American University of Armenia

College of Science and Engineering

CS 246A: Artificial Intelligence

The Game "Snake"

Team: Leonid Sarkisyan, Karen Sahakyan, Arutiun Dzhilavyan, Areg Hovakimyan

Fall, 2023

Abstract

Our paper will present four search algorithms to play “Snake” and analyze their outcomes. The algorithms used in the Study include Greedy Best First Search, A* Search, Depth First Search, and Genetic Algorithms. These methods will be beneficial in solving the “Snake” game, but the performance of each algorithm may vary. To determine the most effective strategy for achieving our goals, we conducted several experiments to evaluate the efficiency of each algorithm used and eventually came to a logical conclusion.

Keywords: “Snake,” Greedy Best First Search, A* Search, Depth First Search, Genetic Algorithm, Manhattan Distance, Heuristic

Contents

List of Figures

Chapter 1

An Introduction

Completing this report is required for the Artificial Intelligence course, CS 246. We made the decision to use several AI search algorithms to play Snake in order to evaluate their performance. This required developing a game application using four distinct AI Search Algorithms. Through a series of in-depth studies, we will compare and practically illustrate how well and efficiently they work.

The legendary "Snake" game dates back to the late 1970s. It was developed by Taneli Armanto and gained popularity in the early 2000s when it was pre-installed on early Nokia models. The rules of the game are very straightforward: control the Snake around the screen, consuming food (represented by dots) to grow longer while avoiding the walls (borders), and most importantly, itself. The aim of the game is to grow as large as possible while obeying the rules mentioned above.

Despite its simple rules, the game becomes very challenging and technical as the Snake gets longer and longer. Players become very competitive by working on their reflexes and critically planning each and every move possible to achieve the highest score [Ang21]. Being part of those competitive players, it would be interesting to see how an AI model containing various search algorithms will perform in a similar environment.

1.1 The Structure of the Project Paper

The Paper will be divided into four Sections:

1. Literature Review
2. Method
3. Evaluation

4. Conclusion

The Literature Review section will be used to study and compare various existing projects related to our topic to understand how they work and what they achieved, followed by the Methods section, which will be devoted to explaining what our implemented Search Algorithms do and how they function. Finally, the Performance section will contain various tables and charts analyzing the efficiency of each implemented algorithm, after which we will conclude our project and address possible future improvements.

Chapter 2

Literature Review

While searching for related topics, we came across various Scientific Papers and Documentation, which will all be listed in Chapter 6 as References. After examining them, we decided to review two of them.

No. 1: *"Automated Snake Game Solvers via AI Search Algorithms"* by S. Kong, J. A. Mayans. [SK16]

In this paper, the authors discussed potential solutions to the Snake game using different Search algorithms. The algorithms mentioned are Best First Search, A* Search and its variations, Random Move Method, and Almighty Method.

No. 2: *"Neural Networks using Genetic Algorithms"* by R. Mahajan, G. Kaur. [RM13]

In the second paper, the authors focus on an in-depth representation of how the Neural Networks function using the Genetic Algorithm. The report is divided into four main sections: Neural Networks, Implementation of GA in Travelling Salesman Problem, Application area of GA/ Neural network, and Pros and Cons, although for our project, we will be focusing on implementing the Genetic Algorithm for our purposes.

2.1 Literature Review 1

In this project, [SK16] S. Kong and J. Aguilar Mayans from The University of California: Irvine took upon the challenge of analyzing five algorithms for solving the Snake game. They include: Best First Search Algorithm, A* Search Algorithm, A* Search Algorithm with Forward Checking, Random Move Method, and Almighty Move Method. The review will contain an in-depth representation of all the above-mentioned algorithms, followed by a brief conclusion.

2.1.1 Greedy Best First Search Algorithm

Firstly, they examined how the Greedy Best First Search Algorithm comes up with a solution for solving The Game. The experiment showed that it focuses on short-term goals and considers the action of moving the Snake to the position on the board closest to the goal (Apples, as they describe). It also guarantees that the Snake will eat the apple in an optimal (shortest) way possible only for the first four apples since a snake with a length of five cannot construct a circle on the board. It also used the Manhattan distance to measure the distance from the head of the Snake to the apple.

2.1.2 A* Search Algorithm

After experimenting with the Best First Search Algorithm mentioned in the previous subsection, the authors noticed that, occasionally, as the Snake got longer, it tended to run into dead ends or, in other words, falling into infinite loops. The A* Search Algorithm was then used, considering the Manhattan Distance as a heuristic function to measure the distance from the Snake's head to the apple. The Algorithm also kept track of each step, storing them separately as a cost function. As the authors mentioned, in a way, the A* Search Algorithm was a success since it tried to find a complete path to the apple before making its first step. This approach decreased the probability of falling into an infinite loop. However, there were still a few exceptions that the Algorithm could not get around.

On the other hand, it guaranteed finding the shortest way possible to the goal. It is also important to mention that the experiment used a slightly modified version of the A* Search Algorithm. The two main differences are stated below:

1. Whenever the Algorithm ran into two nodes with the same Cost Estimate, it would continue its path through the node with the least Heuristic.

2. A limit was placed on the maximum number of nodes the Algorithm could expand. Once the limit was reached and the Algorithm could not find an optimal path, it would automatically switch back to using the Best First Search Algorithm to achieve the desired goal (Apples) instead.

2.1.3 A* Search Algorithm With Forward Checking

The last Search Algorithm mentioned in the paper is another version of the A* Search Algorithm, this time with Forward Checking, which considers the post-

apple eating actions, thus minimizing the chances of running into dead ends (Infinite Loops). Before taking each step, the Algorithm considers all possible outcomes that can result from it, thus minimizing the threat of falling into loops. This version of A* Search is equipped with the Breadth First Search Algorithm, which is specifically used to avoid loops. Whenever an iteration of the A* Search Algorithm ends, the Breadth First Search Algorithm is initiated at the goal state found by the previous Algorithm (A* Search).

2.1.4 Random Move Method

Apart from the Search Algorithms mentioned above, two baseline methods were also tested, the first being the Random Move Method. As the name suggests, this Method chooses the next move randomly with only one condition: the next move selected by the Method should not result in losing the game.

2.1.5 Almighty Move Method

The second non-search algorithm approach is the Almighty Move Method. An Almighty Move, as used in Artificial Intelligence, is a particular tactic or Algorithm that ensures the highest possible score or result in a given situation. Almighty Moves can be highly complicated and computationally costly, but they are usually incredibly potent and successful. They can, therefore, be helpful in researching the theoretical bounds of what artificial intelligence is capable of.

This approach is optimal when the board has an even height and width (Fig. 2.1, 2k x 2k) and guarantees that the Snake will eat the maximum number of apples. By doing some slight modifications, the authors also devised a similar method, which would work on boards consisting of odd-number height and width (Fig. 2.2 2k+1 x 2k+1). The main functionality of this approach is that it creates a circuit for the Snake to follow. The authors concluded that the Almighty Method is an extreme case, and since the game does not have a time limit, this Method will, with certainty, maximize the score.

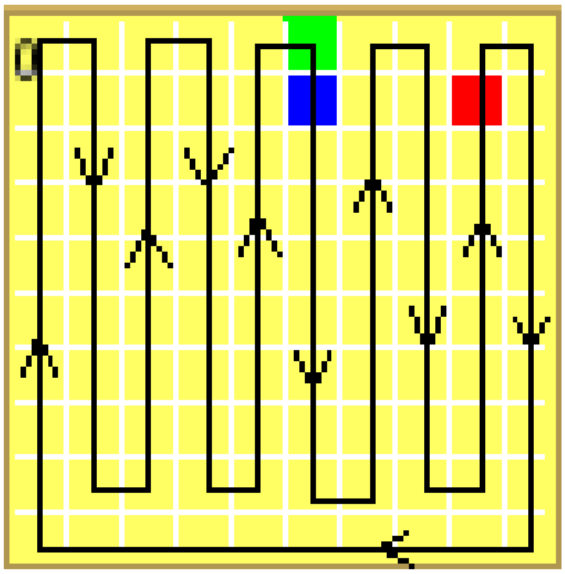

Figure 2.1: Optimal Almighty Move circuit for even-numbered height and width.

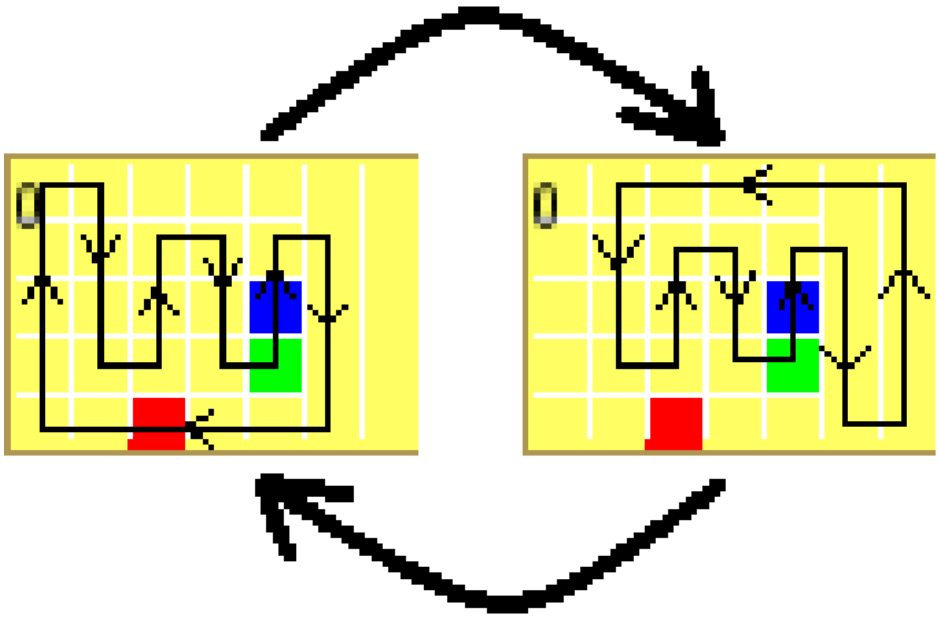

Figure 2.2: Optimal Almighty Move circuit for odd-numbered height and width.

2.1.6 Conclusion 1

In this report, the authors presented five algorithms and methods for solving The Snake Game. The authors concluded that apart from the Random Move Method, which is considered a baseline method, the remaining algorithms each had their advantages and disadvantages, depending on whether the Algorithm was designed for reliability or speed.

The results showed that compared to other algorithms used, the Almighty Method had a slow start, but it was the only one that guaranteed the maximum score and highest efficiency at the end of the game. The authors also noted that a combination of various algorithms can be used to achieve the optimal solution.

2.2 Literature Review 2

As mentioned above (2), this literature [RM13] review will contain sections about Neural Networks and Genetic Algorithms. The report by R. Mahajan and G. Kaur from Dev University, Amritsar, helped us understand how genetic algorithms operate, what they do, and what their limitations are.

2.2.1 Neural Networks and Genetic Algorithm

Dr. David Goldberg, 1989 offered the following definition: "Genetic Algorithms are Search Algorithms Based on The Mechanics of Natural Selection and Natural Genetics." [RM13] The Genetic Algorithm contains a binary bit string whose initial values are randomly selected and evaluated into a fitness function. The Algorithm then differentiates between the success probabilities of each individual string and ranks them by their fitness value. The Genetic Algorithm consists of three main phases: Selection, Crossover, and Mutation. [Chr19]

1. **Selection** - The parent Snake whose Fitness Value is higher has a better chance of being selected to play in the next game.

2. **Crossover** - The first step of the crossover, also known as recombination, includes randomly selecting two Snakes to become parents of a new Snake. The second part consists of choosing the crossover point and exchanging all the bits up to and after it. The crossover point is chosen randomly as well. In the Figure below (Fig. 2.3), the crossover point is seven.

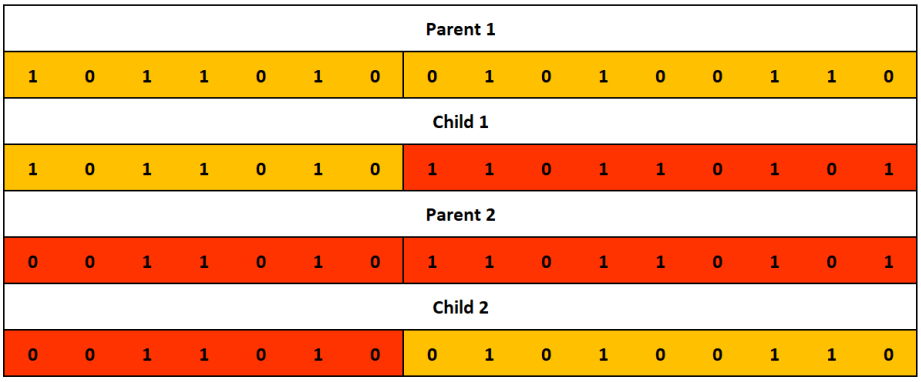

Figure 2.3: Two Parent and two Child Snakes with the crossover point chosen to be seven.

3. **Mutation** - Each node is assigned a random, independent probability, which describes how likely each bit is to undergo a mutation. If the mutation is applied to a bit, its value will change. If the initial value is 0 (zero), it will become 1 (one) and vice versa. In Figure 2.4, the mutation is applied to the seventh bit.

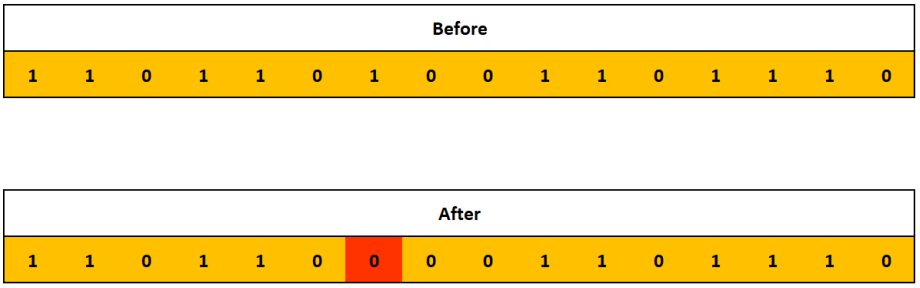

Figure 2.4: The application of Mutation on the 7th bit.

2.2.2 Pros and Cons of The Genetic Approach

Advantages

- The Algorithm allows the Snake to evolve over time.
- The Algorithm is very efficient in exploring large Search Spaces, which allows it to find an optimal solution.
- A diverse population of Snakes, through the Genetic Algorithm, can overcome challenges during different times of The Game.

Disadvantages

- Training the data through Genetic Algorithms can become very costly largely because of the growing population size.
- The Algorithm may struggle if the environment changes.
- Defining a Fitness Function for the Algorithm can be complicated.

2.2.3 Conclusion 2

To conclude, the authors mentioned that the Genetic Algorithm has three main operators: Selection, Crossover, and Mutation. Neural Networks, on the other hand, have various processing elements called neurons. Together, these elements form black boxes, which produce an output once fed with information. The authors also mentioned that the Genetic Algorithm has many limitations, which, if resolved, can help it become much more efficient in solving real-life problems.

Chapter 3

Methods

In this chapter, we will define the rules of the game, formulate the problem as a Search Problem, and present four different AI Search Algorithms that will be used to achieve their maximum possible score.

3.1 Rules of The Game

This section will clarify some characteristics and basic rules that have been used to implement our game. To put it into simple words, the game consists of a Snake with an initial length of 3 (three) trying to consume as many dots (food) as possible while moving around on a n-by-n square board surrounded by borders (walls). The objective is to achieve the highest score possible, which, in our case, will be n*n. The list of the rules is presented below.

1. The size of the square (Board) that the game is played on is always consistent. (Never Changes)

2. There are four directions the Snake can take: left, right, up, and down. However, only three of the directions can be available at any time. The Snake cannot return to the path it originally came from. In other words, it can never swap opposite directions, such as up and down or left and right.

3. After passing through the dot (consuming the food), the Snake will grow by one unit measure, which will be added to its tail and reflected immediately after the Snake's tail passes through the dot (food).

4. Only one dot will be available on the board until the Snake consumes it. After successfully consuming the dot, another dot will randomly appear (With

Uniform Probability) on the board. New dots cannot appear on occupied tiles. (Tiles are Occupied by The Snake)

5. The game will end as soon as the head of the Snake touches either the border (Wall) surrounding the board or crosses paths with its own body.

6. After the completion of the game, the score will be calculated by the number of dots (food) the Snake consumes.

7. The goal of the game is to achieve the highest score possible. The first priority for our model will be not losing the game (Following the rules mentioned in Point 5), while the second priority will be increasing our final score (Consuming as many dots as possible in a finite amount of time).

3.2 Algorithms

In this section, various AI Search Algorithms will be used to achieve their highest potential score. We will analyze the way each individual algorithm works, talk about their advantages and disadvantages, and finally document their results in Chapter 4.

3.2.1 Formulation of The Problem as a Search Problem

1. **State Space** - Fully Observable, Single-Agent, Partially Non-Deterministic(Randomized food placement in the initial state), Sequential, Dynamic, Discrete, Known.

2. **Initial State** - (x, y) coordinates of the Snake and (i, j) coordinates of the dot (food)

3. **Possible Actions** - Up, Down, Left, Right

4. **Transition Model** - When the Snake moves, it changes its position; when it consumes food, it increases its length; when it collides with a border or itself, the game ends.

5. **Goal State** - Achieve the Highest possible score by consuming food while avoiding collisions.

6. **Cost Function** - Distance traveled by the Snake's head measured in units of one.

7. **Heuristic Function** - Distance between the Snake's head and the food measured in units of one.

3.2.2 Greedy Best First Search Algorithm

The first algorithm that we are going to explore is The Greedy Best First Search Algorithm. This algorithm is optimal in a way that it will find the closest path to the dot (food). We will consider the Manhattan Distance as a way to measure how close the head of the Snake is to the dot. We can also be confident that the algorithm will undoubtedly find the optimal (shortest) path to the dot; however, in doing so, it can overlook certain crucial factors, such as accidentally hitting itself or any of the borders, which will cost us to lose the game. It also has the ability to get stuck sometimes, as we inferred from our results in Chapter 4. (2.1.1)

3.2.3 A* Search Algorithm

The A* Search Algorithm is different from the Best First Search Algorithm in that it considers two parameters instead of just one. The algorithm examines not only the goal (The Dot) and the distance from it but also the current state and the area that it has searched so far (The Cost). As a heuristic, the algorithm uses the Manhattan Distance from the head of the Snake to the dot (Our Goal) while also keeping track of the number of steps it has taken as a Cost Function. The A* Search Algorithm is an improvement from the Best First Search Algorithm as it finds the complete path to the dot (Our Goal) as a pose to taking one step at a time. This kind of search has many advantages, most notably that it avoids getting stuck (Falling Into Loops) on the way to our goal. Since the game does not have any time restrictions, the algorithm is guaranteed to find an optimal path if, of course, it exists. (2.1.2)

3.2.4 Depth First Search Algorithm

Compared to the previous two algorithms, the Depth First Search algorithm takes a lengthier approach. Since our game has no time limit, the Depth First Search will eventually be guaranteed to find a solution; however, as we will see in Section 5 after the Snake reaches twice the length of our board (2n if we have an n-by-n Board) it may occasionally trap itself and lose the game. Using the DFS algorithm, the Snake, from its original starting point, will start making its way to the left until it reaches the wall, after which it will either move one block up or down depending on the location of the following randomly generated dot (Our Goal).

3.2.5 Neural Networks and Genetic Algorithms

Neural Networks used in the Snake game implementation consist of three layers of nodes, also known as neurons. The layers are: [Chr19]

1. Input layer
2. Hidden layer
3. Output layer

In the Input Layer, the agent perceives the information about the game's current state. It keeps track of the Snake's and the dot's (Food) location and coordinates while also keeping track of the direction of the head and tail. The Hidden Layer is responsible for choosing the proper action in each state of the game, while the Output Layer is where the network chooses the action to perform from the Hidden Layer. The actions are directions for the Snake to follow one direction at a time. The Genetic Algorithm consists of two components for playing the Snake game: a Training Model and a Loading Model. Before starting the training of the model, we should specify four essential values.

1. **Number of Population** - The number of Snakes competing against each other.
2. **Number of Generations** - The number of generations of Snakes that will go through selection, crossover, and mutation.
3. **Number of Hidden Nodes** - The number of nodes the Hidden Layer will contain.
4. **Mutation Rate** - (2.2.1) The percentage of probability that a bit of a node should exceed to undergo a mutation.

After setting each parameter, the training of the model will begin, after which the Snake will be able to play from any state. X (Number of Population) number of Snakes will play Y (Number of Generations) times, and they will choose their available actions to perform from the Neural Network. Then, after each generation, X number of Snakes will go through three main phases (as mentioned above (2.2.1) of the Genetic Algorithm, and new Generations of Snakes will proceed to the next round. After Y rounds (Generations) of gameplay, the Snake with the best gameplay tactic will be selected throughout all Y generations of X Snakes, which

will become our best model, after which the model will be tested and analyzed to obtain its performance and results.

Chapter 4

Performance and Evaluation

This chapter will cover the results of the Algorithms mentioned in the previous Chapter (3), along with an in-depth interpretation of each one of them. It will also contain various tables and charts to make it easier to visualize.

4.1 Greedy Best First Search Evaluation

As depicted in the figure below (Figure 4.1), the Greedy Best First Search Algorithm (3.2.2) achieved a reasonable mean score of 60.80 after 10 observations, with its peak score reaching 83 and a low point of 41. The Greedy Best First Search, however, failed around its mean score, where the Snake occasionally trapped itself and eventually ran into its own body. Despite this limitation, the algorithm's overall performance highlighted its ability to navigate the Snake game environment effectively, strategically hunting for food using the shortest possible path.

Greedy Best First Search	
Observation Count	**Score**
O1	63
O2	83
O3	58
O4	61
O5	69
O6	41
O7	45
O8	45
O9	82
O10	61
Mean	**60.80**

Figure 4.1: Observation, Score and Mean value for the Greedy BFS Algorithm.

4.2 Depth First Search Evaluation

In comparison, the Depth First Search Algorithm (3.2.4) demonstrated a different approach in our Snake game, yielding an average score of 37.9 after 10 observations. While its highest score reached 40 and the lowest dipped to 37 (Figure 4.2), the algorithm had a critical flaw. Once the Snake extended to twice the length of the board, it often found itself in a predicament, either colliding with a wall or being trapped within its own body. This limitation underscored the challenge of balancing exploration and self-preservation, showcasing the algorithm's struggle when navigating the maze-like dynamics of the game.

Depth First Search	
Observation Count	**Score**
O1	40
O2	37
O3	38
O4	37
O5	39
O6	38
O7	37
O8	38
O9	37
O10	38
Mean	**37.90**

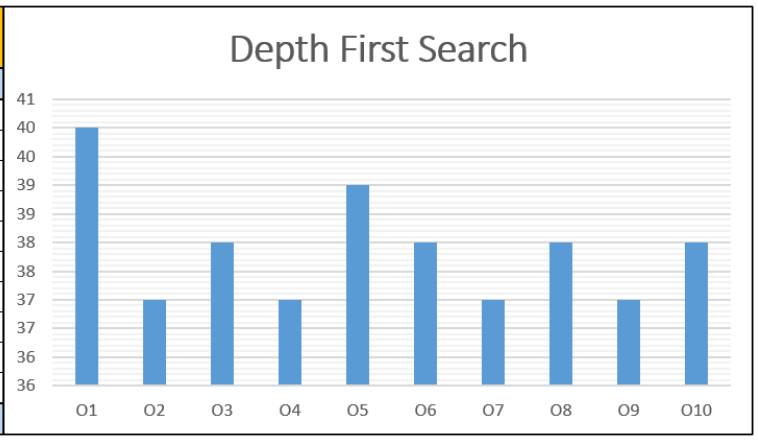

Figure 4.2: Observation, Score, and Mean value for the DFS Algorithm.

4.3 A Star Search Evaluation

The A Star Search Algorithm's (3.2.3) performance was comparable to that of the Greedy BFS, showing remarkable consistency with a 60.5 average score after 10 observations. Interestingly, it demonstrated an impressive high score of 100 and a good low score of 44, as illustrated in Figure 4.3. This steady high performance highlights how well the algorithm handles the task's intricacies. Nevertheless, the algorithm had difficulties in the mid-50s range, especially while negotiating turns, which resulted in situations where it ran out of space. This subtle discovery highlights the need for improvements to the navigational capabilities of the A* Search Algorithm and throws light on an important aspect of its functionality. In order to maximize the algorithm's performance in every aspect of the game and guarantee more resilient and flexible navigation techniques, it is imperative that these particular issues be resolved.

A* Search	
Observation Count	Score
O1	84
O2	48
O3	100
O4	47
O5	56
O6	47
O7	73
O8	59
O9	47
O10	44
Mean	60.50

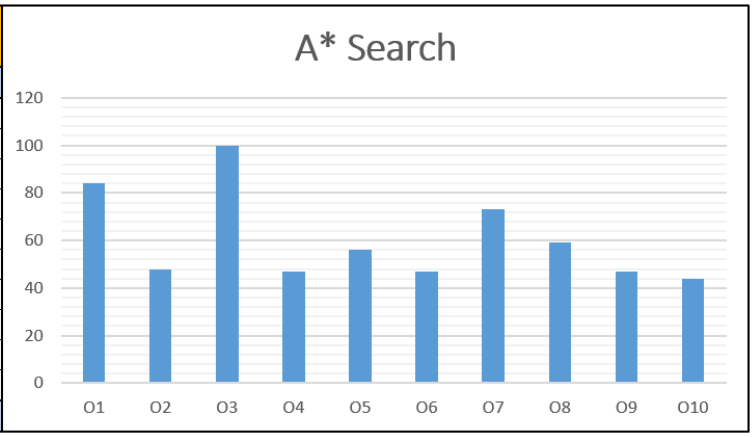

Figure 4.3: Observation, Score, and Mean value for the DFS Algorithm.

4.4 Genetic Algorithm Evaluation

As we can infer from the Figure below (Figure 4.4), the Genetic Algorithm (3.2.5) overall performed the best, achieving a mean score of 63.9 after 10 observations, with its highest score being 77 and a low score of 51, which is still very respectable. To achieve these results, the algorithm had to be trained, for which it required some additional information, which included the Number of Populations equivalent to 300, the Number of Generations equal to 30, eight Hidden Layers, and a Mutation Rate of 12 percent.

Genetic Algorithm			
Observation Count	Score	Stats	Number
O1	58	No. Population	300
O2	77	No. Generations	30
O3	54	No. Hidden Nodes	8
O4	62	Mutation Rate %	12
O5	72		
O6	51		
O7	72		
O8	71		
O9	66		
O10	56		
Mean	63.90		

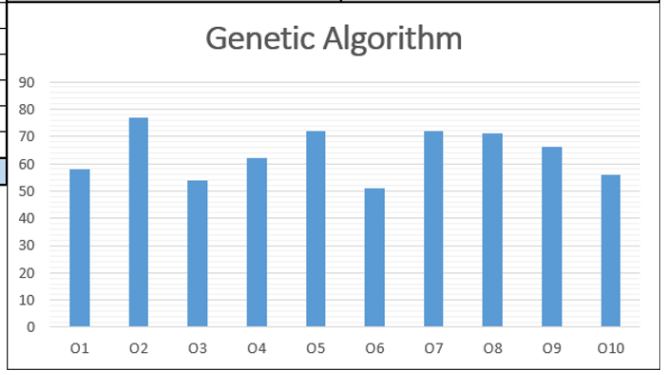

Figure 4.4: Observation, Score, and Mean value, along with information about No. of Population, No. of Generations, No. of Hidden Layers, and Mutation Rate Percentage for the Genetic Algorithm.

Chapter 5

Conclusion

In conclusion, our exploration of various algorithms in the Snake Game project uncovered unique strengths and limitations. The Greedy Best First Search showcased strategic decision-making but struggled with self-trapping. Depth First Search faced challenges in navigation and spatial awareness, while A* Search demonstrated versatility but struggled with space limitations during turns. The Genetic Algorithm, however, was the most reliable and consistent, delivering outstanding performance with an average score of 63.9. Its strategic training approach, featuring a population of 300, 30 generations, 8 hidden layers, and a 12 percent mutation rate, allowed it to excel. As we move forward, refining these algorithms presents exciting possibilities for optimizing gameplay and pushing the boundaries of artificial intelligence in gaming scenarios.

Algorithms	**Average Score**	**Highest Score**
Greedy Best First Search	60.8	83
Depth First Search	37.9	40
A* Search	60.5	100
Genetic	63.9	77

Table 5.1: Example Table with 3 Columns and 4 Rows

5.1 Further Improvement

The foundation of the code for the initialization of this project was taken from GitHub.com [Rim22]. It is important to note that the Algorithms used for the project were heavily modified to achieve higher results. The methods can further be improved and optimized by combining multiple algorithms to achieve higher

results. The Greedy Best First Search Algorithm, for instance, can be combined with the Genetic Algorithm consisting of an improved Neural Network which can be accomplished by using the "PyTorch" library. This way, for the first 10 dots (Food), The Greedy BFS can be used for the optimal solution after which it can switch to the Trained Genetic Algorithm to further advance into the game.

Bibliography

[RM13] Gaganpreet Kaur Richa Mahajan. "Neural Networks Using Genetic Algorithms". In: *International Journal of Computer Applications* (2013). https://shorturl.at/fkl34.

[SK16] Joan Aguilar Mayans Shu Kong. "Automated Snake Game Solvers via AI Search Algorithms". In: *WPMUCDN* (2016). https://shorturl.at/gtux1.

[Chr19] Chrispresso. *AI Learns to Play Snake!* https://www.youtube.com/watch?v=vhiO4WsHA6c. 2019.

[Ang21] Ayla Angelos. "The History of Snake: How the Nokia Game Defined a New Era for the Mobile Industry". In: *It's Nice That* (2021). https://shorturl.at/diAPX.

[Rim22] Rimya. "AI Snake Game". In: *GitHub* (2022). https://github.com/RIMYA/AI-SnakeGame/tree/main.