NEUROEVOLUTION OF AUGMENTING TOPOLOGIES

GROCERY RUN

How the Game Works

The game is fairly simple. The player controls a character that can only jump, hop, or move down. If the player is midair when moving down, the character accelerates quickly towards the ground. If the player is already on the ground, the character crouches.

Obstacles will begin moving toward the character at random intervals, and the player must evade them with the aforementioned movements. The game gets progressively more difficult by increasing the speed in which obstacles travel, and lowering the interval between obstacle spawns.

A variety of obstacles can spawn, which demand that the player either jumps or crouches. As the game gets more difficult, the player gets less time to make such a decision.

Problem Space

The game itself is not terribly difficult. Theoretically, the entire game could be beaten on the basis of pure simple calculation. Set some conditions to always crouch under flying obstacles. For other obstacles, calculate the distance to the obstacle and how long it takes to jump. With that bit of information, it takes simple math to determine the correct jump timing. However, this went against the spirit of artificial intelligence, so we decided to take a more "interesting" route by going with a genetic algorithm.

The genetic algorithm largely takes care of itself with regards to improving the agent. It is meant to emulate natural evolution, so provided the correct environment, it takes care of the rest. For the programmers, the focus is creating the optimal environment.

Therefore, the problem space focused primarily on input. What inputs should be provided to the agents? Too much data can result in slower evolution, as unnecessary mutations are wasted on irrelevant input. However, too little data would mean that the agents would not have sufficient data to actually beat the game.

The Algorithm

The Neuroevolution of Augmenting Technologies (NEAT) was used to create the neural network of the AI agents. In short, the algorithm works by initializing agents with a minimal neural network, an input layer, an output layer, and one connection between the two. The agents play the game, and their scores are recorded as their fitness level. Weaker players are selectively removed, and better performing agents are allowed to reproduce. At each generation, players are also mutated to allow for diversity and more complexity in neural networks. In a neural network context, a mutation can either be a mutation in the weight of an existing connection between nodes, or a completely new addition of a connection or node.

However, simply removing weaker players would also remove any valid attempts at mutation. While most mutations are bad, the trend towards beneficial behavior is often characterized by a series of individually harmful mutations. To protect these "innovative" mutations, the population is divided into species of similar agents (agents with a similar genome), and agents must only compete with members of their own species. With speciating, even if a particular agent does poorly compared to the entirety of the population, it can still be allowed to reproduce if it does well within its own species. This is key in allowing the population to produce an optimal agent. If a species does not improve after a set amount of generations, the species will then be deleted.

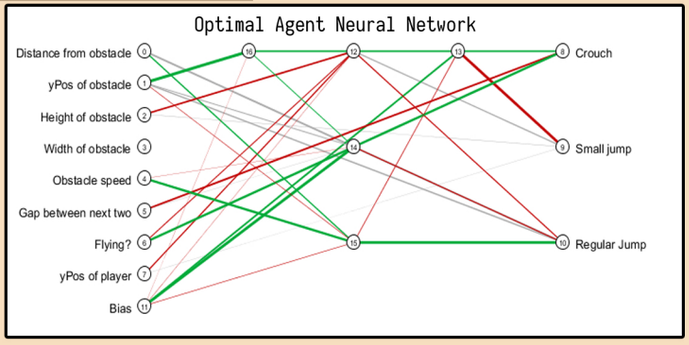
Another problem that NEAT resolves is the problem of competing conventions. Standard cross over can result in a loss of important nodes in the neural network. A simple example to demonstrate this is to do the following crossover:

```
  A B C
x C B A
```

This could result in genes ABA or CBC, both of which have lost potentially valuable nodes. To avoid such an issue, NEAT implements historical markings, or what they call innovations, for every gene. Simply put, NEAT labels every gene with a number, and any gene that matches the gene is given the same number. When performing a crossover, NEAT only will crossover genes with the same innovation. As a consequence, no genes can be overwritten by accident.

This combination of speciation and innovation markers, among other attributes, characterize the NEAT algorithm and allow it to properly generate an optimal agent.

Empirical Analysis



The above neural network is that of an optimal player. For the purposes of explanation, an optimal player is defined as one that continues to survive for at least five minutes after the game has reached its peak difficulty. In the visualization, a green connection has a positive weight, a red connection has a negative weight, a grey connection is a disabled connection, and the thickness of the line represents the magnitude of the weight.

Looking at the neural network, we can see that there is an immediate positive direct connection from the input "Flying?" to the output "Crouch". This matches what our expectations where if the next obstacle is flying, the player should always crouch, with no exceptions. We can also see a disabled connection from "Distance from Obstacle" (which was inverted before being passed to the agent, so the lower the distance, the higher the value) to "Regular Jump". It appears this was replaced with a "better" connection to node 15, which also accepts input from "Obstacle Speed" before passing its input to "Regular Jump". It is clear why this was the better connection. With just a connection from the distance to the obstacle to jumping, the player would only jump if the distance was close enough. However, the speed of the obstacle should also be taken into consideration. If the speed is high enough, the distance to the object becomes less relevant, and the player should jump in order to clear the obstacle in time. There is a plethora of connections to analyze, but given limited space, we will leave that analysis to the reader.

One thing to keep in mind, however, is that this is not necessarily the only "optimal" neural network. By running the program multiple times, we can find countless variations of neural networks that result in an optimal player. However, analyzing each of these should result in a few similar observations.

Kevin Cao and Zach Petroff (Team 27)