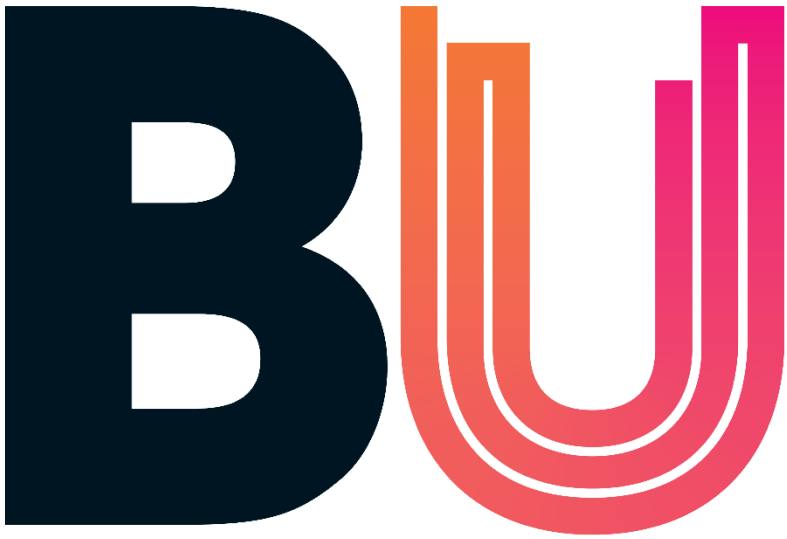

Bournemouth University

Jack M Gilbride

AI Game Programming Assignment 1

Introduction

A* Pathfinding Algorithm

The A* search algorithm is a “graph traversal” and “path search algorithm” which is commonly used in games and travel routing systems e.g. sat navs. While this algorithm will always find the optimal solution it is also very resource intensive as each individual node has to be stored in memory.

The A* algorithm works out the most efficient path from a starting node to a goal node, what really separates A* from other graph traversal algorithms is the use of a cost function to assign the most efficient parents to each individual node. The generic cost function for A* is:

$$f(n)=g(n)+h(n)$$

where

f(n) = the total cost of the path through node *n*
g(n) = the total cost to reach *n*
h(n) = the **estimated** cost from *n* to the goal.

The A* algorithm begin by creating two lists, the open list and the closed list then adding the starting node to the open list. After that move to the node from the open list with the lowest cost, if this is the goal node the algorithm stops here. If not, the current node is moved to the closed list and all adjacent nodes examined and if they aren’t already in the open list a cost is calculated for them and they are added. This process is repeated until the goal node is reached.

Genetic Algorithm

A genetic algorithm is an algorithm modeled on Charles Darwin’s theory of evolution, it attempts to replicate the process of natural selection or “survival of the fittest”, allowing the most successful individuals to reproduce. The concept was first introduced in 1960 by the American scientist and Professor of Psychology John Holland, this concept was later built upon by his student David Goldberg in 1989.

The algorithm starts by creating an initial population and then calculating their fitness, after that there is a new population generated based upon the previous generation. This is most commonly done using a “Roulette Wheel Method” to crossover two chromosomes and basic gene mutation, by selecting two chromosomes based on their fitness and a little bit of randomness you can create two new chromosomes using characteristics of both parents. You

can then occasionally mutate some of the genes and repeat the process until the problem is solved.

Genetic algorithms are almost a middle ground between the extreme training times of neural networks and the extremely short pathing time of A*. Depending on how efficient you want your path to be it can take seconds, minutes or hours to generate a desirable chromosome.

Artificial Neural Network

Artificial Neural Networks are virtual systems based loosely on the physical neural networks that make up real brains, these systems are designed to learn to perform designated tasks such as recognising certain images. For example, Google are currently training an ANN to analyse footage from surveillance cameras and locate snow leopards automatically allowing for more time to be spent actually working to protect them as well as reducing the number of cases of mistaken identity or cases where the animals are missed altogether.

ANNs are made up of artificial neurons which can transmit signals to other neurons, the signal is normally represented by a real number which is then compared to the neuron's threshold and it fires accordingly.

A neural network can have a varying amount of layers and range from a collection of 5 neurons to hundreds or even thousands, this allows for extremely fine tuned values and very accurate results.

One big issue with using an ANN in this limited timeframe is the amount of time needed to effectively train a neural network, depending on the complexity this can take multiple weeks and on a tight schedule this can become a massive issue.

What I Used and Why

When choosing between the use of a genetic algorithm or an artificial neural network I ended up settling on the use of a genetic algorithm, this is because for this task in particular the use of an ANN would be complete overkill. It would be extremely hard to train a network to react logically to a maze of any size and would likely be impossible to create an ANN capable of solving the larger mazes within the given timeframe.

Another reason I chose not to use an ANN is that the only benefit for using either an ANN or a GA over A* pathfinding is that it would give a more organic feel to the pathing. A* always finds the optimal route and while it could be modified to create a more realistic path this would stray away from the core algorithm. When trying to use an ANN to solve the maze if you somehow managed to create the network and train it effectively in the given timeframe you would likely

end up with something extremely similar to A* which took much longer to make and provides arguably worse results.

When you look at GA compared to A* it will most often give a much more natural feeling path, since in real life animals and people don't just instantly calculate the most efficient path and follow it perfectly they make mistakes and take a little longer than is necessary in certain situations. This provides an actual noticable difference to the A* pathfinding and allows for more interesting paths to be generated. This is the reason I chose to compare A* and Genetic Algorithms.

My Implementation

Next I will talk about my language and engine choice for this assignment briefly, I chose to use the Unity engine and C# scripting to complete this assignment. I did this because I anticipated having the Unity development environment and access to Unity GUI elements as well as sprite renderers etc. would be extremely useful. This allowed me to spend more time working on perfecting my algorithms and less time doing back-end work, meaning I was able to add unique features such as an integrated heat map for my genetic algorithm to make it much more effective.

A* pathfinding

When implementing the A* algorithm I chose to use handle the main part of the algorithm through a manager class and use a struct to hold the individual node data, I also made use of a map class that allowed me to import maps at runtime and using the map and manager together I could dynamically draw the map every time a new one was loaded.

During the process of fine tuning my parent cost function I decided to edit the formula a little, this is because having the distance to the goal and the distance to the starting point both impact the cost the same amount made my path lean towards the goal point too much, meaning that the optimal path wasn't always found. I fixed this by making the distance to the goal only impact the cost half as much as the distance to the start point, cost function can be seen below:

$$f(n)=s(n)+g(n)$$

where

f(n) = the total cost of the path through node *n*
s(n) = the distance from *n* to the starting point
g(n) = the direct distance from *n* to the goal divided by two

By doing this my algorithm went from finding close to the optimal path to finding the optimal path every time (at least in my testing).

During testing I also found that I could easily change my A* from 4 directional movement to 8 directional movement by changing the “moveDistance” variable from 1.2f to 2f, this variable is used when calculating valid moves for the final path. After discovering this I added a button to the UI allowing you to toggle between the two movement settings, I did this because I thought it was interesting to see the difference in the final paths the two settings generated and it was a unique feature I didn’t see in anyone else’s project.

Genetic Algorithm

After creating my A* I had a good idea of how to implement the genetic algorithm, I started by using the same map class from my A* project and creating a new manager that included the map importing and drawing features. This gave me a really solid base to work from. I then focused on creating the chromosomes, which I made in their own class which included a fitness variable and a list of integers called “bits”. After that I got to work making the actual genetic algorithm, I defined the generic variables like population size, mutation chance etc. and created a basic animal class that would be used to calculate the fitness of each chromosome. I did this by creating a move function on the animal and calling it with each element of the chromosome’s bits list and then calculating the fitness based on the distance to the goal point.

After generating all of the fitness scores for the entire population I used the roulette wheel method to select a pair of chromosomes, these were then used as the parents to create two new chromosomes with a crossover function. Then both of these offspring chromosomes were given the chance to mutate, based on the mutation chance and added to the new generation.

Once I managed to get all of that working I could easily solve the first two test maps we were given, they were relatively small and required little more than a basic algorithm to find the most efficient route possible. The third map we were given proved to be much more difficult however. I tried multiple ways of getting around this such as increasing the mutation chance as the generations went on or rewarding the animal for getting far away from the start point.

After all of these methods failed I decided to strip them all out and go back to the basic algorithm since my fitness function and algorithm in general had become very messy and I felt like I was making no progress. I then decided to try a new approach, I added another hidden layer to my map which was used to store the amount of chromosomes that had finished on each square of the maze. For example, if five animals finished on the square (4, 8) then the value of that square would become 5. I then added this value to the distance from the goal in the fitness function and my genetic algorithm instantly became able to solve the third map.

This method worked incredibly well because it heavily punishes the chromosomes for stagnating, if they stay in the same place for too long without finishing the maze their fitness becomes so low that the first species to successfully explore a new area ends up producing a

huge amount of offspring pushing the species forward. This system of rewarding exploration and punishing stagnation allowed my algorithm to solve much more complex mazes and become much more effective.

References

Anon., 2019. *Heat map* [online]. En.wikipedia.org. Available from: https://en.wikipedia.org/wiki/Heat_map [Accessed 13 Dec 2019].

Anon., 2019. *Genetic algorithm* [online]. En.wikipedia.org. Available from: https://en.wikipedia.org/wiki/Genetic_algorithm [Accessed 13 Dec 2019].

Anon., 2019. *Artificial neural network* [online]. En.wikipedia.org. Available from: https://en.wikipedia.org/wiki/Artificial_neural_network [Accessed 13 Dec 2019].

Anon., 2019. *Neural network* [online]. En.wikipedia.org. Available from: https://en.wikipedia.org/wiki/Neural_network [Accessed 13 Dec 2019].

Anon., 2019. *A* Search | Brilliant Math & Science Wiki* [online]. Brilliant.org. Available from: https://brilliant.org/wiki/a-star-search/ [Accessed 13 Dec 2019].

Anon., 2019. *YouTube* [online]. Youtube.com. Available from: https://www.youtube.com/watch?v=VnwjxityDLQ [Accessed 13 Dec 2019].

Anon., 2019. *YouTube* [online]. Youtube.com. Available from: https://www.youtube.com/watch?v=qiKW1qX97qA [Accessed 13 Dec 2019].

Anon., 2019. *YouTube* [online]. Youtube.com. Available from: https://www.youtube.com/watch?v=rGWBo0JGf50 [Accessed 13 Dec 2019].

Bling, S., 2019. *MarI/O* [online]. Youtube.com. Available from: https://www.youtube.com/watch?v=qv6UVOQ0F44 [Accessed 13 Dec 2019].