Genetic Algorithm:

A Genetic Algorithm (GA) is a type of program that uses an element of randomness to generate a species of genes for a set of chromosomes in the hope of getting close to some kind of end goal. Through “breeding”, a term used to describe crossing different chromosome’s genes, the program will attempt to create a better path to the end goal from the better of the previous paths.

However, whilst this would make for a useful program for generating a path to an end goal faster than ANN’s (discussed in the next topic), or an A\* algorithm, instead the GA attempts to act in the same way as evolution. This means that for the most part the algorithm will be trying to breed the best parents together to create better offspring, that then breed together to create even better offspring. Despite this, GA will also attempt to work in the same way as natural evolution through the way of mutation, meaning that every so often one of the chromosomes in the gene pool will have a random difference thrown in that has the possibility to completely change the outcome of that chromosome. Through this and acknowledging the fact that sometimes the most desirable parents don’t always breed together, supposing they still do, they aren’t guaranteed to pass on the most desirable genes, we can tell that GA is a model based on a real-life concept in order to showcase evolution in a computing environment.

For the purpose of making a GA run through a map and reaching the end goal from the start goal, GA theoretically is great at finding a solution when there are multiple possible solutions and the map size is relatively small (Less than 50 tiles). However, as the map size would increase so too would the runtime and resources used which would make running GA on larger map sizes incredibly slow and taxing on the computer. GA will also struggle to find a solution when there is only one possible path to the end of the maze as it will constantly be turning around on itself to begin with before eventually evolving enough to start to head towards the end goal.

Due to this GA is best run on smaller map sizes where there are multiple paths to the end goal, however GAs “can be designed to meet any optimization goal, and is often a way to find an approximation to an optimal solution, when the path to finding that solution is unclear” (Beasley et al. 1993). Meaning that GAs can be used in order to find out what an optimal solution might look like if you’re unsure about how to start looking for it.

Artificial Neural Network:

An Artificial Neural Network (ANN) is a program that runs on the model of a brain in order to think and solve problems like a human through recognition of what it has been taught based off of training and watching correct solutions before trying to accomplish that goal itself.

Neural Networks run off Neurons that recognise patterns based off of input data, it will add this input data along with some weights assigned to it and if the total of that data is more than its assigned percentage value then it will trigger an output. Basically in simpler terms, the AI recognises patterns in the inputs it receives and once it thinks it has enough data to correctly catalogue them it will start to give its own outputs, much like a child’s brain attempting to learn something new.

For the purpose of training an Artificial Neural Network how to navigate through a maze the Neural Network would do a fantastic job after receiving enough data and it may even get the shortest possible path faster than the A\* algorithm which is great news for when it comes to relatively big mazes ( > 100 tiles total).

However here comes the downsides, unfortunately due to the way ANNs work, they need to receive training data and analyse it before they can start to output their own anticipated paths, and whilst initially generating their outputs they can be prone to getting things wrong. This is where the programmer creating the ANN needs to finetune the weights given to the ANN to create the least possible error for the map it’s attempting to find the shortest path through which can take up to a few hours to do. Unfortunately for us this isn’t even considered the tedious part, for every new map you run the ANN on, the weights will need to be tweaked all over again for the new map.

Doing this process multiple times over can be very tedious, boring and for a good neural network could take days upon days to get right, never mind perfect. Some of the best Artificial Neural Networks have been trained for years to be as good as they are, such as the Neural Network that beat the computer chess championship against a then-current undefeated champion. (pete) 2019)

Due to the fact that Neural Networks can take so long to program, then train and finally be good at its intended function, and the fact the Neural Network has to be trained off of something that can already do that function, the question then begs why would anyone spend their time using up resources training a neural network to find the shortest possible path to the end of a maze when in order to train it they must already have a program to do that or they’re training it by hand which isn’t very effective.

Comparison: GA or ANN

If I had to compare these two methods of pathfinding for a maze together to then chose one and compare it to the A\* algorithm, I’d say in my personal opinion GA is a better choice, but not a good one. ANN is by all intents and purposes way too big for the scope of the project and quite frankly would eat up resources and time that could be spent researching into even more methods that are possibly more efficient. We already know that an ANN would have to be trained off either human success or a program that can already do the job, and if we’re trying to create a program that needs the lowest amount of maintenance then an Artificial Neural Network just isn’t going to be an efficient use of time.

However this doesn’t make GA much better, GA doesn’t take anywhere near as much time to program nor does it require constant maintenance to make sure it’s performing as expected after the program has been made, however the potential for it to get one of the shortest paths in a maze is purely down to a random number generator creating the exact right path before it even needs to evolve anything, completely removing the need for you to have an actual Genetic Algorithm in the first place.

Overall GA is a much better choice than ANN for the current task at hand, however it suffers drawbacks that personally I can’t see it making up for in comparison to the A\* algorithm, but we’ll get to see that in our tests of the next task.

Analysis:

Unfortunately, due to GA having lots of unforeseen bugs during testing and the time left until the project’s deadline I had no choice but to leave my Genetic Algorithm unfinished. This is frustrating as it means the only data I can compare my A\* Algorithm to is research I have found that doesn’t document the same things my testing has documented.

Due to this I will analyse my A\* Algorithm and compare it to what I believe the Genetic Algorithm results would have looked like.

The A\* algorithm is without a doubt an amazing method of pathfinding your way to an end goal in a maze, the speed at which the algorithm fully analyses all the nodes it needs to and then returns a path is extraordinary and the amount of memory the algorithm uses up whilst running is next to completely unnoticeable (20Mb).

The A\* algorithm always reached the end goal in one of the shortest possible paths regardless of the difficulty of the map, and the speed at which it accomplished this was still on average less than 2 seconds for a map of size 100.

The expected outcome for Genetic Algorithm is that the memory usage would have been higher due to the fact that you have to choose a population size and a path size for the GA to generate, for every individual in the population size a path of value path size would have to be generated. This means if you wanted to go for a realistic expectation of finishing TestMazeC then you would need a path size of 60 tiles and a minimum population size of 4. This in all means you would have 240 bits of memory that need to be memorised for the duration of the loop which would without a doubt use up a noticeable chunk of memory (100Mb Is an estimate), and then for every iteration of the loop you replace those 240 bits with new ones meaning effectively every iteration you’re dealing with 480 bits of data.

This outside the obvious method of how GA works being based off the first paths generated are entirely random, so sometimes GA would run incredibly fast and be done in less than a second, other times GA would take at least a few seconds before it is done. It’s a safe guess to say the Random Number Generator wouldn’t give a good path every single time right at the start, in fact it would probably only generate something nice like that quite rarely. This in turn means that the average speed GA takes to reach the end position would be longer than the A\* algorithm’s average speed.

Conclusion:

In conclusion I would say the A\* algorithm is still currently the best of the options given for finding not only the shortest possible path, but also finding it quicker than anything else without using intense amounts of resources or time to program. The GA has a lot of things it can do and a lot of possibilities for other types of problem solving however for this project it just doesn’t do enough against the A\* Algorithm for it to be justifiable using it. Finally, an ANN would just take too long to program, train and also too many resources for any small company or individual to warrant programming. And even a bigger company would be better off not using an Artificial Neural Network for this kind of small project, however if the project was something with more logical reasoning and pattern recognition then the benefits would far outweigh the costs.

References:

JONASSON, A. and WESTERLIND, S., 2016. *Genetic algorithms in mazes* [online]. Diva-portal.se. Available from: http://www.diva-portal.se/smash/get/diva2:927325/FULLTEXT01.pdf [Accessed 13 Dec 2019].

Beasley, D., Bull, D. and Martin, R., 1993. *An Overview of Genetic Algorithms : Part 1, Fundamentals* [online]. ebook. Cardiff: University of Cardiff. Available from: http://mat.uab.cat/~alseda/MasterOpt/Beasley93GA1.pdf [Accessed 13 Dec 2019].

(pete), P., 2019. *Lc0 Wins Computer Chess Championship, Makes History* [online]. Chess.com. Available from: https://www.chess.com/news/view/lc0-wins-computer-chess-championship-makes-history [Accessed 13 Dec 2019].

Dacombe, J., 2017. *An introduction to Artificial Neural Networks (with example)* [online]. Medium. Available from: https://medium.com/@jamesdacombe/an-introduction-to-artificial-neural-networks-with-example-ad459bb6941b [Accessed 13 Dec 2019].

Graham, R., McCabe, H. and Sheridan, S., n.d. *Pathfinding in Computer Games* [online]. ebook. Dublin: Institute of Technology, Blanchardstown. Available from: http://neuro.bstu.by/ai/To-dom/My\_research/Papers-2.0/Path-planning/pathgraham.pdf [Accessed 13 Dec 2019].

Russell, S. and Norvig, P., 1995. *Artificial Intelligence A Modern Approach* [online]. ebook. New Jersey: Alan Apt. Available from: https://www.cin.ufpe.br/~tfl2/artificial-intelligence-modern-approach.9780131038059.25368.pdf [Accessed 13 Dec 2019].

Truong, T., 2015. *Solving A 2D Maze Game Using a Genetic Algorithm and A\* Search - Part 2* [online]. Tony Truong. Available from: https://tonytruong.net/solving-a-2d-maze-game-using-a-genetic-algorithm-and-a-search-part-2/ [Accessed 13 Dec 2019].