AI:
The Game “Snake”

December 2023

**Arutiun Dzhilavyan, Areg Hovakimyan,
Karen Shahakyan, Leonid Sarkisyan**

Our presentation explores five search algorithms—Greedy Best First Search, A* Search, Depth First Search, and Genetic Algorithms—in the context of playing "Snake." We conducted experiments to assess their efficiency and draw conclusions on the most effective strategy for our goals.

CS246A

The Structure of the Presentation

CS246A

CHAPTER 1

An Introduction

Introduction

For our CS 246 AI course, we tackled the challenge of playing Snake using four distinct AI search algorithms. Our goal was to evaluate their performance through in-depth studies and practical demonstrations. The classic "Snake" game, dating back to the late 1970s, serves as our testing ground. Despite its simple rules, the game becomes increasingly challenging as the Snake grows longer. As enthusiasts of the game, we're eager to observe how AI, armed with diverse search algorithms, fares in this competitive environment.

CHAPTER 2

Literature Review

Literature Review 1

“AI Solutions for the Snake Game” by S. Kong, J. A. Mayans.

The Greedy Best First Search prioritizes short-term goals, aiming to move the Snake closest to the goal (apples). However, it's optimal only for the first four apples due to the Snake's inability to form a circle with a length of five.

The A* Search Algorithm addressed issues of falling into infinite loops by finding a complete path to the apple before the first step, though exceptions remained. Modifications included heuristic prioritization and a node expansion limit, reverting to Greedy Best First Search if needed.

A* Search Algorithm with Forward Checking minimizes dead ends by considering post-apple eating actions, employing Breadth First Search to avoid loops.

Baseline methods included the Random Move Method, choosing moves randomly without losing the game, and the Almighty Move Method, ensuring the highest score through a circuit for the Snake to follow, optimal for even and odd-sized boards. (Figure 1,2)

In conclusion, each algorithm presents strengths and limitations, providing a comprehensive analysis of their effectiveness in tackling the Snake game.

Joan Aguilar Mayans Shu Kong. “Automated Snake Game Solvers via AI Search Algorithms”. In: WPMUCDN (2016).

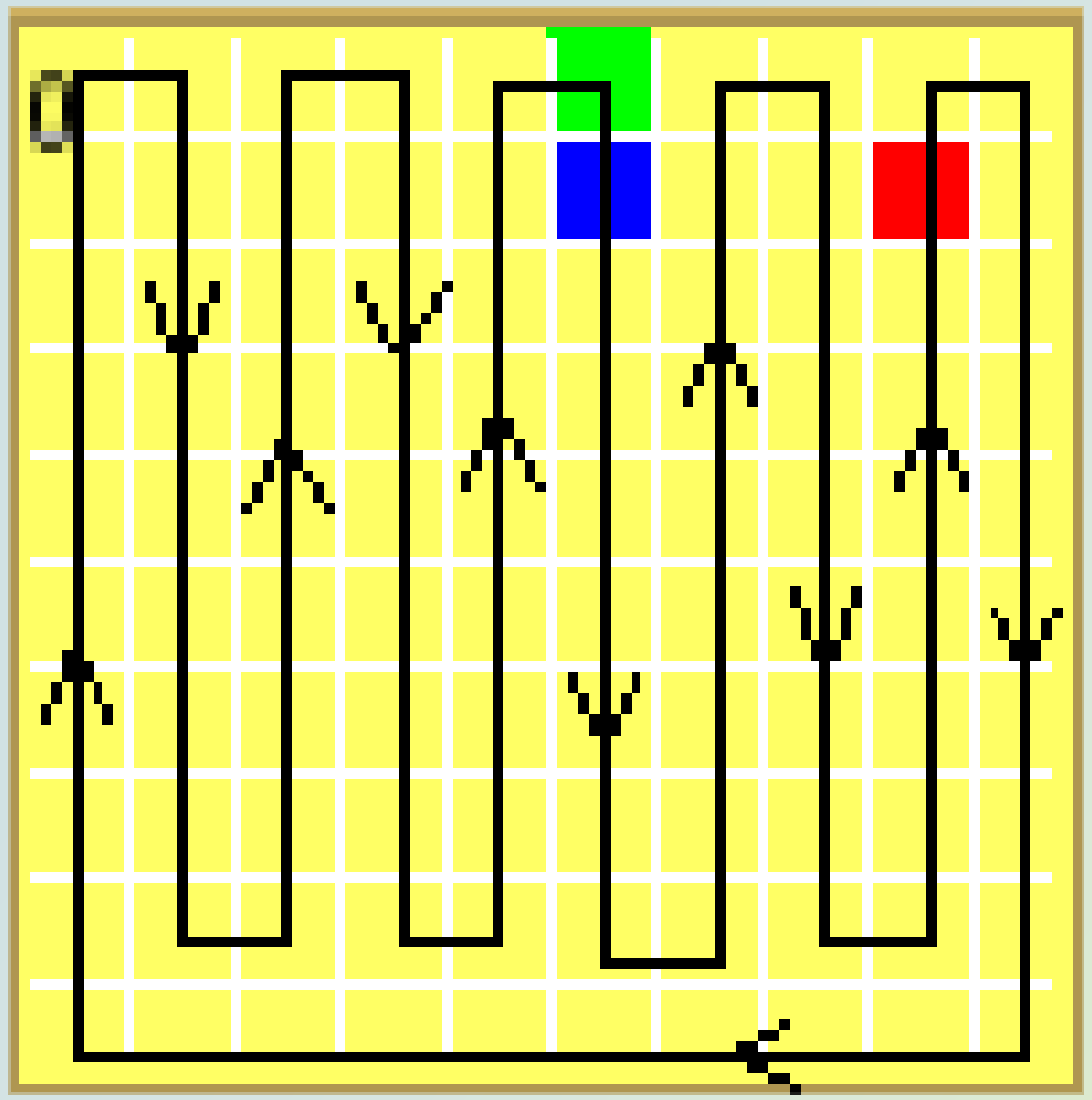


Figure 1

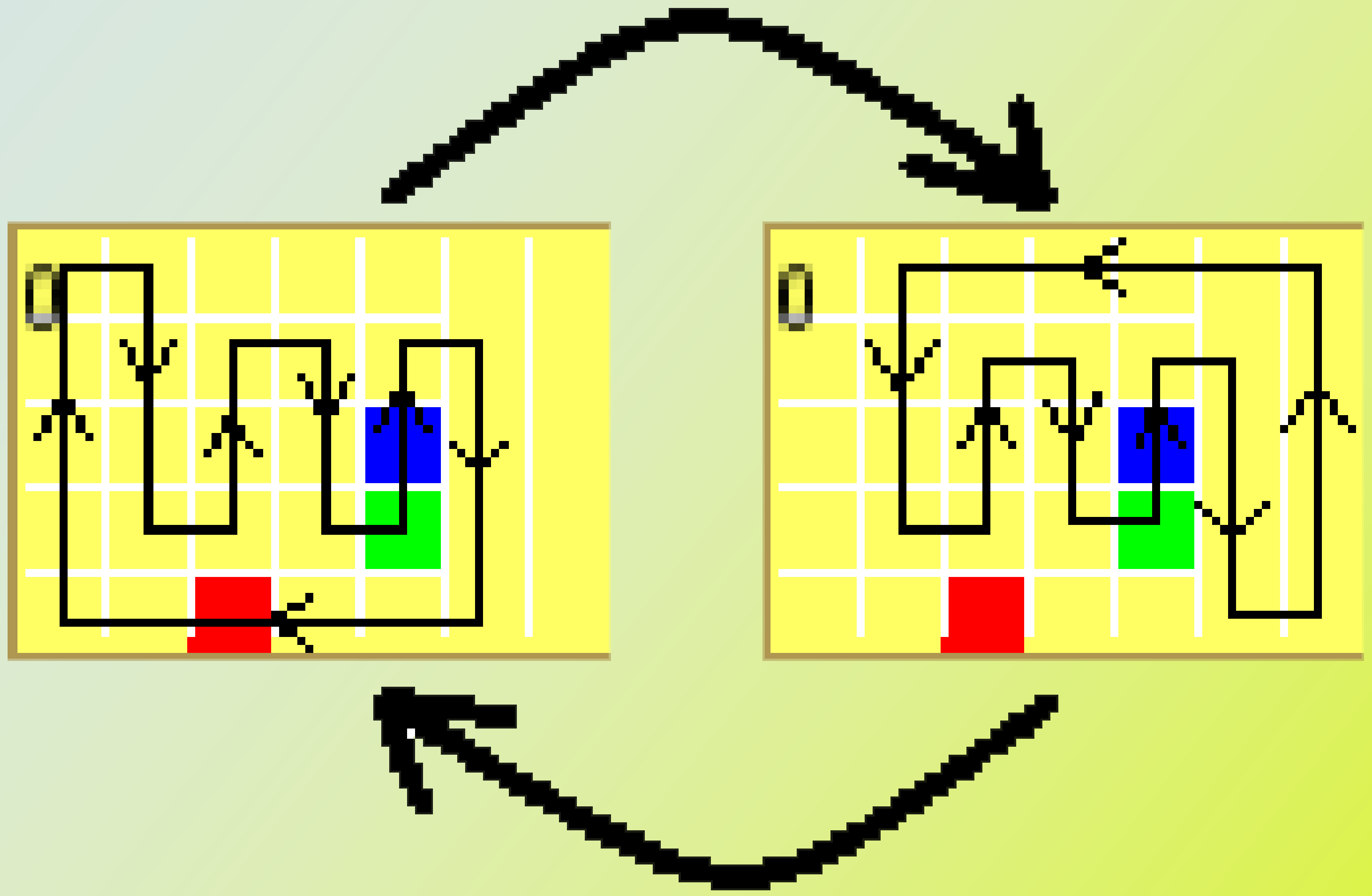


Figure 2

CS246A

Literature Review 2

“Neural Networks using Genetic Algorithms” by R. Mahajan, G. Kaur.

Dr. David Goldberg's 1989 definition describes Genetic Algorithms as search algorithms based on natural selection and genetics. The algorithm employs a binary bit string with randomly selected initial values, evaluating them through a fitness function. It then ranks individuals based on their success probabilities. The Genetic Algorithm consists of Selection, Crossover, and Mutation phases.

Pros:

- Enables the Snake to evolve over time.
- Efficient in exploring large search spaces for optimal solutions.
- Diverse populations can adapt to different game challenges.

Cons:

- Training data can be costly due to the growing population size.
- May struggle with environmental changes.
- Defining a fitness function can be complex.

Gaganpreet Kaur Richa Mahajan. “Neural Networks Using Genetic Algorithms”. In: International Journal of Computer Applications (2013).

The Genetic Algorithm

Selection

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

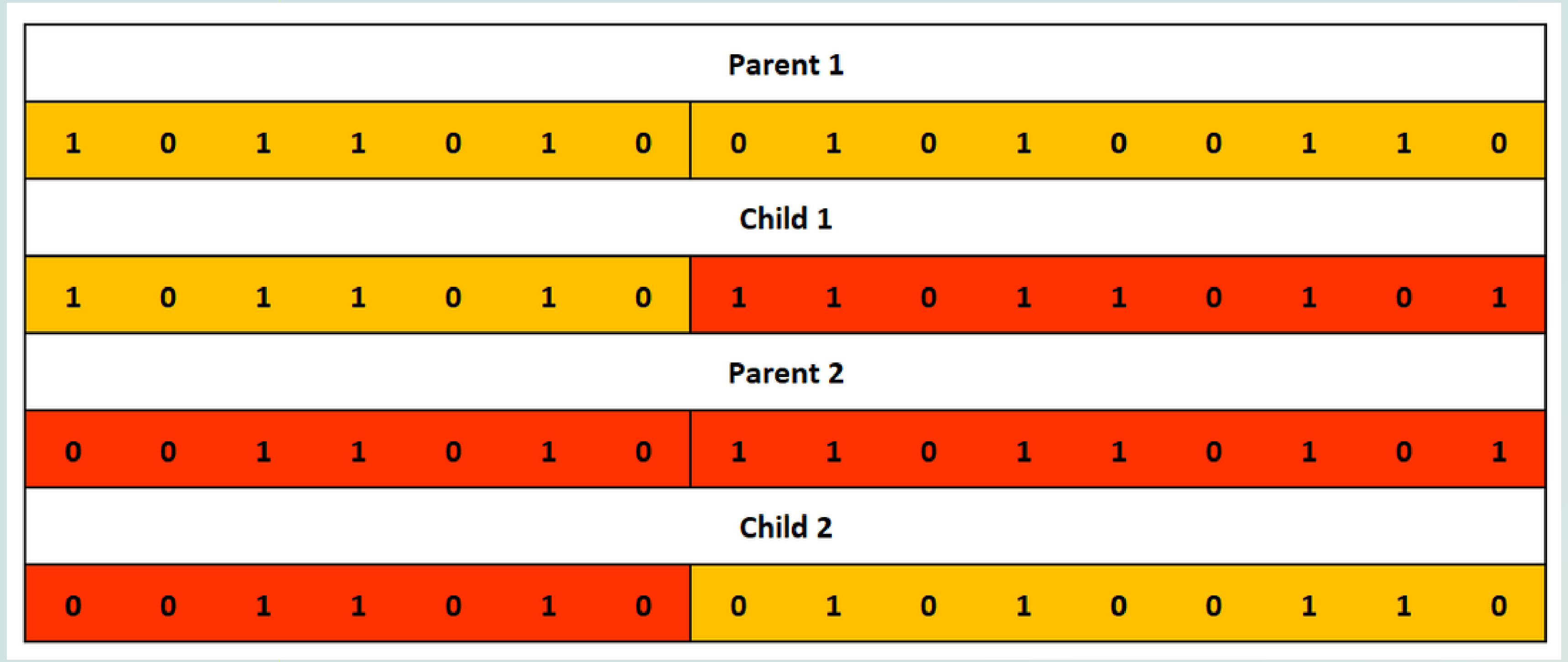
Crossover

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 3 below, the crossover point is seven.

Mutation

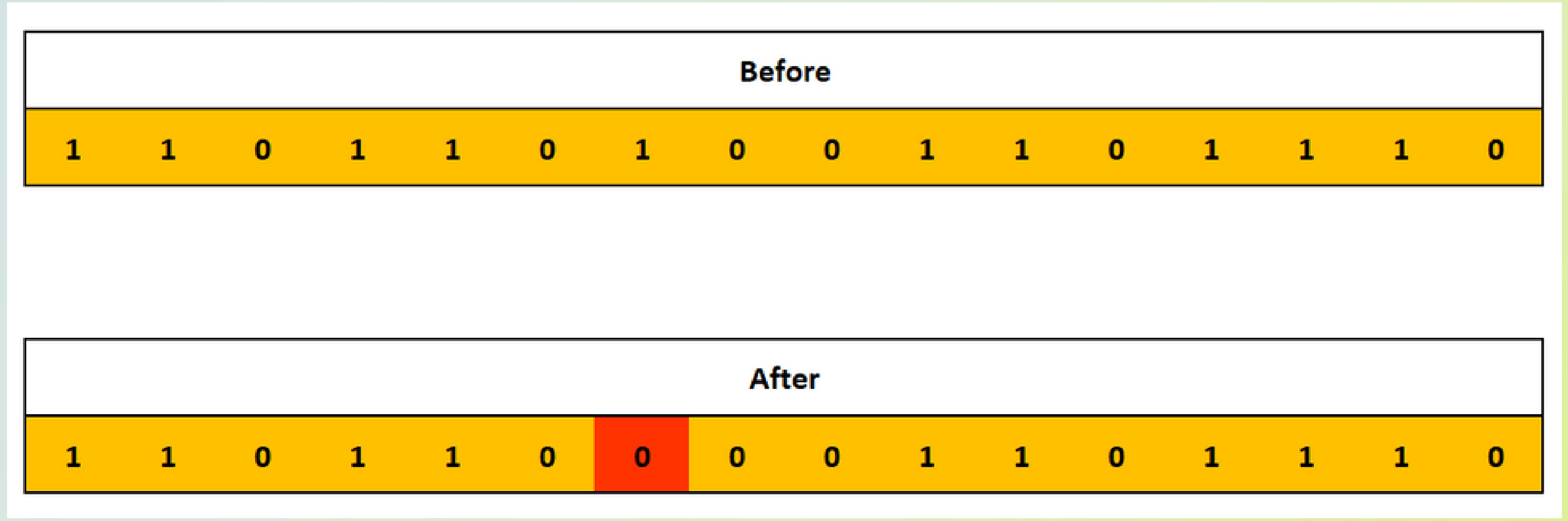
Mutation - Each node is assigned a random, independent probability, which describes how likely each bit is to undergo a mutation. If the mutation is applied to a bit, its value will change. If the initial value is 0 (zero), it will become 1 (one) and vice versa. In Figure 4 the mutation is applied to the seventh bit.

Gaganpreet Kaur Richa Mahajan. “Neural Networks Using Genetic Algorithms”. In: International Journal of Computer Applications (2013).



Parent 1	1	0	1	1	0	1	0	0	1	0	1	0	0	1	1	0
Child 1	1	0	1	1	0	1	0	1	1	0	1	1	0	1	0	1
Parent 2	0	0	1	1	0	1	0	1	1	0	1	1	0	1	0	1
Child 2	0	0	1	1	0	1	0	0	1	0	1	0	0	1	1	0

Figure 3



Before	1	1	0	1	1	0	1	0	0	1	1	0	1	1	1	0
After	1	1	0	1	1	0	0	0	0	1	1	0	1	1	1	0

Figure 4

CS246A

CHAPTER 3

Formulation of The Problem

Rules of The Game

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).

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.

CHAPTER 4

Algorithms

Algorithms

Greedy Best First Search Algorithm

Let's delve into the Greedy Best First Search Algorithm, known for finding the closest path to the food dot by using Manhattan Distance. While it guarantees an optimal (shortest) path to the dot, it may overlook factors like self-collision or hitting borders, resulting in game loss. Our experiments also revealed instances where the algorithm could get stuck.

A* Search Algorithm

Contrasting with the Best First Search Algorithm, the A* Search Algorithm considers both the goal (the dot) and the current state, using the Manhattan Distance and the number of steps as parameters. Unlike the Best First Search, A* finds the complete path to the goal, preventing getting stuck in loops. With no time restrictions in the game, the algorithm ensures an optimal path if one exists.

Depth First Search Algorithm

Contrasting with the quicker algorithms, Depth First Search takes a longer approach. While guaranteed to find a solution without a time limit, it may trap itself and lose the game after the Snake reaches twice the length of the board (2n for an n-by-n board). The Snake follows a leftward path until reaching a wall, then moves up or down based on the location of the randomly generated dot (the goal).

Neural Networks and Genetic Algorithms

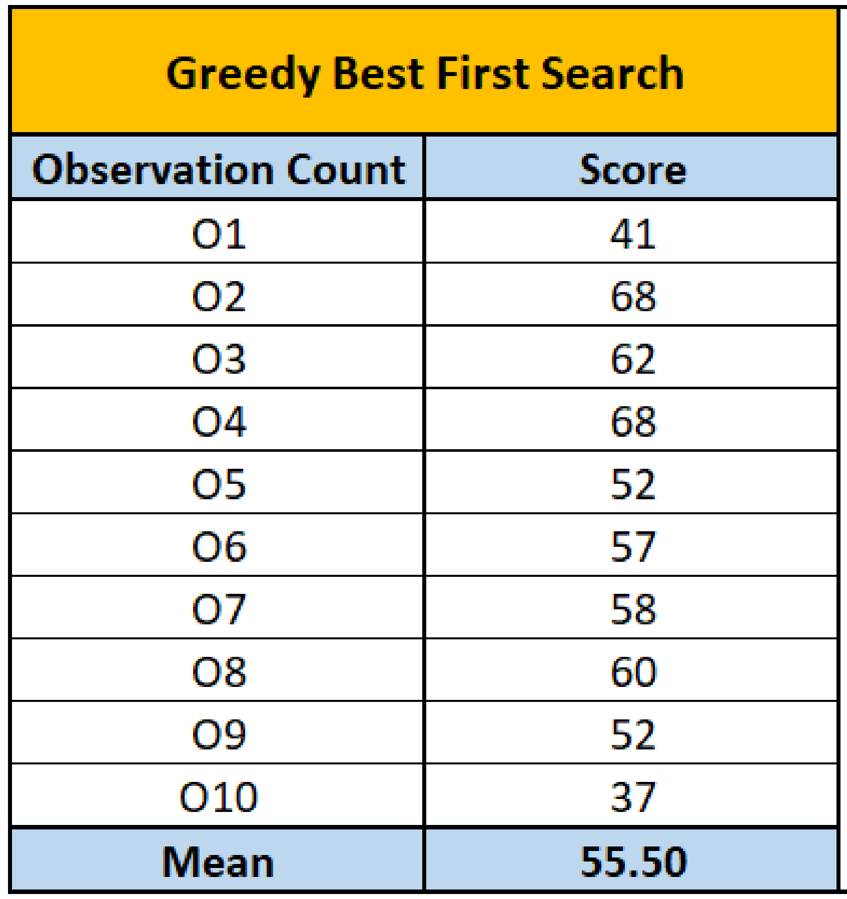
In the Snake game's Neural Network, three layers—Input, Hidden, and Output—guide the agent. The Input layer tracks Snake and dot locations, head-tail direction. The Hidden layer decides actions, and the Output layer executes chosen actions.

The Genetic Algorithm has two components: Training Model and Loading Model. Four key parameters are set before training: Number of Population, Number of Generations, Number of Hidden Nodes, Mutation Rate. Training starts with X Snakes playing Y times, selecting actions from the Neural Network. After each generation, X Snakes undergo Genetic Algorithm phases. The best tactic after Y rounds becomes the model for testing and analysis.

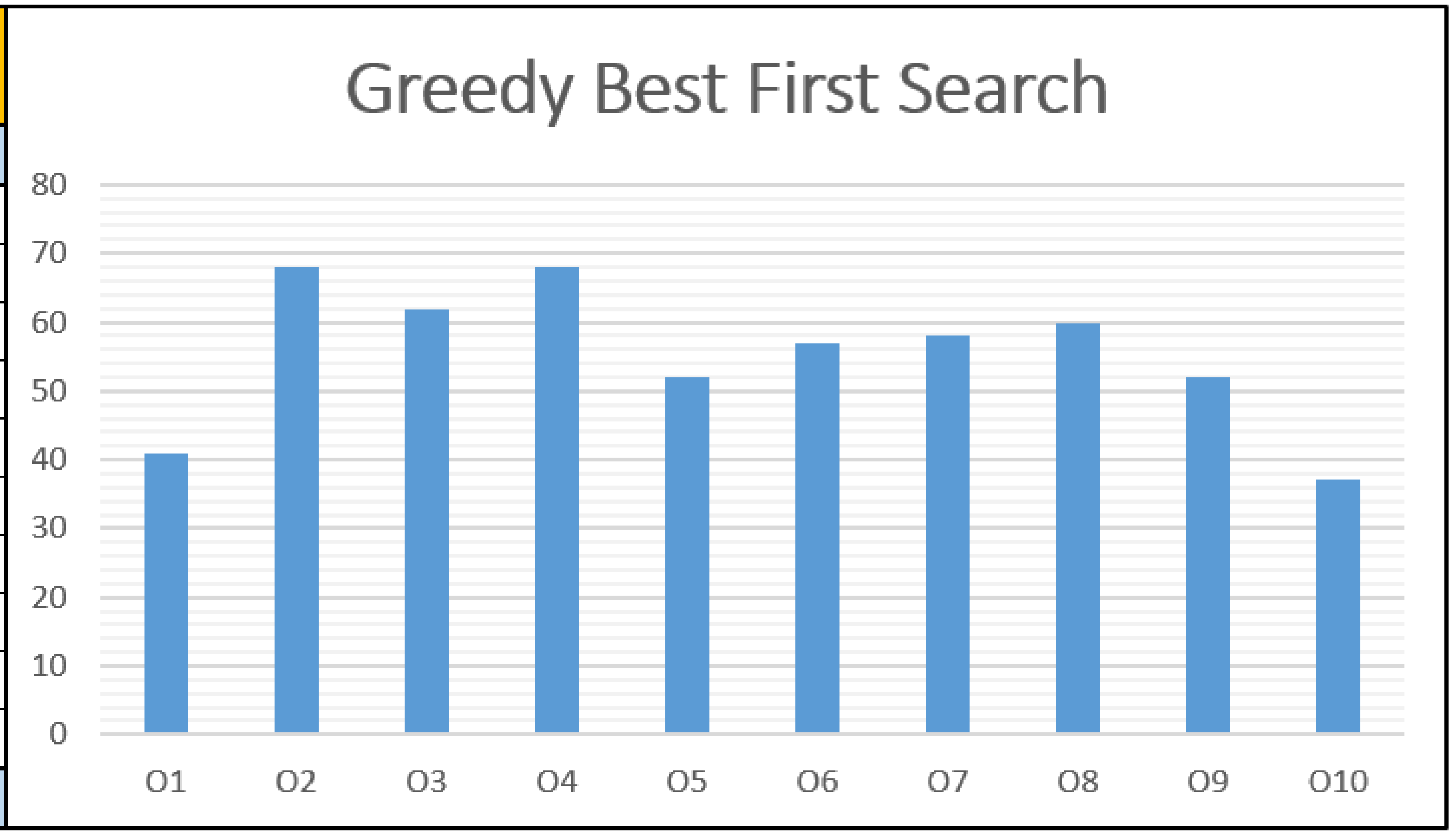
CHAPTER 5

Performance

Greedy Best First Search



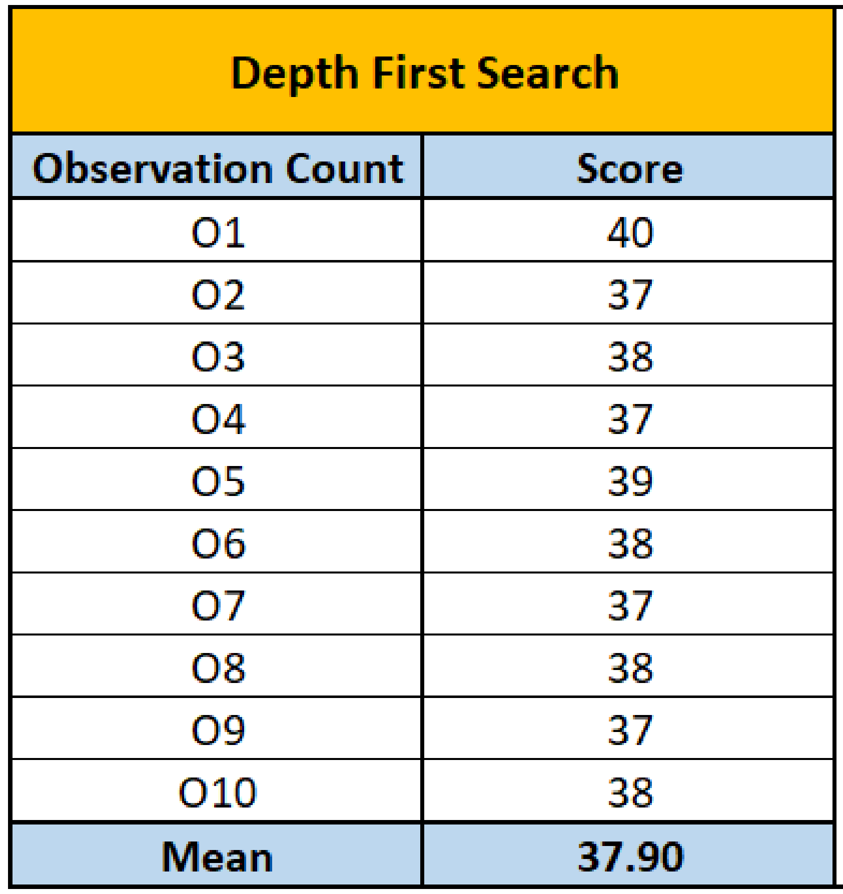
Greedy Best First Search	
Observation Count	Score
O1	41
O2	68
O3	62
O4	68
O5	52
O6	57
O7	58
O8	60
O9	52
O10	37
Mean	55.50



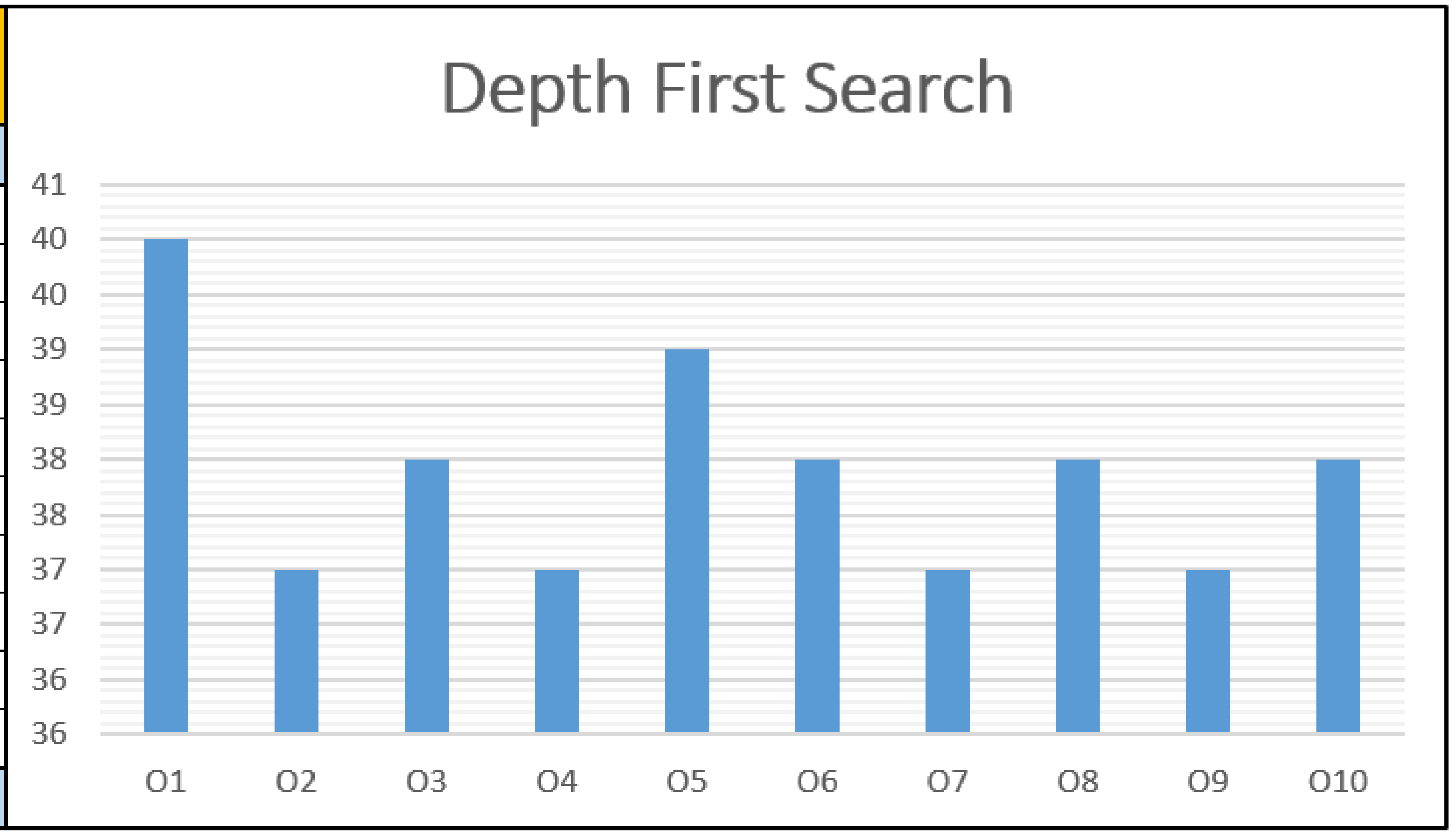
Results / Conclusion

After 10 observations, the algorithm achieved an impressive mean score of 55.5, with a peak performance reaching 68 and a low point at 37. However, the Greedy BFS failed around the mean score, where the Snake tended to trap itself by inadvertently maneuvering 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 while occasionally falling prey to itsself.

Depth First Search



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

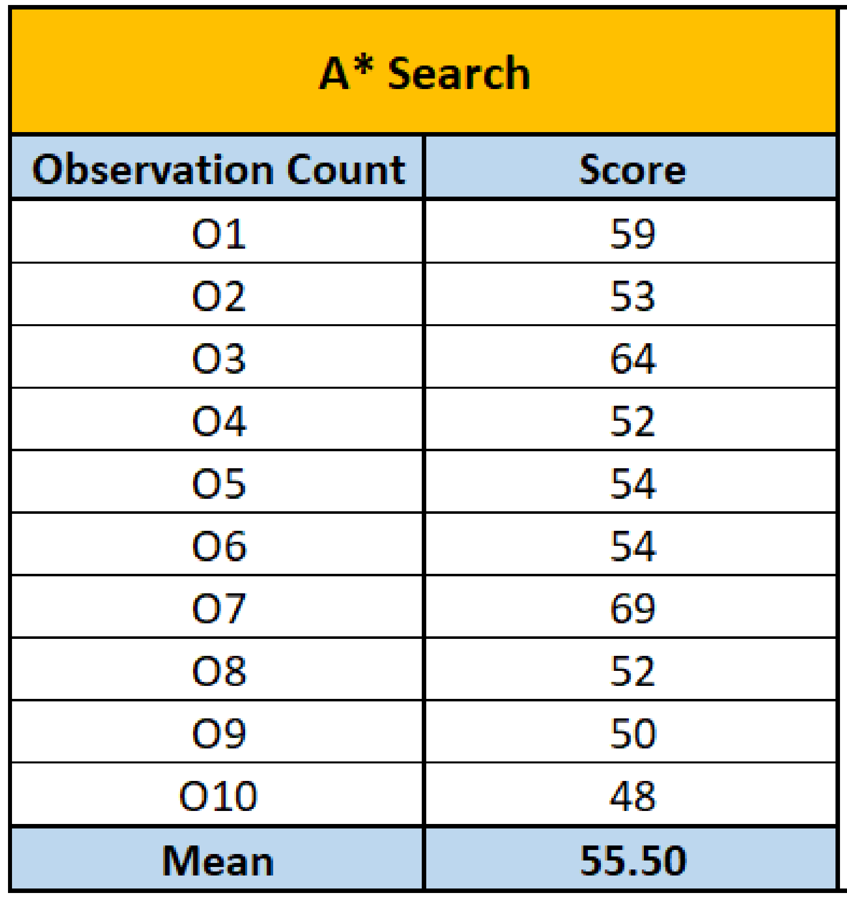


Results / Conclusion

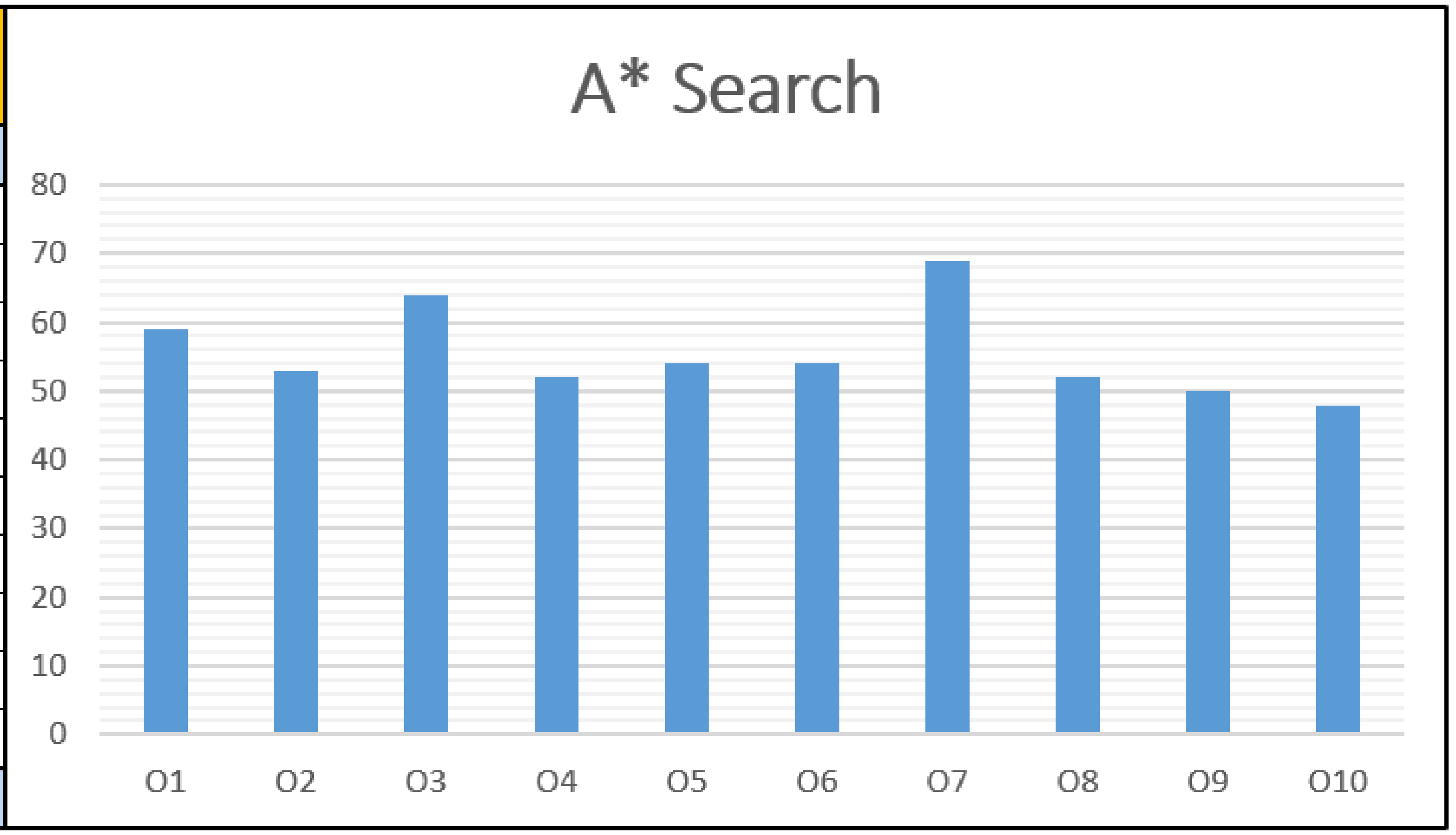
In contrast, the Depth First Search Algorithm demonstrated a different approach in our Snake game project, yielding an average score of 37.9 after 10 observations. While its highest score reached 40 and the lowest dipped to 37, the algorithm faced a critical flaw. Once the Snake extended to twice the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

CS246A

A* Search



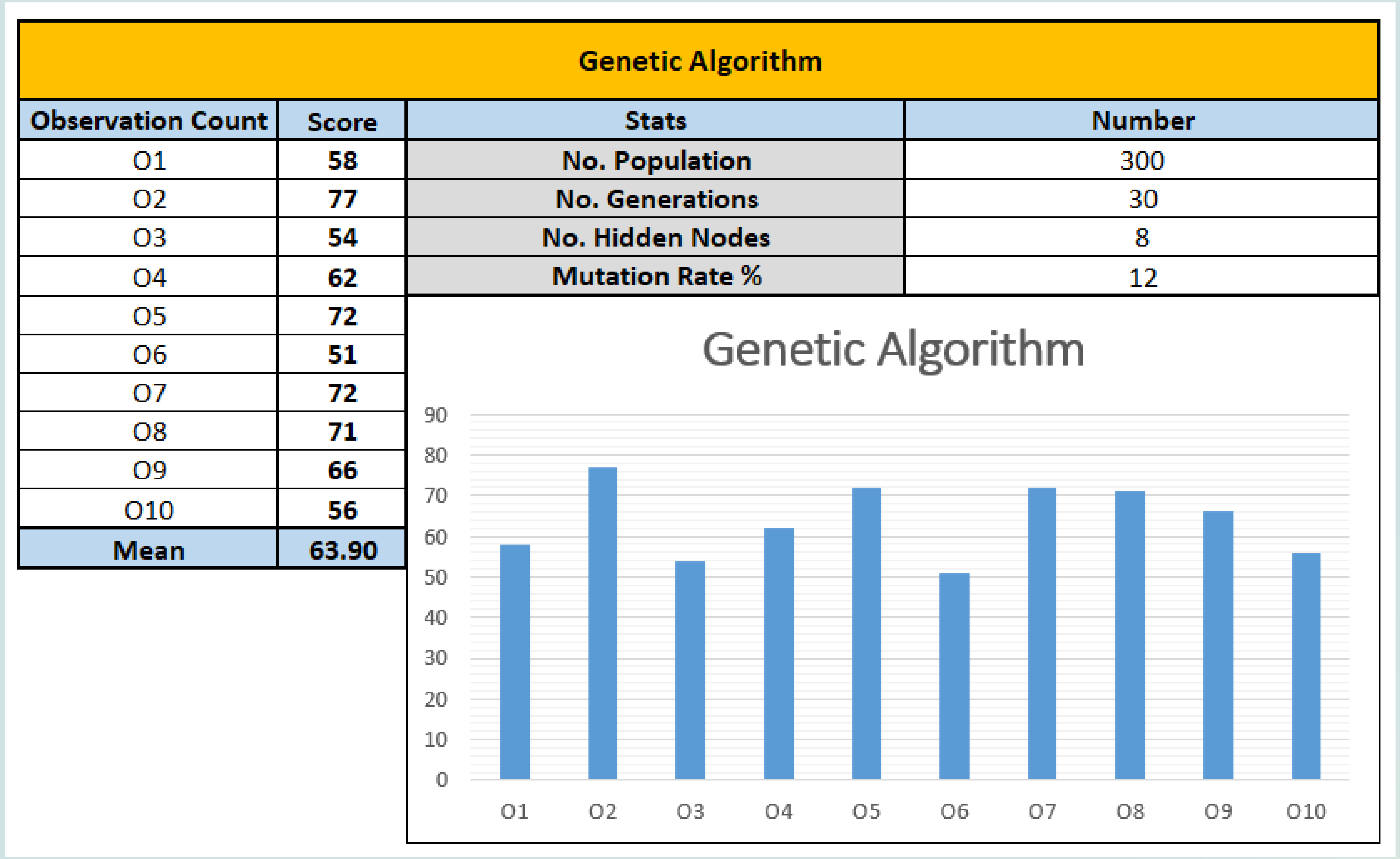
A* Search	
Observation Count	Score
O1	59
O2	53
O3	64
O4	52
O5	54
O6	54
O7	69
O8	52
O9	50
O10	48
Mean	55.50



Results / Conclusion

In our Snake game project, the A* Search Algorithm held its own, matching the Greedy BFS with an average score of 55.5 after 10 rounds. It reached a high of 69 and a low of 48, showing strong performance. However, things got tricky around the mid-50s, as the algorithm struggled with space when making turns. This highlighted the need for refining the A* Search Algorithm to navigate the Snake game's twists and turns more effectively.

Genetic Algorithm



Genetic Algorithm

Observation Count	Score
O1	58
O2	77
O3	54
O4	62
O5	72
O6	51
O7	72
O8	71
O9	66
O10	56
Mean	63.90

Stats	Number
No. Population	300
No. Generations	30
No. Hidden Nodes	8
Mutation Rate %	12

Genetic Algorithm
0
10
20
30
40
50
60
70
80
90
O1
O2
O3
O4
O5
O6
O7
O8
O9
O10

Results / Conclusion

The Genetic Algorithm outshone the others in our Snake game project, scoring an average of 63.9 over 10 rounds. It hit a high of 77 and a low of 54. The algorithm's success came from smart training with 300 members, 30 generations, 8 hidden layers, and a 12% mutation rate. The best snake from training was then tested 10 times, securing its spot as the top scorer.

Comparison

Algorithms	Average Score	Highest Score
Greedy Best First Search	55.5	68
Depth First Search	37.9	40
A* Search	55.5	69
Genetic	63.9	77

CHAPTER 6

Conclusion

Conclusion

In conclusion, our exploration of various algorithms in the Snake game project unveiled unique strengths and limitations. The Greedy BFS 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 grappled with spatial constraints during turns. However, the star of the show was the Genetic Algorithm, 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% 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.

THANK YOU!!

REFERENCES

Gaganpreet Kaur, Richa Mahajan. “Neural Networks Using Genetic Algorithms”. In: International Journal of Computer Applications (2013).
https://research.ijcaonline.org/volume77/number14/pxc3891153.pdf

Joan Aguilar Mayans, Shu Kong. “Automated Snake Game Solvers via AI Search Algorithms”. In: WPMUCDN (2016).
https://bpb-us-e2.wpmucdn.com/sites.uci.edu/dist/5/1894/files/2016/12/AutomatedSnakeGameSolvers.pdf

Chrispresso. “AI Learns to Play Snake!”. In: YouTube (2019).
https://www.youtube.com/watch?v=vhiO4WsHA6c

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://www.itsnicethat.com/features/taneli-armanto-the-history-of-snake-design-legacies-230221#:~:text=After%20launching%20in%201997%20on,its%20origin%20and%20digital%20legacy