Developing Frogger Player Intelligence Using NEAT and a Score Driven Fitness Function

Davis Ancona and Jake Weiner

Abstract

In this report, we examine the plausibility of implementing a NEAT-based solution to solve different variations of the classic arcade game Frogger. To accomplish this goal, we created a basic 16x16 grid that consisted of a frog and a start, traffic, river, and goal zone. We conducted three experiments in this study on three slightly different versions of this world, all of which tested whether or not our robot (the frog) could learn how to safely navigate from the start zone to the goal zone. However, this was not an easy task for our frog, as it needed to learn how to avoid colliding with obstacles in the traffic part of the world, while it also needed to learn how to remain on the logs in the river section of the world. Accordingly, we equipped our frog with 11 sensors that it could use to detect obstacles it needed to dodge and obstacles it needed to seek, and we also gave our frog one extra sensor that provided it a sense of it's position in the world. In all three experiments, a fitness function was used that exponentially rewarded our frog for moving closer to the goal zone. In addition, we used a genetic algorithm called NEAT to evolve the weights of the connections and the topology of the neural networks that controlled our frog. We ran each experiment five times, and it was seen that NEAT was able to consistently find optimal solutions in all three experiments in under 100 generations. As a result, we deem that we were able to successfully demonstrate our method's ability to solve multiple representations of the classic arcade game Frogger.

1 Introduction

1.1 Frogger

The game of Frogger is a relatively simple game. A frog starts at the bottom of the screen, and the frog's objective is to travel across a road full of multiple lanes of traffic and across a river to the get to the top of the screen. The game of Frogger has a built in point system, where each move forward increases the total score by ten points, and each move backwards decreases the total score by ten points. Furthermore, the amount of time taken to get to the top of the screen is reflected in the final score, with a faster time resulting in a higher score, and a slower time resulting in a lower score. If the frog is hit by an obstacle or falls into the river, a life is lost and the points gained from that round are solely proportional to the distance the frog has traveled towards the top of the screen. The frog is given three lives at the start of the game, and only loses a life if it runs into an obstacle or falls into the river. Lastly, in some versions of Frogger, there are extra items placed around the world that can give point bonuses or act as extra obstacles.

Our implementation of Frogger contains the basic aspects described above: our frog must travel from the bottom of the screen to the top of the screen, it has to avoid five lanes of traffic, and it needs to avoid falling into the water by hopping from log to log. We also implemented a fly in

Experiment 3, which provided a significant bonus if our frog was able to "eat" it. However, there are a couple of differences between our representation of Frogger and the actual arcade version, the first of which is that we don't reward our frog for making it to the top of the screen in as little time as possible. Instead, we give our frog a maximum of 50 steps each life, but the final score does not reflect how many steps it takes for our frog to reach the goal, as long the number of steps taken is under the 50-step limit. In addition, in our version of Frogger the frog is given only one life per game instead of three. We also differ in our scoring system by exponentially rewarding our frog for moving closer to the goal zone, rather than using a simple linear function.

1.2 NEAT and Related Works

To create a robot capable of learning and solving our complex world, we chose to implement a genetic algorithm called NeuroEvolution of Augmenting Topologies (NEAT) in all three experiments [3]. In many ways, NEAT is very similar to other genetic algorithms, and accordingly it evolves members of a population by selecting the most fit individuals and using them to reproduce the next generation via crossover and mutation. NEAT evolves both the weights of the connections and the topology of neural networks, and the evolutionary process continues until an individual is created that has a desired level of fitness. NEAT is distinct from other genetic algorithms in a few keys ways. First of all, it starts the evolutionary process with a population of small, simple networks, and then complexifies these networks over multiple generations. This procedure is utilized because it mimics how organisms evolved in nature: basic single cell creatures eventually evolved into complex, multiple-celled animals. The other unique aspect of NEAT is that it employs speciation, which protects innovation within a population. This is done by NEAT keeping track of how closely related individuals are to one another, and it allows certain species to grow and advance without having to compete directly with previously established members of the population. In addition, NEAT does not allow different species to crossover and mate with one another, and as a result speciation leads to simultaneous solutions being evolved in the same population, increasing the chances of finding an individual with the desired level of fitness.

While NEAT can be used in many different settings, it has been shown to be an effective strategy to evolve agents in a world similar to the one that we created. In a study conducted by Wittkamp, Barone, and Hingston, NEAT was utilized to evolve different strategies in the game of Pacman [2]. However, unlike our experiments, which attempt to imitate a competent human Frogger player, their study focused on developing an alternative method of intelligence for the computer controlled "bad guys". It was shown that an effective team strategy was able to be found by using NEAT, and thus, despite the fact that their study evolved solutions for a team of "bad guys", rather than one individual, we deem it showed that we could realistically evolve a successful player in our game world as well

In addition, based off the findings of Chasins and Ng, it can be seen that there are two distinct ways in which a fitness function can be created: one-reward-based or multiple-reward-based [1]. In their experiment, the one-reward-based fitness function worked in an all or nothing manner, and the reward was given only if the goal was achieved. In reality, another small reward was added to ensure a continuous function, but the overall structure of the fitness function was still the same. On the other hand, the multiple-reward-based function awarded points for reaching the goal state, but additionally rewarded the robot for the distance it had traveled from the start point or its last know location, thus rewarding progress and not just an end state. After testing these two approaches, it was concluded that both methods demonstrated relatively equivalent rates of success in evolving

solutions for their robot's task. Nonetheless, we felt that using a multiple-reward-based function was the best approach for the game of Frogger, and thus we employed this method in all three experiments we ran. We deemed this was the case because it was extremely unlikely for our frogs to stumble upon the goal by sheer luck, and consequently we felt that if we only rewarded our frogs when they reached the goal, the majority of our frogs would have the same fitness level. By using a multiple-reward-based function, we were able to reward incremental progress, and hence we were able to consistently see improvement from generation to generation, regardless of if the goal was ever reached. Furthermore, we also wanted our function to closely reflect the scoring of the actual game of Frogger, and we believed that this incremental approach accomplished that better than its all-or-nothing alternative.

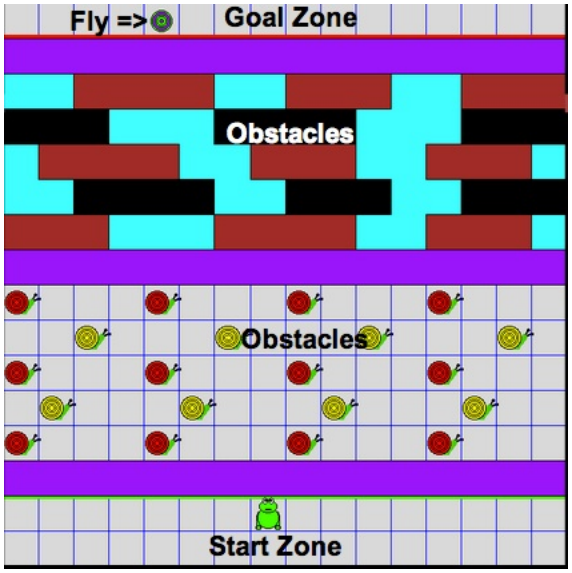


Figure 1: The world used in Experiment 1.

2 Experiments

2.1 Environment

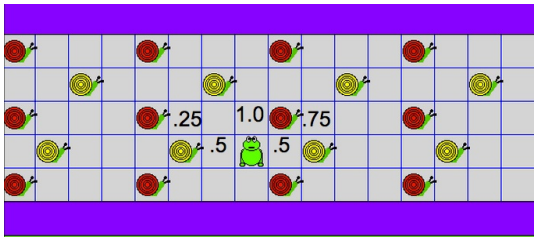
The Frogger world, which can be seen in Figure 1, is a 16x16 grid divided into three distinct regions:

Start Zone: This region spans from the bottom of the window to the green line. It is completely free of obstacles, and is the initialization point of our frog for each trial. Specifically, the frog is initialized in the top center of this region.

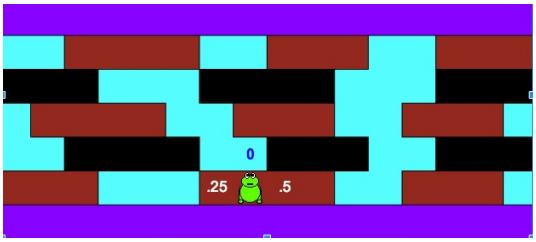
Traffic Zone: This is the area between the first and second purple rows. In this region, there are five rows of snails, and each snail moves at a constant speed. Three rows of red snails move from left to right, and two rows of yellow snails move right to left. When a snail moves off the screen, it immediately moves to the opposite side of the screen, where it is randomly placed in either the closest or second closest square to the edge and then continues its normal trajectory. If our frog ever collides with a snail, it immediately dies, and the game is over.

River Zone: This is the area between the second and third purple rows. In this region, there are five rows of logs floating on water with each log moving at a constant speed. The first, third, and fifth row of brown logs all move from left to right, while the second and fourth row of black logs all move from right to left. When the entire log moves off the screen, it immediately moves to the opposite side of the screen, where it then continues its previous trajectory. In the first experiment, the logs are three grids wide, while in the final two experiments they only fill one grid each. If our frog ever falls off of a log into the water, our frog instantaneously dies, and the game is over.

Goal Zone: This region is the area between the red line and the top of the window. A trial is immediately completed once our frog moves past the red line. In Experiment 3, a fly was randomly placed in this area. Our frog was not allowed to move once it entered this region, but if it landed on the fly when it first moved into this section, it received a huge point bonus.



(a) The frog is equipped with five sensors in the snail section of the world. The side left and side right sensors detect snails that are directly horizontal to the frog, and the top left and top right sensors detect snails that are one row above our frog to the left or right. The center sensor detects if a snail is in one of the three squares directly in front of our frog, and it returns a value of either 0 or 1. A value of 1 indicates that a snail is in the center sensor's range, while a 0 indicates that no snails are present. The other four sensors all register a value of 1, .75, .5, .25, or 0, depending on how far away an object is from the frog. For example, if a snail is one square away from the frog, the sensor's value will be .75, if it is two squares away the value will be .5, if it is three squares away the value will be .25, and if it is four or more squares away the value will be 0.



(b) The frog is equipped with four sensors in the river section of the world. The side left and side right sensors detect how much space there is on the log to the left or right of our frog, with higher values indicating that there is more room. In this figure, there is one grid the frog can move to the left, so the left sensor's value is .25, and two grids the frog can move to the right, so this value is .50. The front sensor simply indicates whether or not a log is directly in front of the frog, with a 1 indicating that this is the case, and a 0 indicating no log is present. Lastly, there is a sensor that simply indicates whether or not our frog is in the river section of the world, outputting a 1 if our frog is in the river section and a 0 if not.

Figure 2: A description of the sensors used in the traffic and river zones.

2.2 Sensor Inputs and Motor Outputs

In order for our frog to be able to navigate the world, we gave it twelve independent sensors, all of which returned values between 0 and 1. Five of these sensors were used in the snail section of the world, with a large value indicating that a snail was very close to the frog. Two of these sensors detected snails that were positioned directly to the right or left of our frog, two sensors detected snails that were positioned one row above our frog to the left or right, and one sensor detected if a

snail was in one of the three squares directly in front of our frog. Except for the center sensor, each of the other sensors perceived objects up to three squares away, with the boundaries of the grid not appearing as obstacles/objects. Each sensor could only see the closest object; thus, if there were two objects in the center sensor's view, only the closest would be registered. The snail sensors can be seen in Figure 2a.

There were four sensors that were used in the river section of the world, the first of which simply returned a 1 if our frog was in this region, and a 0 if not. The other three sensors detected the presence of logs. Two of these sensors detected how much room there was for our frog to move to the right or left on the log, with higher values indicating that our frog had a lot of space to move on the log in that direction. The fourth sensor detected whether or not a log was directly in front our frog, with a 1 indicating that a log was present, while a 0 indicated that a log was not. The log sensors can be seen in Figure 2b.

Finally, we provided three sensors that were used only in Experiment 3, and they detected the presence of a fly bonus. These three sensors worked similarly to the snail sensors. Two sensors detected whether or not the fly was one row above our frog to the left or right, and a high value indicated that the frog was close to the fly. Both side sensors could sense a fly up to three squares away. The last sensor returned a 1 if the fly was directly in front of our frog, and a 0 if not. The fly sensors can be seen in Figure 3.

Based on these sensor values, our frog was given four possible motor outputs: left, right, forward, and stay. The first three of these movements corresponded to our robot jumping one square in the specified direction, and the last movement, stay, kept the robot in its current grid position. We chose not to implement a backwards motion since, due to our fitness function, forward, sideways, or no progress are always more desirable.

The NEAT configuration file was adjusted to take in twelve sensory inputs and output an array that consisted of four values. Each index of the output array was a value between 0 and 1, and the motor output chosen by the robot was the highest value in this array. For example, the frog would move one square to the left if the first value of the output array were the largest.

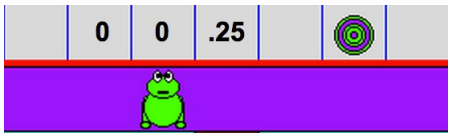


Figure 3: The frog is equipped with three sensors to detect the presence of a fly, and these sensors were only used in the third experiment. The side left and side right sensors detect if the fly is one row above our frog to the left or right, and the center sensor detects if a fly is directly in front of our frog. The center sensor registers a 0 or 1, with a 1 indicating that the fly is one grid above our frog, and a 0 indicating no fly is present. The side sensors register a value of 1, .75, .5, .25, or 0, depending on how far away the fly is from the frog. For example, if the fly is one square away from the frog, the sensor's value will be .75, if it is two squares away the value will be .5, and if it is three squares away the value will be .25.

2.3 Fitness Function

In a perfect game of Frogger, the frog would never collide with an obstacle or fall into the river, and it would land on a fly as it entered the end region. Our fitness function reflected this scoring system by giving a perfect score if our frog was able to do this; however, our fitness function also rewarded incremental progress in the world. We exponentially rewarded our frog the closer it got to the goal zone. Specifically, this was done by raising the fitness of the frog by two to the power of the number of squares travelled in the vertical direction. For example, if our frog travelled four rows before colliding with a snail, its final fitness would by 2^4, or 16. Furthermore, we did not explicitly penalize falling into the river or colliding with a snail, but we immediately ended the trial, and thus the frog was unable to acquire a higher score. Lastly, if our frog ever landed on the fly, we greatly rewarded the frog by doubling its final score.

The minimum score each frog could acquire was 1, or 2^0, which meant that the frog did not move forward at all. The maximum score that could be achieved was 2^{14} in the first two experiments, or 16384, and 2^{15} in the third experiment, or 32768. This difference in the maximum level of fitness was due to the fact that the third experiment included the presence of a fly, and our frog was able to double its final score if it entered the goal region by landing on it.

2.4 Experimental Procedure

In all three experiments, we evolved our population for one hundred generations, and we repeated it five times to ensure consistency and reliability. Every population consisted of 200 members where each member had three trials in the world. The fitness of each member of the population was calculated by averaging all three of these trials' fitness scores. This information was stored in archives based on generation, and could be graphed for enhanced visualization of our frog's fitness evolution. Lastly, each experiment took approximately two hours to complete.

All three experiments followed the procedure described above; however, all the experiments had uniquely altered world structures. Experiment 1 consisted of 3x1 grid logs, while Experiment 2 consisted of 1x1 grid logs. Finally, Experiment 3 added the presence of a fly on to Experiment 2, and gave the frog a large reward for landing on it.

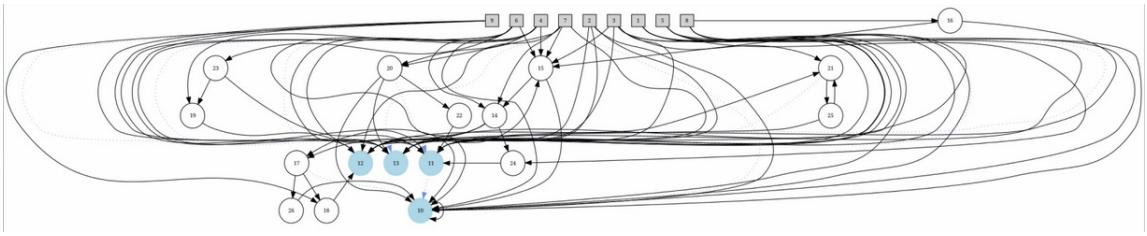
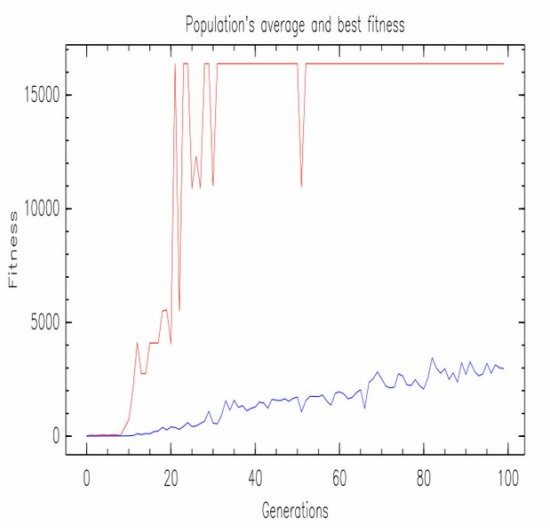


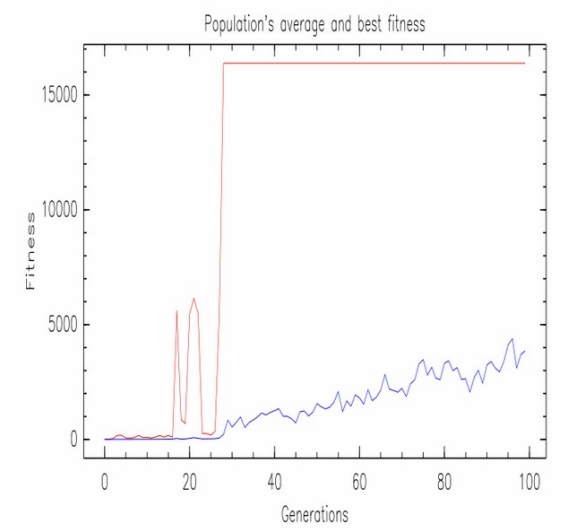
Figure 4: This picture shows a typical structure of a network in the final generation of the first experiment.

3 Results

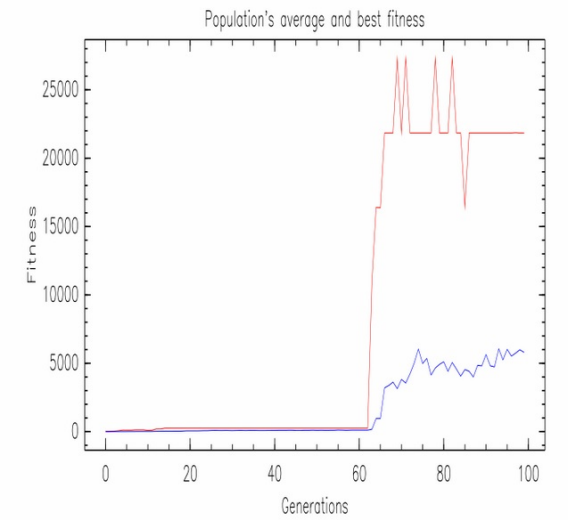
The world in the first experiment consisted of logs that were three grids wide, and it was seen that in all five runs we were able to evolve a frog that could consistently and efficiently reach the goal



(a) This graph shows the progression of a typical population's average and maximum fitness in the first experiment.



(b) This graph shows the progression of a typical population's average and maximum fitness in the second experiment.



(c) This graph shows the progression of a typical population's average and maximum fitness in the third experiment.

Figure 5: This figure displays a typical progression of a population's average fitness level from each experiment.

zone. On average, it took our frog approximately 20 trials to do this; however, in all five runs it took our frog around 50 generations to dependably achieve a maximum score. We believe that this pattern emerged because our frog was able to sometimes reach the goal zone without fully learning how to best navigate the logs in the river section of the world. This was possible because the logs take up over half of the surface area in each row in the river section of the world. Thus, it was relatively easy for our frog to partially learn how the logs worked and still make it to the top of

the screen. As the experiment progressed, however, our frog eventually learned to never fall into the river, which illustrated NEAT's ability to consistently solve a simple Frogger world. Figure 5a depicts the results of typical run from Experiment 1, and Figure 4 shows the structure of a typical network in the final generation.

In Experiment 2, we changed the width of the logs from three squares to one in an attempt to prevent our frog from stumbling upon an incomplete strategy that only worked part of the time. In this experiment, while it took our frog a longer time to find a solution than in experiment one (usually around 40 generations instead of 20), once the maximal score was achieved, it was continuously realized in every subsequent generation. Figure 5b depicts the results from a typical run from this experiment. This experiment illustrated how NEAT could evolve a solution to a world that could not be mastered with a sub-optimal strategy.

In the third experiment, we expanded upon Experiment 2 by adding the presence of a fly to the goal zone, and if our frog landed on the fly, it received a huge bonus. Similar to the second experiment, it was seen that not only was our frog consistently able to make it the goal zone, but it was also able to reliably locate and "eat" the fly. Nonetheless, it is important to note that the best frog rarely found the fly in all three trials. We believe this was the case because the fly was randomly placed in the top row, and because our fly sensor could only sense the fly if it was three squares away, often times the fly was too far away for our frog to sense. Thus, our frog would continue to go straight, and consequently it would miss out on the fly bonus. However, our frog still located the fly well above the level of chance alone, and hence we deem this experiment to be a success. Figure 5c depicts a typical run from Experiment 3.

4 Discussion

The results of our experiments show that by using NEAT in conjunction with a good fitness function, we were able to effectively create an individual that could solve basic representations of the game Frogger at maximum or close to maximum level of efficiency. This finding has a couple of implications, the first of which is that NEAT can be utilized to evolve individuals within a video-game world. While evolving the main player might not be that useful in practice, in theory we could use a similar type of method to evolve the non-player-controlled robots that are popular in many video games today, and thus come up with a unique way to customize the game play for each player. Nonetheless, while this finding is encouraging, it is important to note that Frogger is a very simple game that consists of a limited number of predictable moving parts, and thus we think that in the future it would be interesting to see if NEAT could find a solution to a more challenging world. Nevertheless, this study shows yet another example of how NEAT can successfully evolve robots, and thus this study could potentially serve as a template of how to evolve robots that could be employed in different types of virtual worlds, as well as the real world we live in.

References

[1] Sarah Chasins and Ivana Ng. Fitness Functions in NEAT-Evolved Maze Solving Robots. *Tech Report*, 2010.

[2] Philip Hingston Markus Wittkamp, Luigi Barone. Using NEAT for Continuous Adaptation and Teamwork Formation in Pacman. *Computational Intelligence in Games*, 2008.

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