Genetic Algorithms

Ryan De Haven
California Polytechnic State University, San Luis Obispo

State Based agents: Mario

A useful test-bed for experimenting with genetic algorithms is in the field of video games. The paper “Infinite Mario Bross AI using Genetic Algorithm” presents an experiment using a genetic finite state based AI to move Mario through different levels. [3] Each test run puts Mario through a level made up of monsters and obstacles that the agent must navigate. He is awarded a fitness score depending on how far he gets. The figure below shows an example Mario finite state machine.

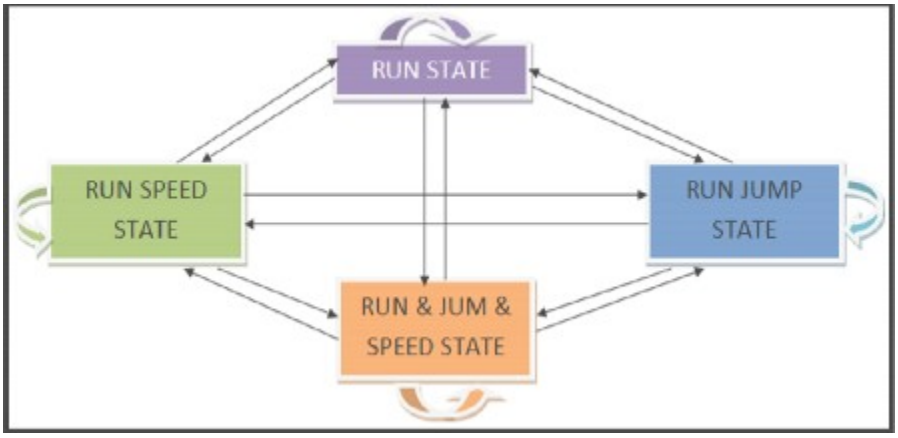

figure 1.

With generated code looking like the following:

```
Switch (State):
case 0: //RunState();
if (seen nothing)
state = 0;
else if (seen enemy)
state = 1;
else if (seen enemy || seen obstacle);
state = 2;
else if (seen enemy || seen hole);
state = 3;
break;
```

The paper found that an 80 percent crossover rate was optimal to create a finite state Mario that could complete the level.

Neural Genetic Agents: Mario and NEAT

The goal of Neural Genetic Agents: Mario and NEAT project was to create genetic algorithm using NEAT and Infinite Mario. NEAT was used to control the genetic algorithm and neural net

configuration of the Mario agents. These agents were run over many generations to create neural topologies that could run in Infinite Mario's world.

The Mario agents have a maximum of 101 inputs from the Mario world consisting of close proximity blocks and enemy placement as well as 6 outputs: up, down, right, left, jump and fire. These inputs and outputs are run through the neural net of each Mario. The agents work collectively through a genetic algorithm to find an agent that can progress the furthest through the level with a given neural net. NEAT allows for agents to develop different neural topologies allowing for new capabilities to be formed. The Mario Agent I implemented used 21 inputs for simplicity and faster results.

Agent design

The genetic description about how to initialize the neural network and how to genetically change the individuals is loaded from a “.ga” file at start up. The file contains parameters for the genetic crossover rates and mutation rates.

After reading the configuration file, the NEAT aspect of the program loads a set of initial genes describing basic neural configurations to start the first generation. The NEATGeneticAlgorithm class is then called to run the first epoch of agents. The agents are run in the Infinite Mario environment and the fitness score returned is given to the NEATGeneticAlgorithm class. The NEATGeneticAlgorithm class then takes certain individuals and creates new topologies by crossing individual and applying mutations according to the configuration file. The process is repeated until the desired Mario is created.

Agent Implementation

Most of the implementation work was creating the framework to allow NEAT to control Mario agents in Infinite Mario. I used the NEATGATrainingManager class as a starting place to create the 'main' method in order to further create and evaluate agents. After I figured out how that was running experiments I created an agent for Infinite Mario that took in a neural net from NEAT and wired the appropriate inputs and outputs. Once Mario started jumping around with a default configuration I moved to tweaking the program to get a Mario that could complete static levels. In order to make sure the algorithm was creating good Mario agents I set up a system that allowed you to see each agents fitness score at the end of its test. Every time a Mario completed a level it would display that Mario neural net actually doing the level. NEAT also let me visualize the neural net topologies that were being generated. That way I could see how each setting changed how the neural nets where formed.

While trying to optimize the settings I found that convergence to a barely working Mario from a new generation of nets took 50 to 75 generations. Looking at the neural nets let me see that with 101 inputs the genetic algorithms would have to guess which one too hook up. I decided to decrease the inputs to 21, where Mario would only look for enemies and blocks one square away. Reducing the number of inputs into the neural net significantly increased the performance in the growth of the Mario agents. Instead of waiting 50 to 75 generations for a fitness value of 1000 it only took 10.

Below is a 40th generation net developed for the 2nd difficulty of Infinite Mario.

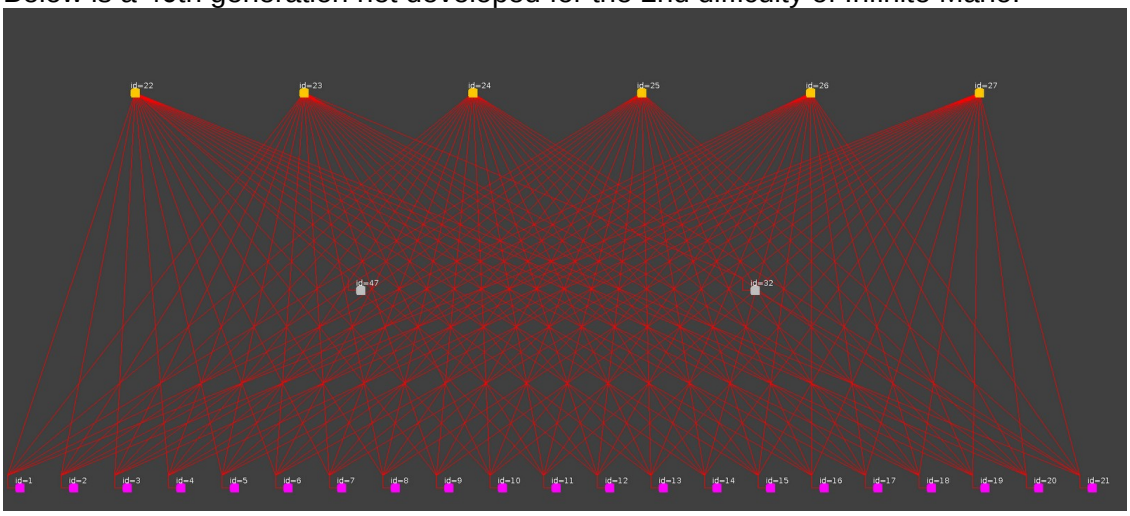

figure 2.

Complex Marios

After the agents were able to complete the first level I implemented a dynamic level change each generation so that the seed for the level would randomly change. When Mario completed two different levels of the same difficulty he would then move to the next level of difficulty. These changes allowed for a more complex Mario to develop as shown by the following topology created for a Mario agent that could complete the third difficulty level.

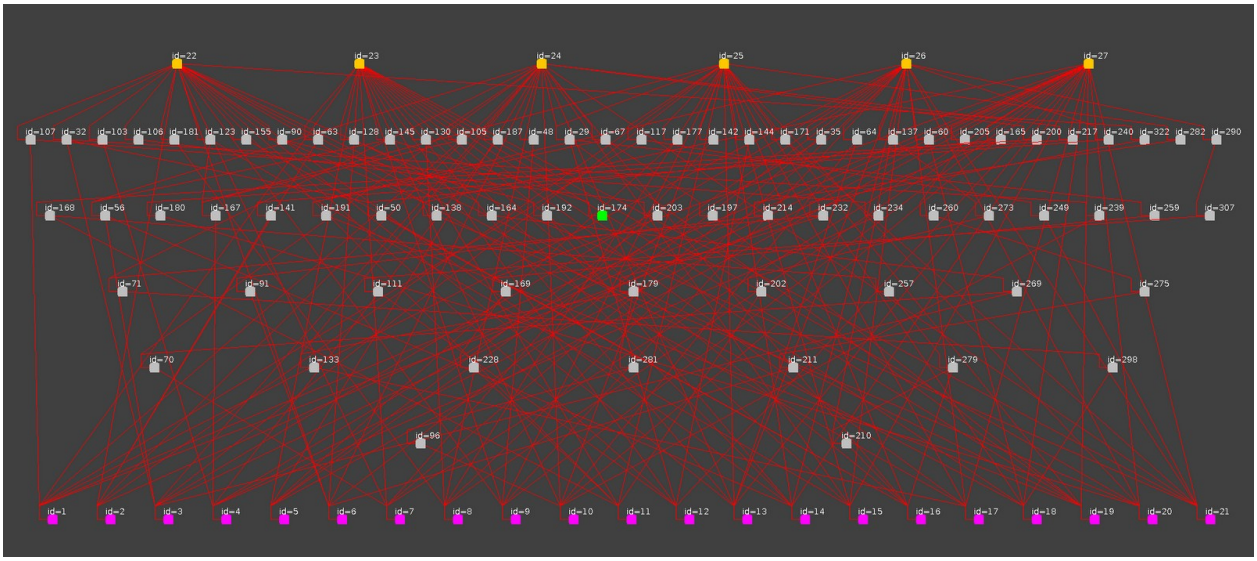

figure 3.

The complexity is derived from the number of situations that Mario is learning, as the level increases, there are more situations that he must memorize in order to move through the level.

This neural net had 5 levels of hidden networks allowing Mario to perform extremely well on a moderate difficulty level. This took several hundred generations. Observing this Mario showed that Mario was relying on luck and a run and jump strategy. Mario evolved in such a way that he would always do well on the heuristic and would succeed by luck or the right jumping pattern. In order to compensate for this reliance on luck, a better heuristic would be needed to show Mario exactly what actions were good in an evaluation that did not change every generation.

Training

After seeing Mario perform poorly at developing intelligent techniques for defeating enemies and jumping over pits I tried training Mario on a level full of random pits. After Mario could complete the level of pits, he was tested on a random level with pits. The Mario agents were able to complete the custom pit level and learn how to recognize and jump over pits. A problem occurred on the random level where monsters and pits where located, all the Marios that had previously learned to jump over pits were eliminated due to the monsters killing the jumping Marios early on before jumping became important.

Results Compared to Finite Mario

The NEAT genetic Mario converged much faster than the Finite state Mario. Where the neural nets took 10 to 50 generations to complete the first level, it took the finite Mario 17 to 89. The finite paper did not describe any tests run past the first level.

I felt that the genetic Mario could not solve problems that required moving backwards due to the way the heuristic was set up. There was no way to award him for doing things in a logical way, only a way to reward him for progress and the number of enemies killed. The heuristic usually meant that Mario would learn certain techniques for levels and would not perform logical operations where the heuristic failed to describe the goal.

When the levels were changed Mario seemed to loose the ability to perform well on the earlier set of levels, at least until those genes that expressed the abilities were turned on.

Neural genetic agents: Galactic Arms Race

The Galactic Arms Race or GAR incorporates a type of NEAT for generating content in the game called cgNEAT.[4] In the game, the player fights aliens and collects weapons for further fights with aliens. The genetic algorithm cgNEAT generates the weapons the player can obtain based upon the ones that all players in the game are collecting and using. The neural networks created are compositional pattern producing networks that generate the way the weapons looks. [4]

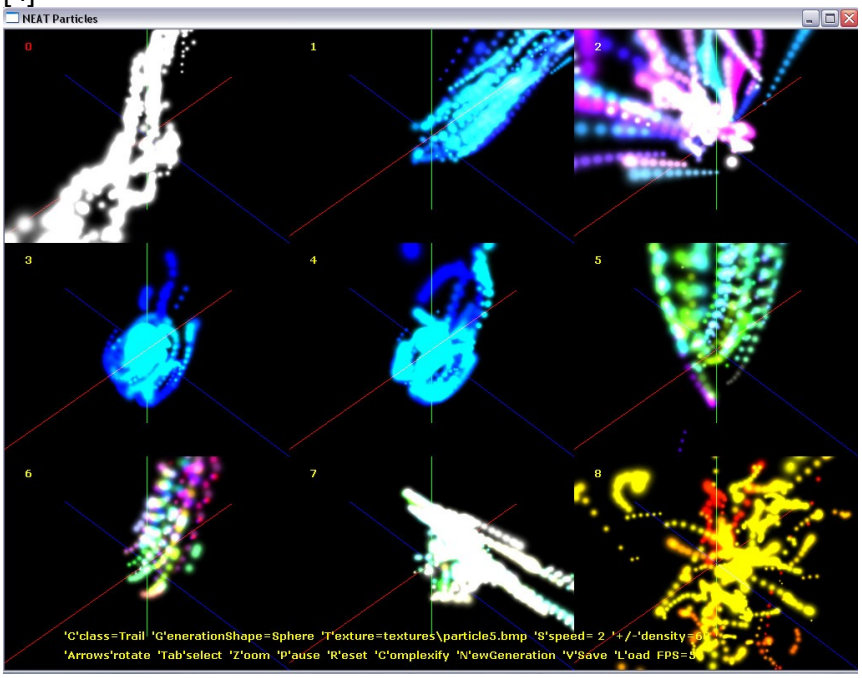

figure 4.

The above picture shows different patterns created by the neural network.

GAR starts the users in the game with a set of weapons that are “starter weapons” that are not in the genetic pool. Then players can find weapons spawned in the game world that are created by cgNEAT. Items that are picked up by players and used are then added to the population of offspring that will reproduce. An example of a weapon evolving the game of GAR is shown below.[4]

figure 5.

Due to players picking weapons that work better for them, it can be seen that later generation perform better than earlier generations as seen above. GAR shows promising development for genetic algorithms that can create usable content.

Neural genetic agents: NERO

Neuro-Evolving Robotic Operatives or NERO for short, is a game based on the rtNEAT implementation of NEAT.[5] In the NERO game the player trains and uses robot units to complete certain tasks. The main part of the game play is defending and capturing towers against another team of trained robots.

The most interesting aspect of NERO and rtNEAT is that the agents are created in real time. This means that during training or game play, agents will be removed and replaced with new neural nets derived from species in the given population. rtNEAT only selects parents for new agents from those that are old enough to have been evaluated. This avoids the problem of removing the fit from the population due to improper evaluation.

Training mode in NERO involves giving the player 50 units to run through a course defined by the player. The player sets a spawn point for the robots where they will have to move from and complete a certain task. After a set amount of time the robots will be restarted from the spawn point with a new brain. The restart is to make sure that no robot is at an advantage when it has its brain replaced. As with the original NEAT neural nets for the initial robots start with a randomly connected topology. The outputs and inputs default to a simple approach with only 3 outputs and 13 inputs as shown in figure 6.

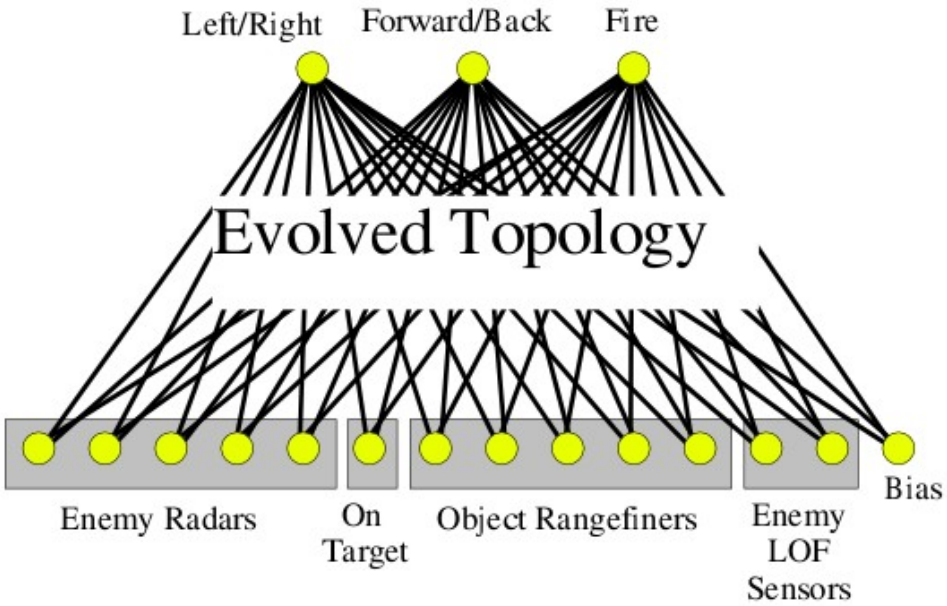

figure 6.

An example of user training is navigating a maze as seen below.

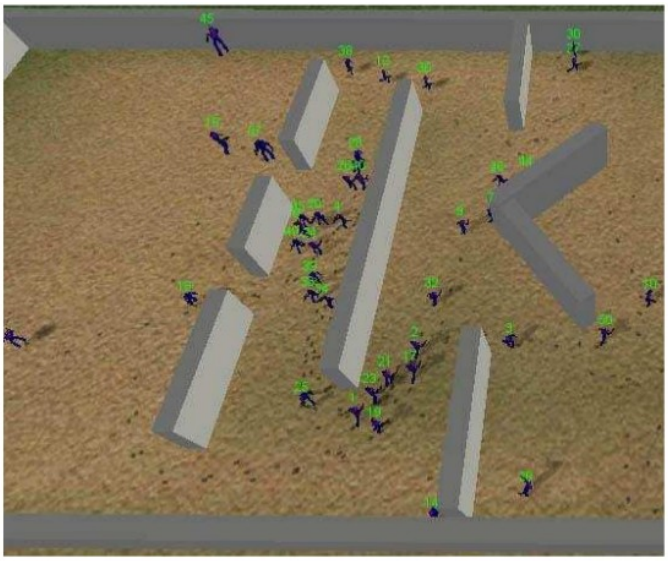

figure 7.

Players could set up different scenarios using obstacles such as turrets and walls. The player would then give the robots a fitness score based on certain performances of the robot. By training agents to do different tasks a player could assemble a team of robots to face against another team.

By allowing real time replacement the user can see clearly how the training is affecting the algorithm and is improving the intelligence of the agents. For this reason, NERO is a great way to learn about genetic algorithms and neural net evolution.

Discussion

The neural genetic algorithms seem to be the most promising type of genetic algorithm, where the end product can function and interact in a dynamic way. Content generation and procedural generation will be able to make use of techniques presented in cgNEAT and I can see these being used in the next generation of multilayer games. With games that self generate content, the developers will not have to develop new content to satisfy the end customer.

NEAT's ability to successfully build topologies without creating bloat allows it to be used in applications like cgNEAT and in my NEAT Mario implementation. In the Infinite Mario environment the agent is only given 40 milliseconds each frame to compute the next move. NEAT does a great job of forming a network that satisfies the computational time constraint. During the tweaking phase of the NEAT algorithm I found many of the neat variables could kill a genetic population. If the genetic algorithm had a way to detect what variables were causing it to fail, it could perform much better.

Conclusion

Genetic agents provide an advantage over standard agents with their ability to improve the way they can interact within an environment. For content generation this means that the algorithms can adapt in new ways that standard algorithms cannot. One of the drawbacks to the genetic agents for AI is that the agent needs to be trained. Sometimes it can be difficult to find data that will train an agent properly.

In the future of agents expect to see genetic agents where the solution space is large and the possibilities endless.

[1] Evolutionary Computation, 10(2):99-127, 2002.

[2] Markus Persson. Infinite Mario Bros. http://www.mojang.com/notch/mario/. 2011.

[3] Ng Chee Hou; Niew Soon Hong; Chin Kim On; Teo, J.; , "Infinite Mario Bross AI using Genetic Algorithm," Sustainable Utilization and Development in Engineering and Technology (STUDENT), 2011 IEEE Conference on , vol., no., pp.85-89, 20-21 Oct. 2011

[4] doi: 10.1109/STUDENT.2011.6089330
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6089330&isnumber=6089310

[5] Hastings, E.J.; Guha, R.K.; Stanley, K.O.; , "Automatic Content Generation in the Galactic Arms Race Video Game," Computational Intelligence and AI in Games, IEEE Transactions on , vol.1, no.4, pp.245-263, Dec. 2009
doi: 10.1109/TCIAIG.2009.2038365
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5352259&isnumber=5360672

[6] In Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05), Piscataway, NJ, 2005. IEEE.