MazeSolver Phase III

Marcus Klein

McNeese State University

1. **Introduction**

As a quick recap, Phase I consisted of the introduction to OpenGL and C++, which were previously unknown, as well as the development of the graphics for the maze. Phase II included the research of various pathfinding algorithms, and the implementations of breadth-first search, depth-first search, and Dijkstra’s Algorithm, the last of which has not been successfully implemented, and temporarily delayed to allow for progress of the next phase.

Phase III of development of the MazeSolver program consisted of neural network and genetic algorithms research, as well as the initial stages of implementation of these concepts with the goal of training an artificially intelligent agent to solve the given maze. Aside from these main goals, some additional adjustments were made to the program to allow for better functionality and viewing of the solved maze.

1. **Overview of the Concepts Learned**
   1. **Neural Networks**

Artificial neural networks are processing algorithms that attempt to mimic the way a biological brain works, though simplified and on a much smaller scale. Just as biological neurons in the brain connect to other neurons, artificial neurons are connected together in some way to create the artificial neural network. As a result of this design, artificial neural networks have the ability to learn and generalize based on given information. They are commonly used for pattern recognition, image compression, stock market prediction, and video game AI, amongst other applications.

Neural networks generally consist of the following components:

* A directed graph known as the network topology with nodes representing each neuron
* A state variable, or **input**, associated with each node
* A **weight** associated with each connection between node
* A **bias** associated with each node
* A **transfer function** f for each node such that the state of the node is f(∑xiwi – β), where xi is the input from the connected node, wi is the weight connecting the nodes, and β is the bias of the node.

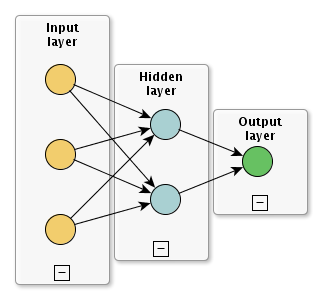


Figure 1: Basic Neural Network Topology

A feedforward neural network consists of an input layer of neurons, an output layer, and zero or more hidden layers, as shown by Figure 1. The network must be executed by initializing the states of its input nodes, then each node in the following layers is updated on intervals with a new state as computed by the transfer function. Once the network’s progress has reached a desired point, the states of the output nodes represent the network’s solution to the problem.

In order to learn, the neural network often requires a training stage, where the network topology and weights, biases, and transfer functions are optimized for the given problem based on training data. The topology and transfer functions are often constant, while the weights and biases are adjusted, commonly through use of the backpropagation algorithm. In the case of its application to the MazeSolver program, the goal is to train the weights and biases using a genetic algorithm

**2.2 Genetic Algorithms**

Genetic algorithms are essentially optimization algorithms loosely based on Darwinian evolution, where a population of genomes have their chromosomes “evolve” throughout several generations through natural selection, where the fittest genomes are typically selected to breed new genomes. These algorithms are especially good for solving problems about which little is known. Genetic algorithms are highly general, which makes them suitable for any search space, though they are often not an efficient solution. They can take a large amount of time to run and are therefore not suitable to real time use, but they do represent a powerful method to create high quality solutions to a given problem. Applications of genetic algorithms include solving the infamous Travelling Salesman Problem, financial decision making, and medical diagnoses.

Each chromosome must be encoded to represent a possible solution to the problem at hand, typically in binary, though not necessarily. They also must have a fitness score, indicating the accuracy of the solution represented by the chromosome. There are three essential operators of genetic algorithms:

* A method of **selection** where genomes are chosen from a population for breeding using the crossover operator. A common method, known as roulette wheel selection, is done based on a probability proportional to each chromosome’s fitness. (See Figure 2)

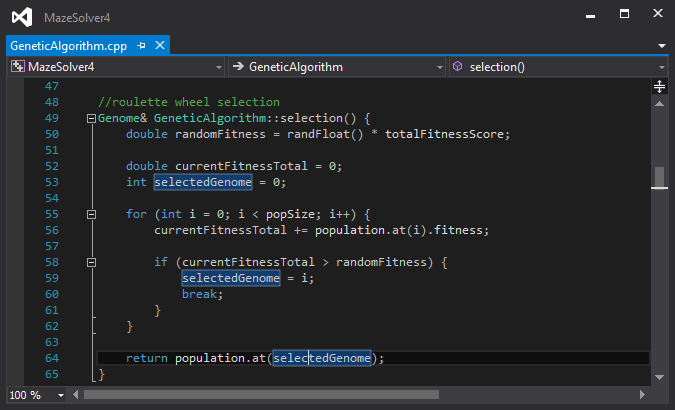
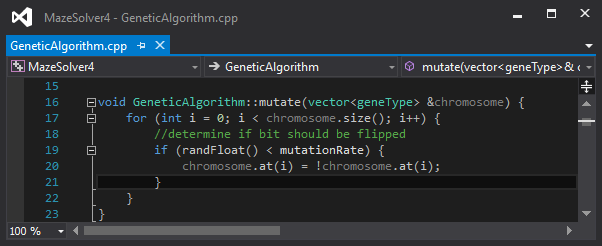


Figure 2: Roulette Wheel Selection

* The **mutation** operator which maintains genetic diversity from one generation of a population of chromosomes to the next. The mutation rate is often set to 0.1%, but similarly to the crossover rate, can vary depending on the problem. (See Figure 3)

Figure 3: Mutation



* The **crossover** operator where the chromosomes of one generation are varied in the next generation, analogous to biological reproduction. Experimentation has shown that a good crossover rate is typically around 70%, but can vary depending on the problem domain. (See Figure 4)

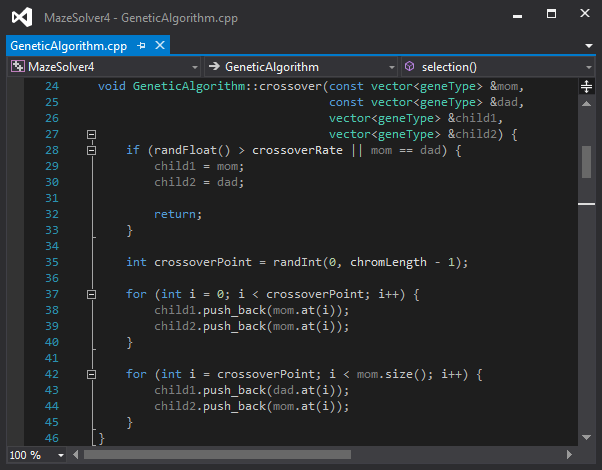


Figure 4: Crossover

The genetic algorithm itself is as follows:

CreateInitialPopulation()

EvaluatePopulation()

**while** solution has not been found **do**

CreateNewPopulation()

EvaluatePopulation()

Genome ParentOne = SelectParents()

Genome ParentTwo = SelectParents()

Genome ChildOne, ChildTwo

ChildOne = Crossover(ParentOne, ParentTwo, ChildOne, ChildTwo)

Mutate(ChildOne)

Mutate(ChildTwo)

**end while**

**3 Development Progress**

**3.1 Genetic Algorithm**

While the intent is to develop a neural network to solve the maze with a genetic algorithm optimizing the weights, a working genetic algorithm was created, implemented as its own class, using chromosomes encoded to represent a path in the maze. This enabled experimentation and experience to be gained with the use of genetic algorithm. The genetic algorithm usually, but not always, solves the given maze in the first generation of genomes, and each genome’s path is represented graphically in a random color.

**3.2 Maze Reset**

A function has been implemented to reset the maze to its original state.

**3.3 Maze Randomization**

A function has been implemented to randomize the maze.

**4 Phase IV Plans**

**4.1 Neural Network**

The goal is to implement a working neural network with the genetic algorithm selecting the weights to be used in training the neural network.

**4.2 Genetic Algorithm**

The genetic algorithm will be altered to use the neural network weights as chromosomes and breed them to find efficient solutions.

**4.3 Loose ends**

In the case of a working solution being created in a timely manner using a combination of neural networks and genetic algorithms, various other features of the program will be fixed and/or added, such as Dijkstra’s Algorithm and a display of the stats of the artificial intelligence agent, such as the fitness and current generation number.