of unique coins that Mario collected in the order that he collected them. This behavior definition would probably encourage more diversity in the evolved behaviors.

8 Conclusion

In conclusion, our hypotheses were partially confirmed. Our results show that both NEAT and Novelty found both hidden rooms at roughly equal frequencies and they produced roughly equal average fitness values. So, we focused on the number of generations required to find the hidden rooms. In the small world, the addition of the coin input made a significant difference. Without the input, Novelty would find the hidden rooms much quicker than NEAT. With the coin input, however, NEAT found the hidden rooms just as quickly as Novelty, with the exception of the hard world's first hidden entrance. The results from the big world show that NEAT is able to outperform Novelty when used in hard mode, and that the addition of coin boxes made it difficult for Novelty to find the first hidden room. This could have resulted from the lack of enough runs or that NEAT simply performs very well with the Coin Type input. From these results, we conclude that there is some correlation between the size of the world and performance and that NEAT can perform on par, and sometimes even better, than Novelty if it is equipped with the Coin Type input. However, Novelty did not significantly outperform NEAT as we had hypothesized.

References

[1] Seth Bling. Mari/o - machine learning for video games. 2015.

[2] Togelius, Julian, et al. Super mario evolution. *Computational Intelligence and Games*, CIG 2009, 2009.

[3] Kenneth Stanley, Bobby D. Bryant, Risto Miikkulainen. Evolving neural network agents in the *NERO* video game. *Proceedings of the IEEE(2005)*, pages 182–189, 2004.

[4] Joel Lehman, Kenneth Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2), 2011.

[5] Kenneth Stanley. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21, 2004.

[6] Mnih, Volodymyr, et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.