AI ASSIGNMENT 2

Research, design, and implementation of pathfinding algorithms

Daniel Holmes (s5304084)

Contents

Section 1: A* Algorithm

A* pathfinding is an algorithm that can determine a path from one position to another as long as both the start and end position are known. It uses 2 primary equations in order to calculate its desired path, the first being a distance calculation equation. The equation can vary depending on what method you use to calculate distance, however within this implementation, the Euclidean distance equation is used:

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_1 - p_1)^2}$$

Both p and q have 2 elements, making the equation work well with a grid maze and easy to implement.

A* also makes use of a node system to represent the positions around the agent that can be moved to, as such the nodes within this implementation are all derived from a node class, making it easy for new nodes to be created. The second equation is that which weights the nodes around the agent to determine which node would be best for the agent to move into next, and is shown here:

$$F = G + H$$

Where:

- F = Score assigned to a node
- G = The distance between the node the agent is currently occupying and the node it wants to move to
- H = The distance between the node the agent could move to and the finish

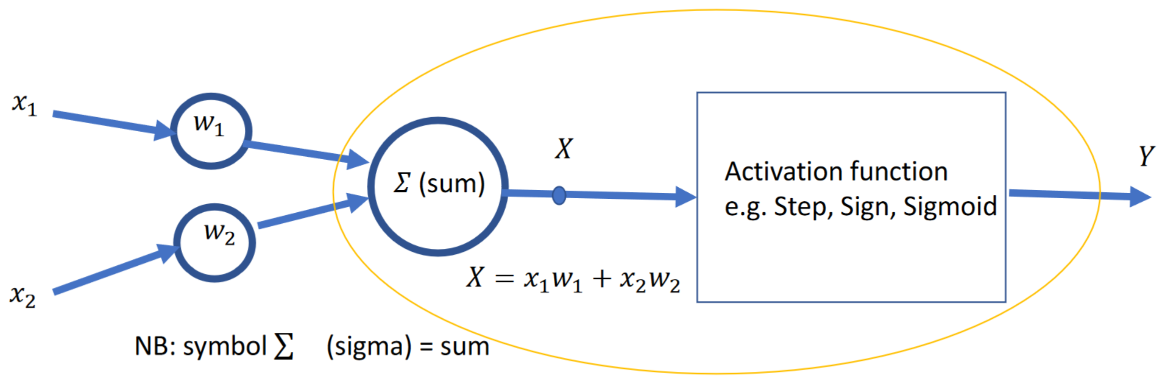
This equation works with the distance equation and is what provides the functionality to the A* algorithm. For this implementation, this calculation is completed for a node after which that node is placed in a priority array which stores the score of each node in order of priority, making it easy for the agent to determine which node to move to. 2 arrays also exist to sort nodes into open and closed categories, with open nodes being nodes the agent can move to and closed being the opposite.

The complete A* algorithm is contained within one function, making it accessible by the rest of the program. The program won't exit this function until either a route or no route is found, with the function returning the result. The algorithm being contained within a function also means that as many values for it to work can be passed in, reducing the need for global variables. The rest of the implementation pertains to the reading of the maze file and how the information is displayed to the user, so it doesn't have much impact on how the A* algorithm is implemented

Section 2: Non-Deterministic Algorithm Research

2.A: Artificial Neural Networks (ANN's)

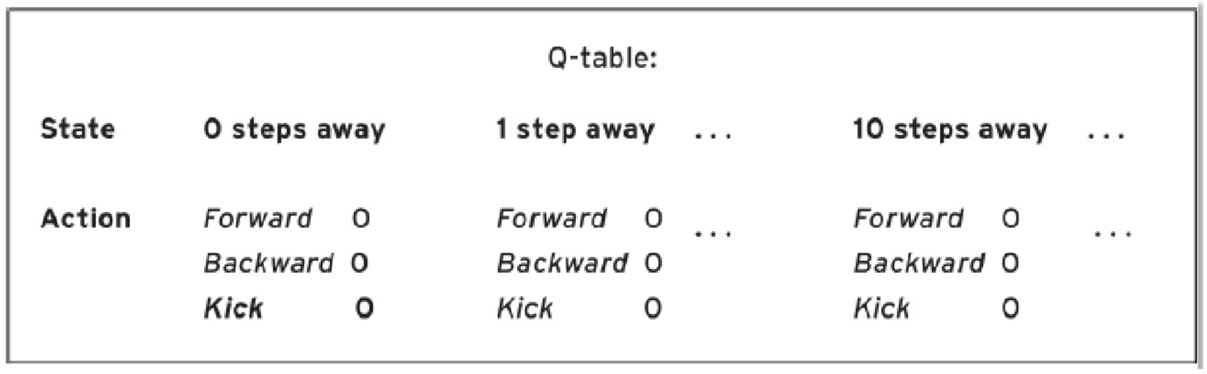
Through my research I have found that Artificial Neural Networks are a type of AI designed to be structured to imitate how a human brain function, with each "Neuron" within a network being modelled after biological neurons. The model of an artificial neuron is shown here:


x_1
w_1
x_2
w_2
Σ (sum)
X
Activation function e.g. Step, Sign, Sigmoid
Y
$X = x_1w_1 + x_2w_2$
NB: symbol $\sum$ (sigma) = sum

An ANN's ability to complete its designated task depends highly upon 3 factors, the first being how well trained the ANN is, requiring high quality training data to use in order to learn how to perform its task. The second being the activation function the network is using, as this function will calculate what signal a neuron will send to the neuron's it's connected to (the output of a neuron being Y in the diagram above). The third factor is the number of layers a network has, the more layers there are the network the quicker the network will learn and complete a task

2.B: Reinforcement Learning (RL)

After researching, I found that Reinforcement learning is a method for training AI that imitates positive reinforcement within living creatures. Reinforcement learning teaches the AI by giving it positive rewards for performing a specific action at the correct point in the agent's execution of its task, teaching the agent to use set actions at set points within its task completion. RL uses of a Q-table containing a list of actions the agent can take at each state it finds itself in during task completion, with the state being dictated based off of an estimate the agent makes of how close it is to task completion, e.g. an agent being in state 10 would mean it is an estimated 10 steps away from the finish point it's trying to pathfind towards. When the agent performs the correct action, a reward is given and the Q-table is adjusted to make it more likely for the agent to perform that action in that state when reperforming the task. Q-Table Example:



Q-table:

State	0 steps away		1 step away		...	10 steps away		...
Action	*Forward*	0	*Forward*	0	...	*Forward*	0	...
	Backward	0	*Backward*	0		*Backward*	0	
	Kick	0	*Kick*	0		*Kick*	0	

2.C: Genetic Algorithms (GA)

Through researching, I discovered that Genetic Algorithms are a method of creating an AI that is suitable for its task by generating and "breeding" a population of "chromosomes" over multiple generations until the optimal "offspring" is created and can complete the task. This method of AI creation and training is modelled off of how human chromosomes of children are a mix of the chromosomes of the parents, sometimes with mutations that alter the chromosomes. Within the context of pathfinding, a chromosome will consist of a list of binary numbers that represent the moves that a chromosome will take within a maze, with a move index table determining said moves.

Example Chromosome:

0110011001000100

Movement Index:

	Direction
00	Up
01	Right
10	Down
11	Left

The initial population of chromosomes are randomly generated, and the program continues to breed new generations of chromosomes until the most effective chromosome is found. There is an equation used to optimise the creation of the most effective chromosome, with said equation determining which chromosomes out of the current population breed with which, producing the most effective offspring by determining which of the chromosomes are the most fit for purpose. This is the “Fitness” function and is created during implementation to best suit the required task. Example:

$$Fitness = \frac{1}{Dx + Dy + 1}$$

Where Dx = horizontal grid offset from finish, Dy = vertical grid offset from finish.

2.D: Evaluation of methods and conclusion

ANN

For the task of pathfinding, I don’t believe ANN to be the most suitable option for creating a pathfinding solution, with the two major issues involving the activation function and the training data. The former causes problems as there are a multitude of activation functions to choose from and no way of accurately determining which to use, meaning that a large amount of testing would need to be done in order to figure out which to choose. Even then, the latter issue of training data may cause your testing to be pointless as the data you are using just isn’t good enough to effectively train the network, and also isn’t something that can feasibly be found in the context of the assignment itself.

RL

Unlike ANN, RL appears to be must better suited for use in the creation of a pathfinding agent as the q-table and states system it uses lends itself very well to determining which direction to move in. The system of determining a path would also work regardless of the layout of the maze, making it a versatile method for pathfinding.

GA

Much like RL, the nature of how a GA functions lends itself very well to being applied for pathfinding, with the chromosome digits allowing for easy encoding of what moves the agent should take. The length of the chromosome can also be adjusted in accordance with the size of the maze to make sure the agent takes the most efficient path possible whilst being able to reach any point within the maze.

Conclusion

From my research, I have decided to choose Genetic Algorithms as my AI pathfinding method, due to its inherent suitability for the task at hand with the way the chromosomes are structured, as well as for the fact that I feel as though it would be the easiest to implement out of the three options discussed above, due to it having somewhat of a similarity to the A* method.

Section 3: Genetic Algorithm (GA) implementation

For the creation of the chromosomes themselves, much the A* implementation, I created a class that contains all the necessary information relating to a chromosome, allowing for easy creation of new chromosomes during population generation or at other points in the program. The creation of the initial population itself is contained within a function, with the generated chromosomes being stored within a global object array, allowing the population to be accessed at any point. The creation process itself involves a loop that creates and stores a new chromosome until the full population is made.

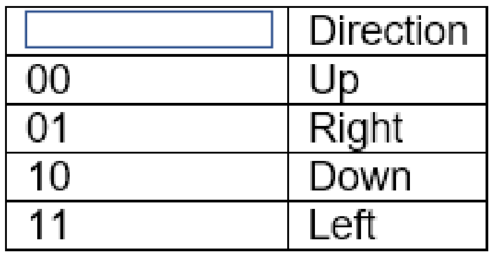
The crossover and mutation rates are set at 0.7 and 0.001 respectively, with the implemented fitness function being as follows:

$$Fitness = \frac{1}{Dx + Dy + 1}$$

Where Dx = horizontal grid offset from finish, Dy = vertical grid offset from finish.

This implementation also makes use of the same Euclidian distance formula as A* to calculate the distance between the chromosome and the finish.

The movement index for the chromosomes is as follows:



	Direction
00	Up
01	Right
10	Down
11	Left

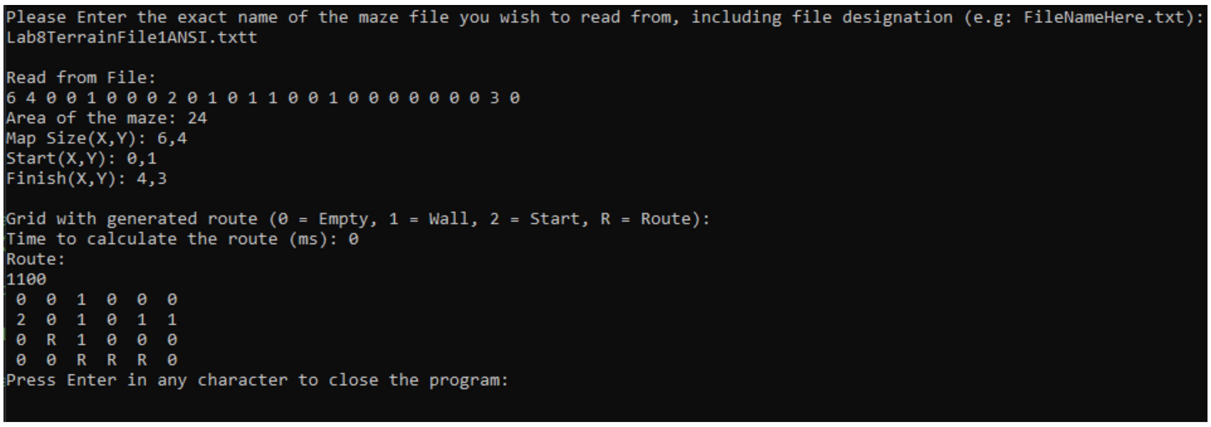
Each time a move is made, 2 checks occur to ensure the chromosome isn't moving either into a wall or outside the maze, and if so, the chromosome will remain in its current position for that move.

The implementation of determining pairs for crossover occurs in the same function as where each chromosome's fitness is calculated, with the determined pairs then being passed into the crossover function for breeding, then following that on into the mutation function to see if any mutation occurs in the newly created population. This process repeats until a successful chromosome is created, in which the program ends and displays the successful chromosome, the time it took to create on and the number of episodes that have occurred.

Section 4: Performance Evaluation and Experiments

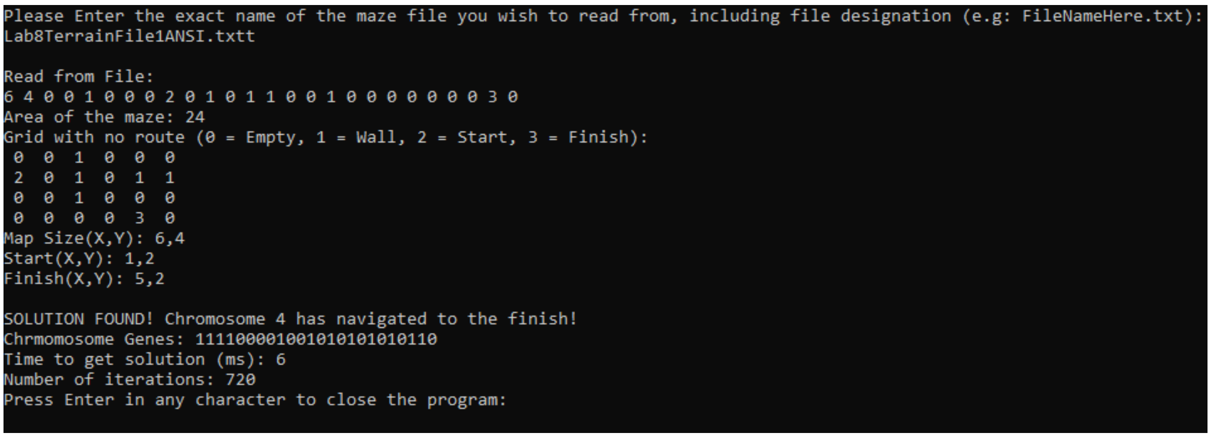
I designed 3 criteria to determine how effective the performance is of each of the implemented methods, with these criteria being as follows: the method's ability to function and produce an accurate pathfinding result, how quickly it was able to produce said result and the method's ability to support a maze of any layout and complexity. As well as this, I have conducted a general evaluation into how well implemented each of the 2 methods are.

Firstly, the evaluation of whether each method functions and returns an accurate result, of which the A* implementation is able to handle easily, even displaying the route that the agent took through the maze upon completion:



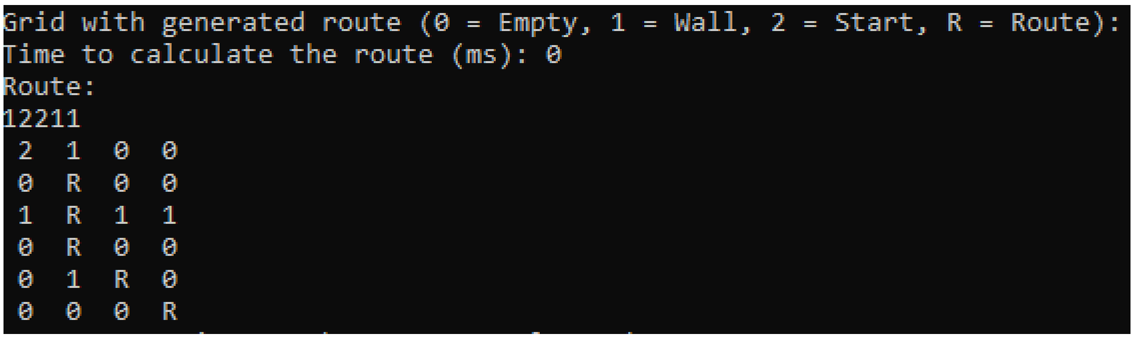
```
Please Enter the exact name of the maze file you wish to read from, including file designation (e.g: FileNameHere.txt):
Lab8TerrainFile1ANSI.txtt

Read from File:
6 4 0 0 1 0 0 0 2 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 3 0
Area of the maze: 24
Map Size(X,Y): 6,4
Start(X,Y): 0,1
Finish(X,Y): 4,3

Grid with generated route (0 = Empty, 1 = Wall, 2 = Start, R = Route):
Time to calculate the route (ms): 0
Route:
1100
 0  0  1  0  0  0
 2  0  1  0  1  1
 0  R  1  0  0  0
 0  0  R  R  R  0
Press Enter in any character to close the program:
```

In comparison, the GA implementation runs without issue and is capable of producing a result, but that result is unfortunately often inaccurate. As shown below, the program produces a result, however when comparing the moves the chromosome makes against the maze layout, the result produced is inaccurate:



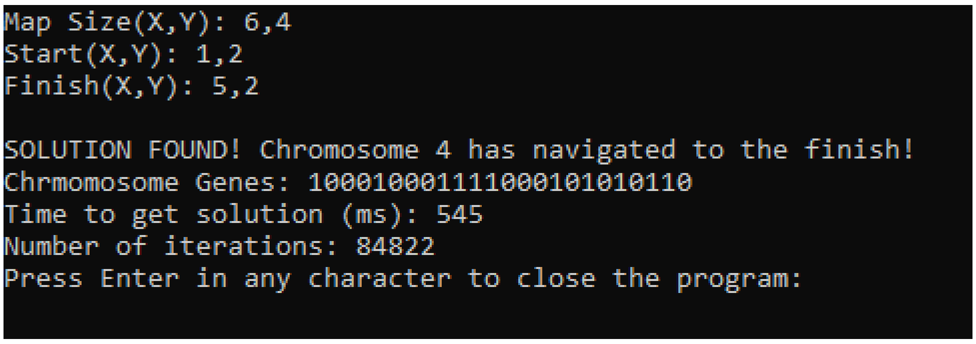
```
Please Enter the exact name of the maze file you wish to read from, including file designation (e.g: FileNameHere.txt):
Lab8TerrainFile1ANSI.txtt

Read from File:
6 4 0 0 1 0 0 0 2 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 3 0
Area of the maze: 24
Grid with no route (0 = Empty, 1 = Wall, 2 = Start, 3 = Finish):
 0  0  1  0  0  0
 2  0  1  0  1  1
 0  0  1  0  0  0
 0  0  0  0  3  0
Map Size(X,Y): 6,4
Start(X,Y): 1,2
Finish(X,Y): 5,2

SOLUTION FOUND! Chromosome 4 has navigated to the finish!
Chrmomosome Genes: 1111000010010101010110
Time to get solution (ms): 6
Number of iterations: 720
Press Enter in any character to close the program:
```

Secondly, the time it takes for each implementation to produce a result, within which A* has no issues as it often doesn't take move than a second to produce a result. The time taken does increase in conjunction with the size and complexity of the maze, however it is never to an unreasonable point:



```
Grid with generated route (0 = Empty, 1 = Wall, 2 = Start, R = Route):
Time to calculate the route (ms): 0
Route:
12211
 2  1  0  0
 0  R  0  0
 1  R  1  1
 0  R  0  0
 0  1  R  0
 0  0  0  R
```

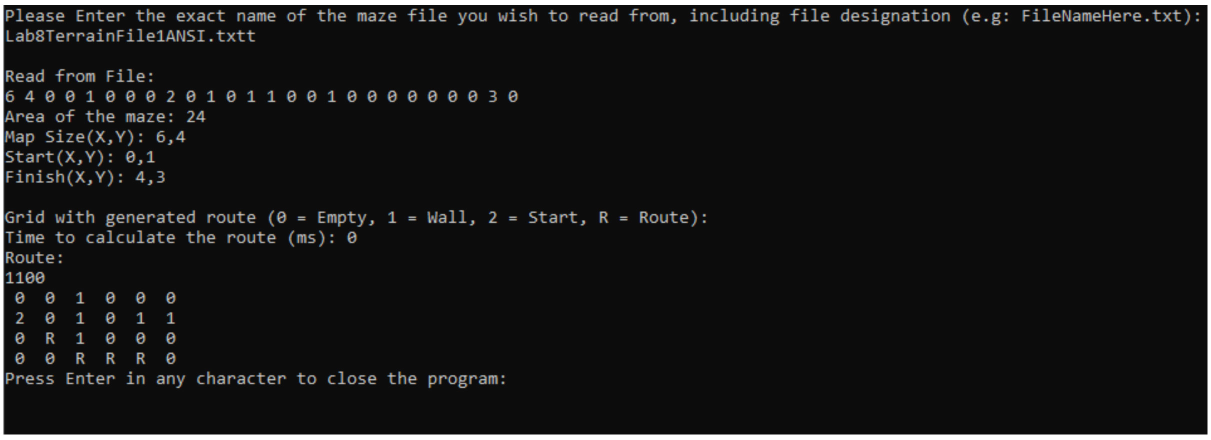
Comparing that to the GA implementation, this method also has no issue with the time it takes to produce a result, and although it on average takes longer to produce a result in comparison to A*, it is no more that a few milliseconds. GA also displayed the total number of episodes/iterations it took in order to produce a result (it is just, once again, unfortunate that the result produced is often inaccurate):



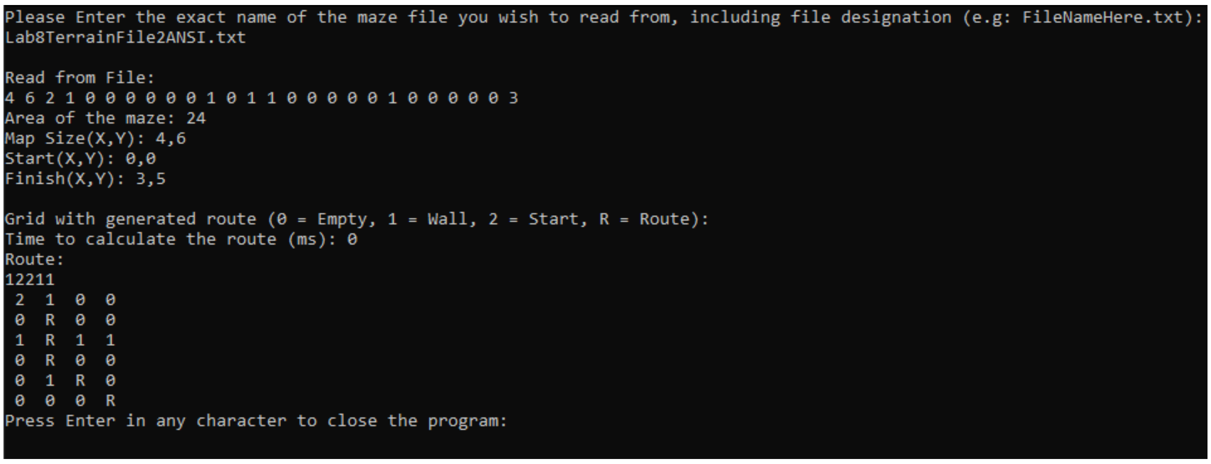
```
Map Size(X,Y): 6,4
Start(X,Y): 1,2
Finish(X,Y): 5,2

SOLUTION FOUND! Chromosome 4 has navigated to the finish!
Chrmomosome Genes: 100010001111000101010110
Time to get solution (ms): 545
Number of iterations: 84822
Press Enter in any character to close the program:
```

Finally, each method's ability to accommodate for any maze layout and complexity. There is little comparison needed when it comes to this factor, as each of the methods are perfectly capable of handling a maze of any layout. Each method working with different maze layouts are shown below:

A*:

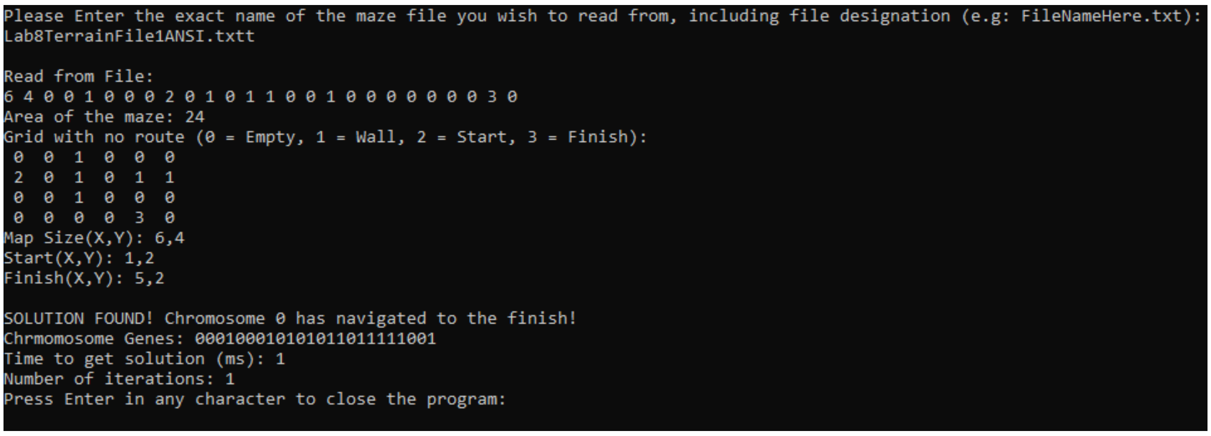


```
Please Enter the exact name of the maze file you wish to read from, including file designation (e.g: FileNameHere.txt):
Lab8TerrainFile1ANSI.txtt

Read from File:
6 4 0 0 1 0 0 0 2 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 3 0
Area of the maze: 24
Map Size(X,Y): 6,4
Start(X,Y): 0,1
Finish(X,Y): 4,3

Grid with generated route (0 = Empty, 1 = Wall, 2 = Start, R = Route):
Time to calculate the route (ms): 0
Route:
1100
 0  0  1  0  0  0
 2  0  1  0  1  1
 0  R  1  0  0  0
 0  0  R  R  R  0
Press Enter in any character to close the program:
```

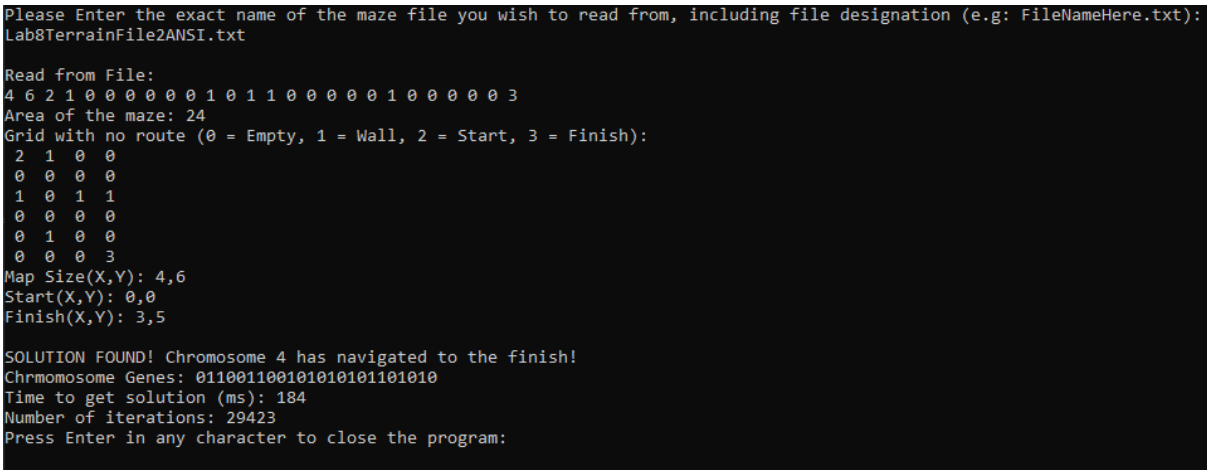


```
Please Enter the exact name of the maze file you wish to read from, including file designation (e.g: FileNameHere.txt):
Lab8TerrainFile2ANSI.txt

Read from File:
4 6 2 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 3
Area of the maze: 24
Map Size(X,Y): 4,6
Start(X,Y): 0,0
Finish(X,Y): 3,5

Grid with generated route (0 = Empty, 1 = Wall, 2 = Start, R = Route):
Time to calculate the route (ms): 0
Route:
12211
 2  1  0  0
 0  R  0  0
 1  R  1  1
 0  R  0  0
 0  1  R  0
 0  0  0  R
Press Enter in any character to close the program:
```

GA:



```
Please Enter the exact name of the maze file you wish to read from, including file designation (e.g: FileNameHere.txt):
Lab8TerrainFile1ANSI.txtt

Read from File:
6 4 0 0 1 0 0 0 2 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 3 0
Area of the maze: 24
Grid with no route (0 = Empty, 1 = Wall, 2 = Start, 3 = Finish):
 0  0  1  0  0  0
 2  0  1  0  1  1
 0  0  1  0  0  0
 0  0  0  0  3  0
Map Size(X,Y): 6,4
Start(X,Y): 1,2
Finish(X,Y): 5,2

SOLUTION FOUND! Chromosome 0 has navigated to the finish!
Chrmomosome Genes: 0001000101010110111111001
Time to get solution (ms): 1
Number of iterations: 1
Press Enter in any character to close the program:
```



```
Please Enter the exact name of the maze file you wish to read from, including file designation (e.g: FileNameHere.txt):
Lab8TerrainFile2ANSI.txt

Read from File:
4 6 2 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 3
Area of the maze: 24
Grid with no route (0 = Empty, 1 = Wall, 2 = Start, 3 = Finish):
 2  1  0  0
 0  0  0  0
 1  0  1  1
 0  0  0  0
 0  1  0  0
 0  0  0  3
Map Size(X,Y): 4,6
Start(X,Y): 0,0
Finish(X,Y): 3,5

SOLUTION FOUND! Chromosome 4 has navigated to the finish!
Chrmomosome Genes: 0110011001010101011101010
Time to get solution (ms): 184
Number of iterations: 29423
Press Enter in any character to close the program:
```

Each method makes use of the same system of asking the user input the name of the file they wish to read the maze from, meaning that each method can support different maze files as long as the configuration within those files is correct and the file name is input correctly.

Conclusion

In the end, the methods producing an accurate result is paramount, and to that end A* clearly is better implemented than GA, as GA is often produces a result it believes is accurate, but isn't. There are other factors that lead to A* being better implemented that GA, but they all seem insignificant when compared to the inaccuracy of GA's result.

References:

ANN Research References:

Artificial Neural Networks [Online] – Ajith Abraham – Available From: http://wsc10.softcomputing.net/ann_chapter.pdf – [Accessed 12th January 2022]

Fundamentals of Artificial Neural Networks [Online] – Mohamad H. Hassoun – Available From: https://www.google.co.uk/books/edition/Fundamentals_of_Artificial_Neural_Networ/Otk32Y3QkxQC?hl=en&gbpv=1&dq=artificial+neural+networks&pg=PR13&printsec=frontcover – [Accessed 12th January 2022]

Principles of Artificial Neural Networks [Online] – Daniel Graupe – Available From: https://www.google.co.uk/books/edition/Principles_Of_Artificial_Neural_Networks/Zz27CgAAQBAJ?hl=en&gbpv=1&dq=artificial+neural+networks&pg=PR7&printsec=frontcover – [Accessed 12th January 2022]

RL Research References:

Deep Reinforcement Learning that Matters (PDF needs to be downloaded from the linked website) [Online] – Pete Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, David Meger – Available From: https://ojs.aaai.org/index.php/AAAI/article/view/11694 – [Accessed 12th January 2022]

Playing Atari with Deep Reinforcement Learning (PDF needs to be downloaded from the linked website) [Online] - Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller – Available From: https://arxiv.org/abs/1312.5602 – [Accessed 12th January 2022]

Grid Path Planning with Deep Reinforcement Learning: Preliminary Results (Example of an actual study done on how effective RL is at pathfinding with results – PDF needs to be downloaded from the linked website) - Aleksandr I. Panov, Konstantin S. Yakovlev, Roman Suvorov – Available From: https://www.sciencedirect.com/science/article/pii/S1877050918300553 – [Accessed 12th January 2022]

GA Research References:

Real-Time Pathfinding with Genetic Algorithm (PDF needs to be downloaded from the linked website) [Online] - Alex Fernandes da V. Machado, Ulysses O. Santos, Higo Vale; Rubens Gonçalvez, Tiago Neves, Luiz Satoru Ochi, Esteban W. Gonzalez Clua – Available From: https://ieeexplore.ieee.org/abstract/document/6363236 – [Accessed 13th January 2022]

Using a Genetic Algorithm to Explore A*-like Pathfinding Algorithms (PDF needs to be downloaded from the linked website) [Online] – Ryan Leigh, Sushil J. Louis, Chris Miles – Available From: https://ieeexplore.ieee.org/abstract/document/4219026 – [Accessed 13th January 2022]

An Introduction to Genetic Algorithms [Online] – Melanie Mitchell – Available from: https://www.google.co.uk/books/edition/An_Introduction_to_Genetic_Algorithms/0eznlz0TF-IC?hl=en&gbpv=1&dq=genetic+algorithms+&pg=PP9&printsec=frontcover – [Accessed 13th January 2022]

General References:

Real-time agent navigation with neural networks for computer games (Covers a multitude of AI methods used within video games, including Artificial Neural Networks, Genetic Algorithms and A* pathfinding) [Online] – Ross Graham – Available From: https://gamesitb.com/wp-content/uploads/2018/10/learning_thesis.pdf – [Accessed 12th January 2022]