TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. ABOUT THE GAME

Snake game is a computer action game in which the snake's goal is to eat the food continuously to grow in size and fill the map with its body without dying. The snake starts with length one. At each move or step, the snake can move forward, turn left, or turn right according to the requirement of the game. The snake cannot stop moving at any point in the game except when it is dead. An Algorithm randomly generates and places a piece of food in the game map. In order to eat the piece of food, the snake has to move towards it and the head and the snake's length grows by one. The goal is to eat as many pieces of food without ending the game by colliding the snake into itself or the walls.

The game ends when one of the two conditions are satisfied.

- The snake hits its own body or the wall.
- The Snake occupies the complete map.

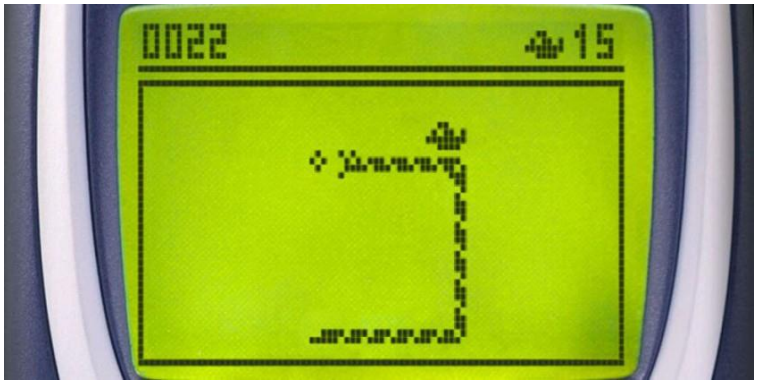


Figure 2: Classic Snake Game from Nokia

1.2. ABOUT THE PROJECT

In our project, we create an automated snake based on couple of different approaches considering smoothness, space, and food. Initially, we explain the most common AI approach used in playing the Snake game i.e. The Greedy Approach and then we explain our approach to solve the Snake Problem and determine its accuracy and other factors to compare it with The Greedy Approach.

We compare both the approaches by calculating scores for each approach. The calculated score will compromise of two important factors.

- **Success Rate**: Rate of success (i.e., the map is filled with the snake's bodies) after playing the game for N (say 100) times.
- **Average Steps**: Average steps the snake has taken to success.

In our game settings, the playing field will be 10 units tall and 10 units wide consisting of 100 available spaces. The snake will initially begin at the top-left corner, facing right, with an initial length of 2 units. Therefore, the snake can eat 98 pieces of food before filling up the entire playing field.

We have made the game more interactive by providing control override to the user whenever the user wishes to play the game him/herself. Also, the game can run on two different modes:

- **Normal Mode**: This mode runs as a single Snake game played by both user and AI without any Scores or accuracy calculations.
- **Benchmark Mode:** This mode runs 'n' iterations of the game and calculates Accuracy and scores.

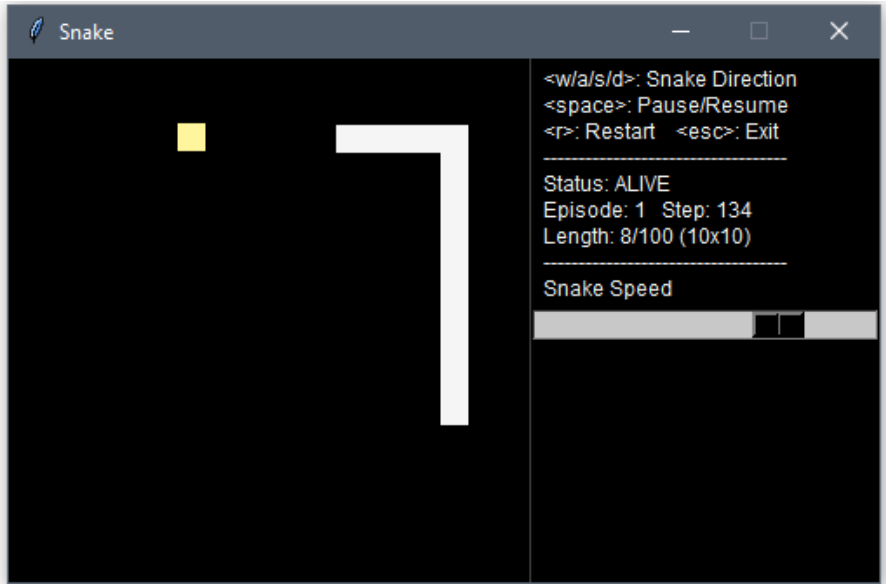


Figure 3: Our Snake Game Window

```
Map size: 10x10
Solver: greedy

Episode 1 - FULL (len: 100 | steps: 1377)

[Summary]
Average Length: 100.00
Average Steps: 1377.00
Total Runs: 1  Successful Runs: 1 (100.00%)
Avg Successful steps: 1377
```

Figure 4: Score and other Output Parameters

1.3. REQUIREMENTS

SOFTWARE:

- Python 3.5.0 or above (Preferably Spyder)

LIBRARIES:

- Random
- Collections
- Enum
- Numpy

GUI:

- TKinter (Graphic Library)

2. OUR APPROACH

We have applied 2 different approaches to tackle the Snake Problem. The Problem Statement that both the algorithms tend to solve is **"Which Direction should the Snake Go next?"**. Before We will look into both the approaches, we explain the various functionalities that both the approaches use in their algorithm.

2.1. PATH SOLVING

This section explains the common functions used by both the Approaches. It provides methods to find the shortest path and the longest path from the snake's head to other points on the game map. It does not directly decide the next moving direction of the snake but, help other solvers to work it out.

2.1.1. Shortest Path (Snake_Head, Destination):

This algorithm uses **Breadth First Search (BFS)** technique to find shortest path from Snake's head to any destination. If there are no obstructions in the map, the straightest path is generally the shortest path.

Used to find shortest path to food or tail.

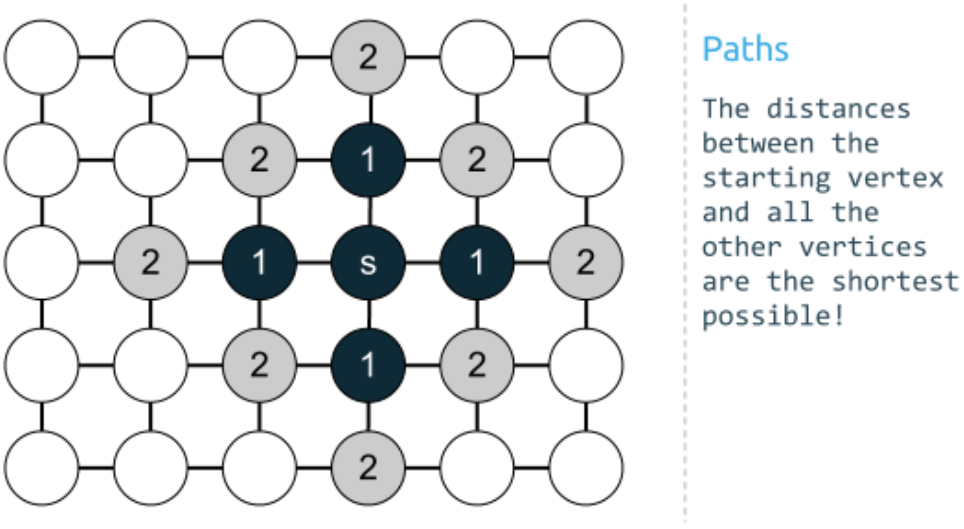


Figure 5: BFS to find Shortest Distance

2.1.2. Longest Path (Snake_Head, Destination):

Finding the longest path between two points in a directed Map is a NP-hard Problem.

Therefore, instead of finding the longest path, we can find the shortest path and extend each edge to the limit where it's no further extendable.

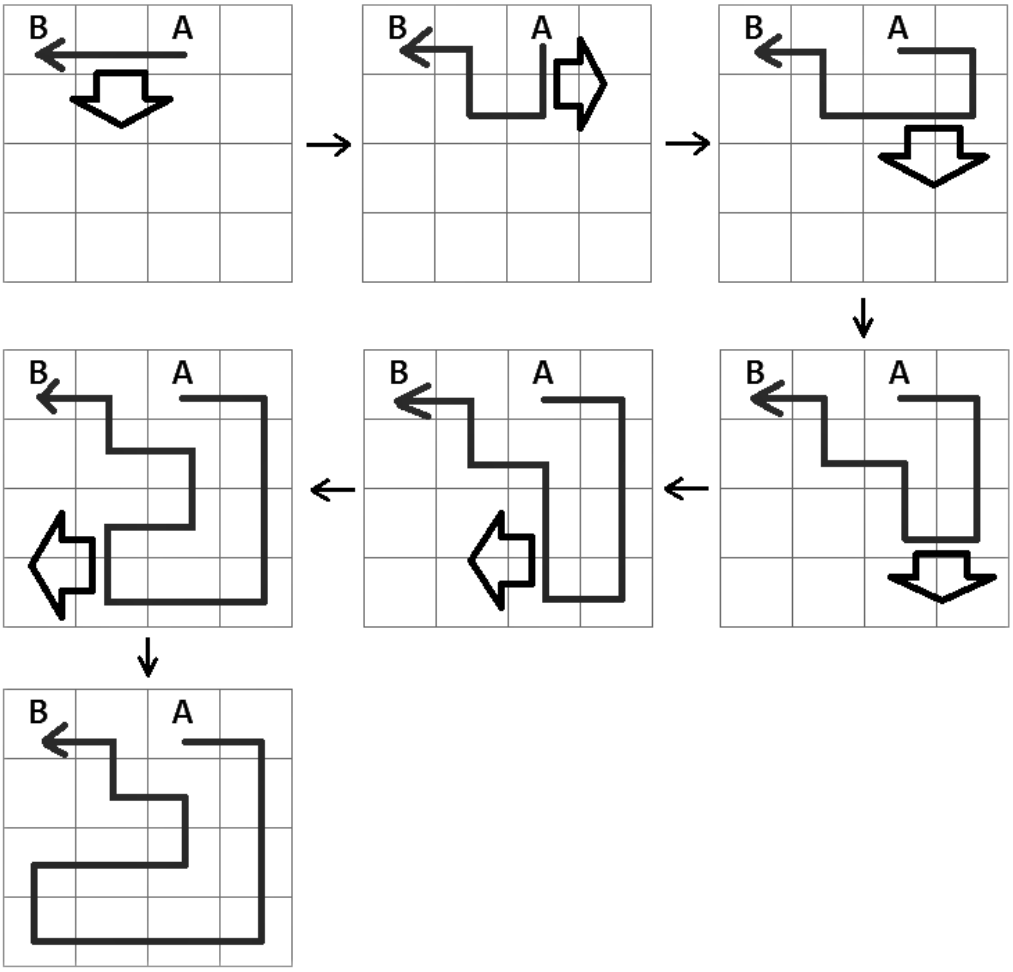


Figure 6: Longest Path finding

2.2. THE GREEDY APPROACH

Greedy Approach, as the name suggests, always looks to find the shortest path to move towards the food if it exists and if it's safe. If at the current step, there is no path towards the food, Snake will move in direction of the longest distance to the food. To check if moving in direction of the food is safe, the algorithm creates a Copy Snake which goes and virtually eats the food. Once, its has eaten the food it has to see if it has a path out of there. The best way to find this is to chase its own tail, if it exists. If it exists, copy snake moves towards the longest path to its own tail. Otherwise, main snake will Conclude that it's not safe to go for food and copy snake will be removed.

Even if it's not possible to move towards the food, the snake will still have to make some move. If there is a path from Snake's head to its tail, then it will move in direction of its tail. Otherwise, it will move in the direction opposite of the food.

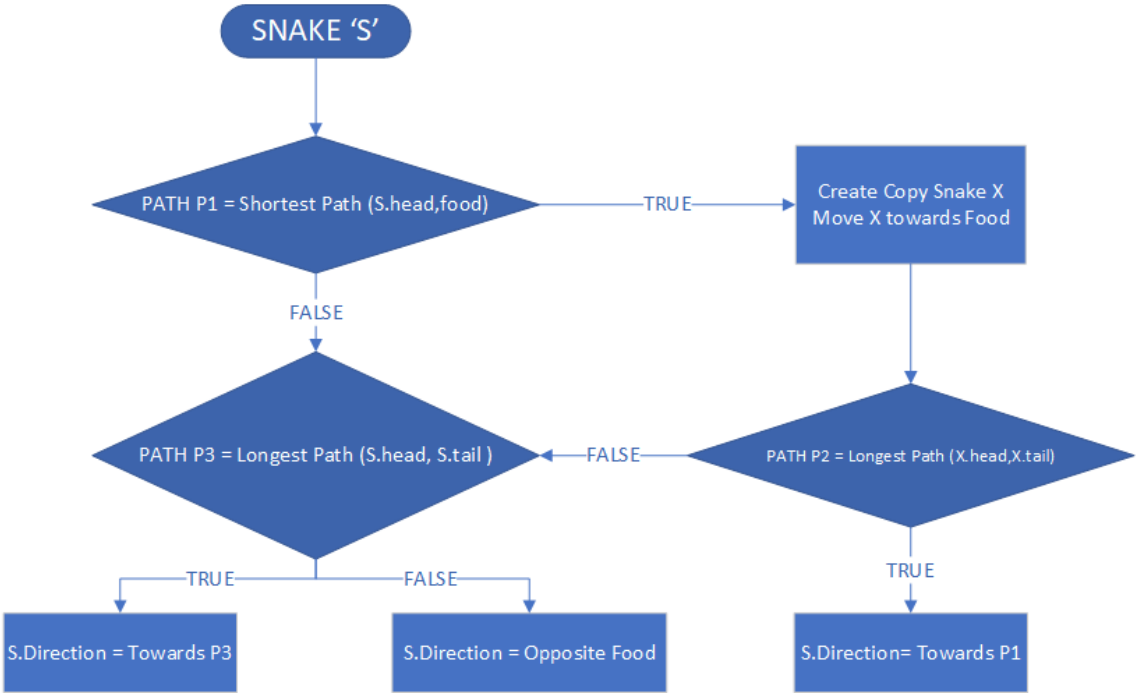


Figure 7: Greedy Approach Flowchart

2.3. THE HAMILTON APPROACH

Hamiltonian Approach directs the Snake towards a well-defined path along the Hamiltonian Cycle and eat the food along the path of Hamiltonian Cycle.

2.3.1. HAMILTONIAN CYCLE

A Hamiltonian cycle, also called as Hamilton cycle, or Hamilton circuit, is a graph cycle (i.e., closed loop) through a graph that visits each node exactly once. A Planar Hamiltonian cycle is a Path such that there is an edge (in graph) from the last vertex to the first vertex of the Hamiltonian Path. This means that there will always be a path from the

Snake's head to its tail. Since, there is always a path from its head to its own tail, it can never collide or bite itself. To build a Hamiltonian cycle, path indexes are assigned to each point on the map such that the last index and the first index are adjacent to each other.

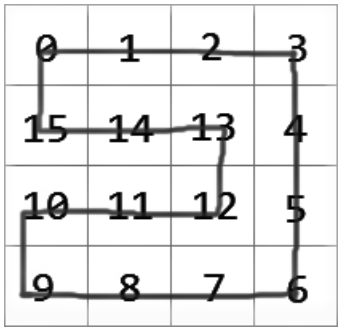


Figure 8: Example of Hamiltonian Cycle on a 4X4 Map

Figure 7 shows a Hamiltonian Cycle on a 4X4 Map. To construct the cycle above, first we fix the point 0, 1 and 2 (considering the snake's initial head and bodies are 2, 1, 0, respectively). Then we make point 1 unreachable and generate the longest path from point 2 to point 0. Finally, we join the starting point 2 and the ending point 0, which forms a Hamiltonian cycle:

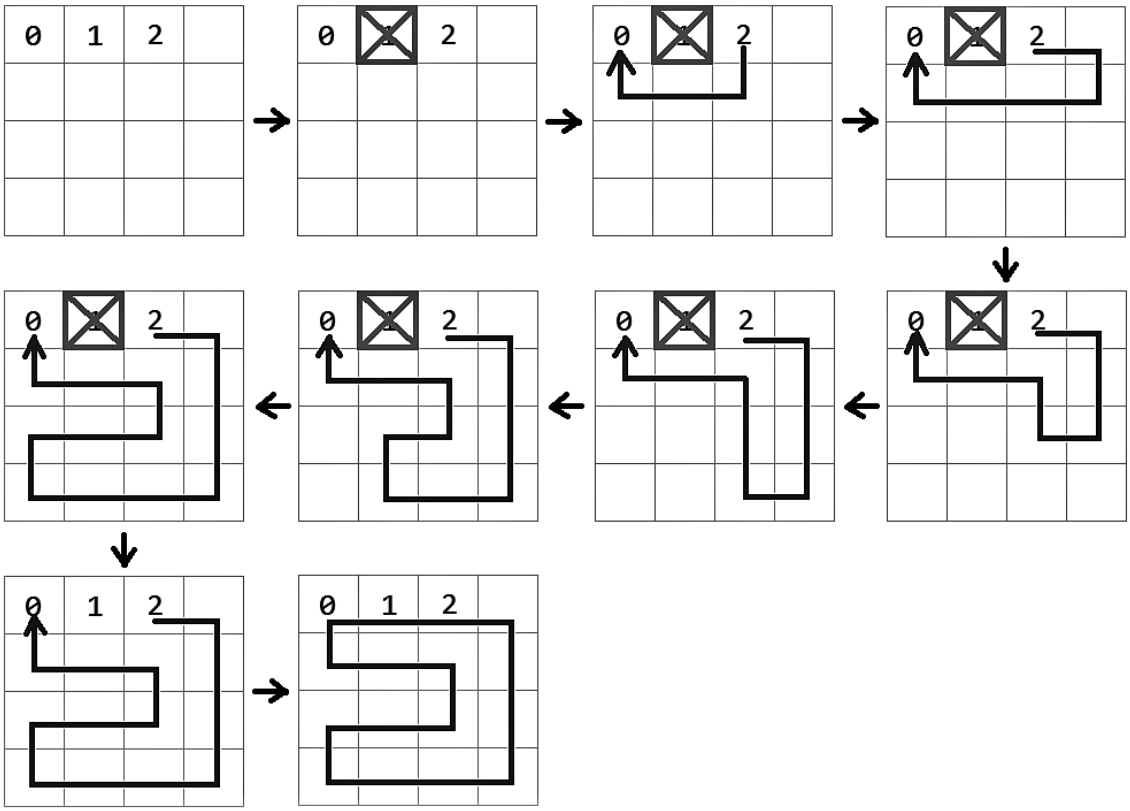


Figure 9: Prim's Algorithm of Building Hamiltonian Cycle

2.4. ENHANCED HAMILTON APPROACH

Clearly, using the Hamilton approach will lead to large number of steps but in turn will provide 100 % accuracy as the snake always follows the same path. However, this results in a boring Snake game with the snake instead of following the food simply follows the predefined Hamiltonian path.

To tackle this issue, we have Modified the Hamilton Approach in such a way that primarily it follows the Hamiltonian Cycle but tries to find the shortest path to the food if it does not disturb the predefined path i.e. the snake can reorganize itself into the predefined Path. We call this Approach as **'Enhanced Hamiltonian Approach'.**

According to this improved Approach, we represent the 2 Dimensioned Hamiltonian Cycle as 1 Dimension array (Path) showing parts of snake at their actual indexes. The key to success of this algorithm is that indexes of Body part of snake should always lie between the Tail and Head indexes.

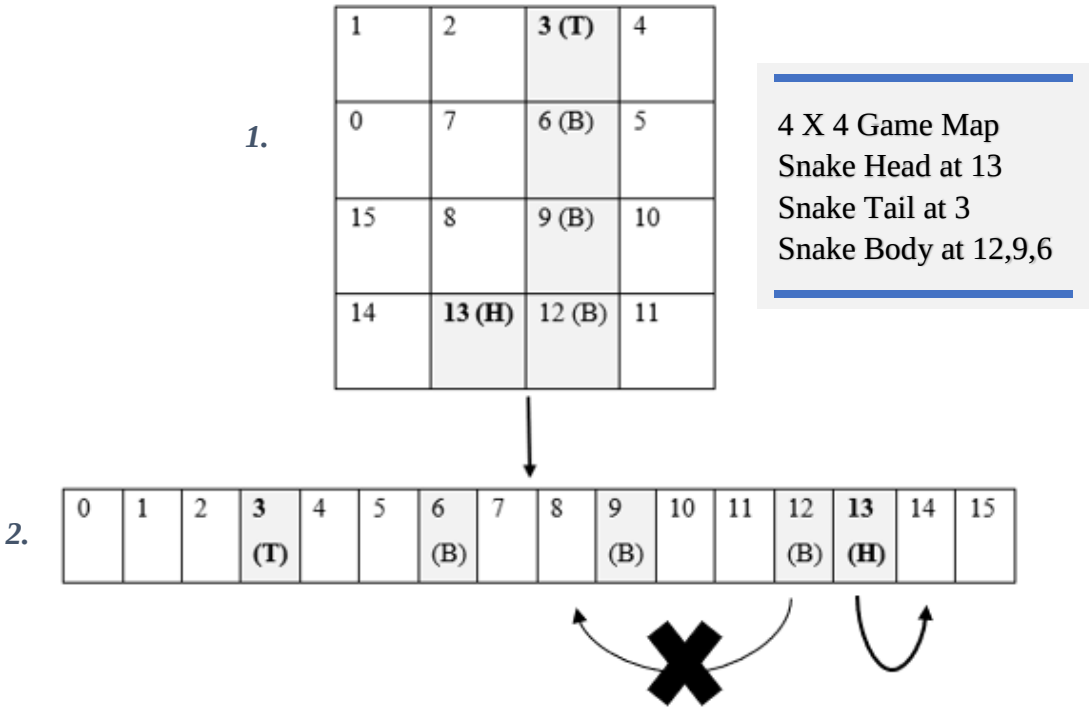


Figure 10: Enhanced Hamilton Approach

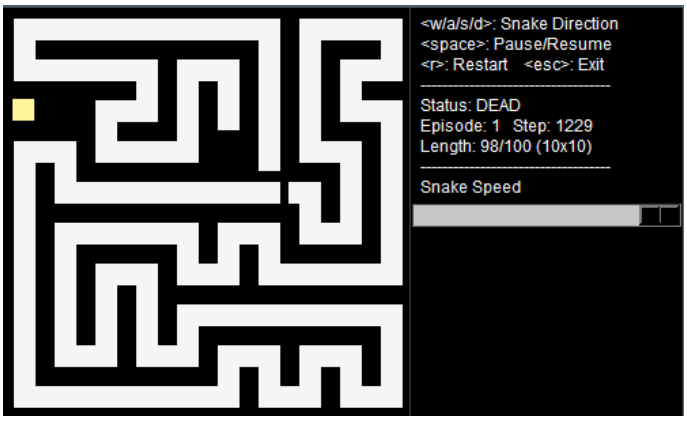
Figure 9.1 shows a 4X4 Game Map with Snake's head at 13, Snake's Tail at 3, and Body at (12, 9, 6). Snake can move to either 14 or 8 in the next step.

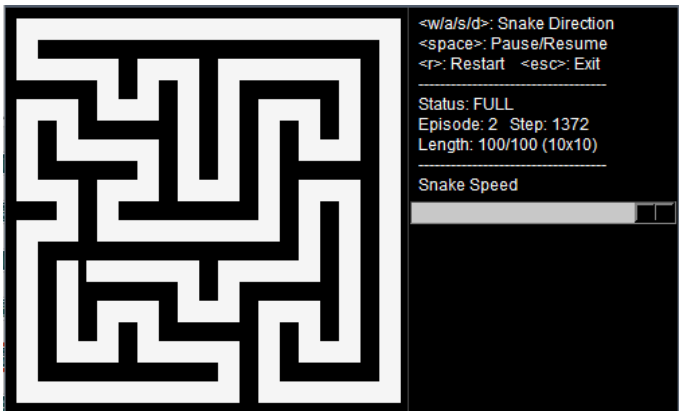
Figure 9.2 show the possible options for the snake to move and explains the direction in which the snake goes. In the 1-D representation of the map if the snake has to cross any of its body part, it will lead to a current or future unsafe situation as the Body will then not lie between Head and Tail. Therefore, Snake's head will move 14 as moving in that direction keeps the Body in between Head and Tail.

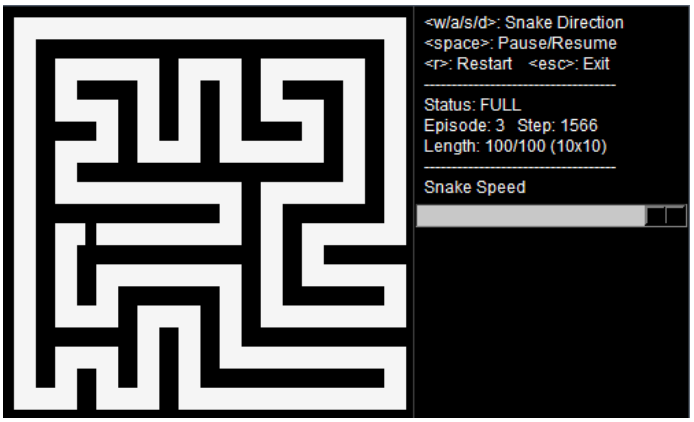
3. RESULTS

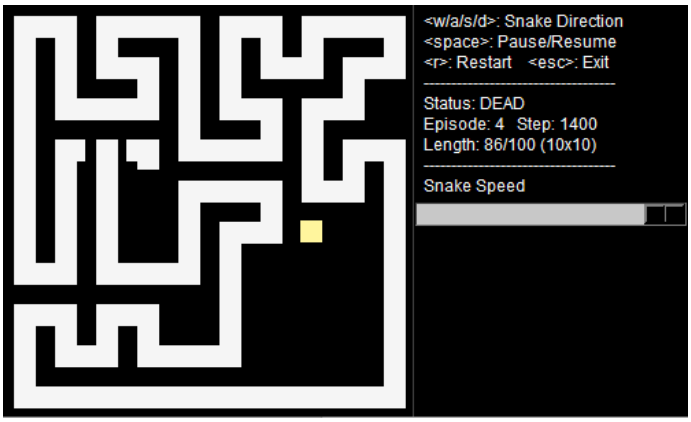
In this Section, we will present Screenshots and Output results of our Project. We will show Benchmarking result for both Greedy as well as Enhanced Hamilton Approach for 5 iterations and compare the results.

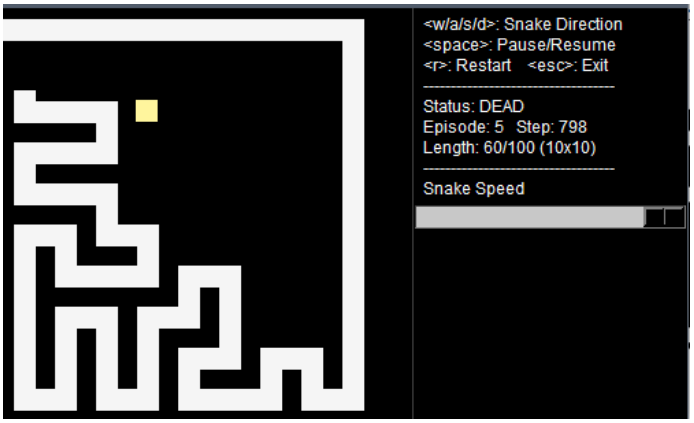
3.1. GREEDY APPROACH







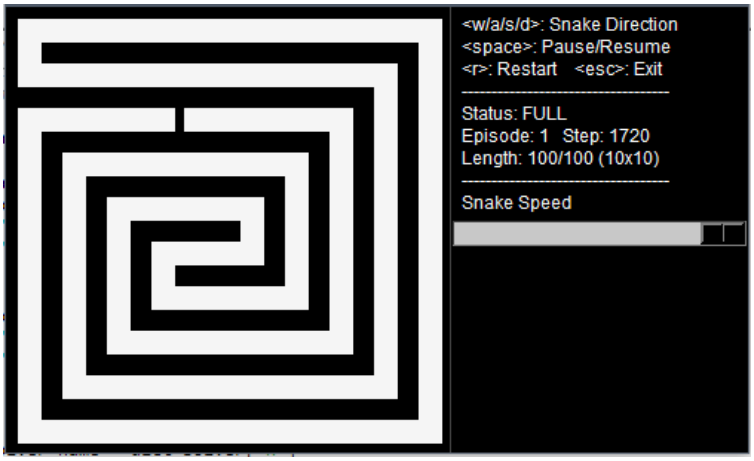


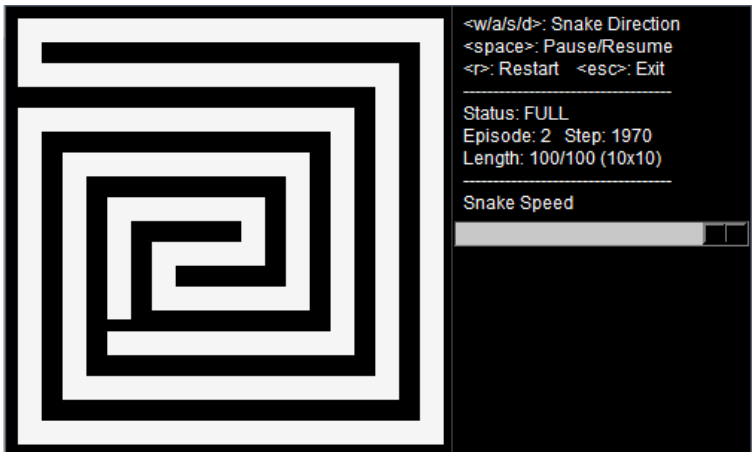


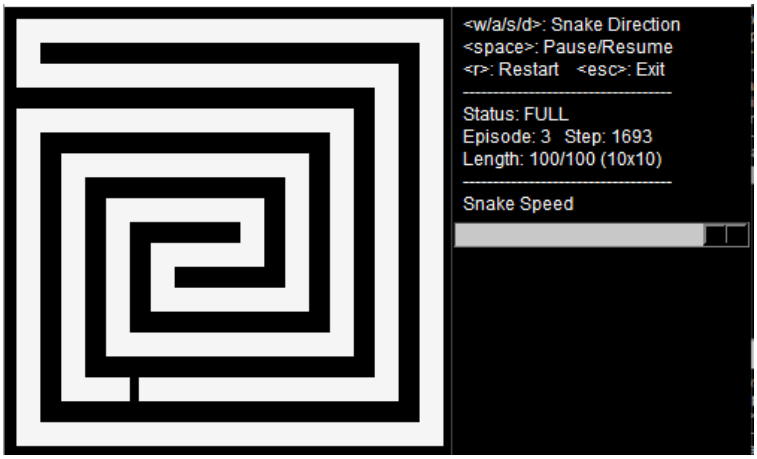

```
[Summary]
Average Length: 88.80
Average Steps: 1273.00
Total Runs: 5  Successful Runs: 2 (40.00%)
Avg Successful steps: 1469
```

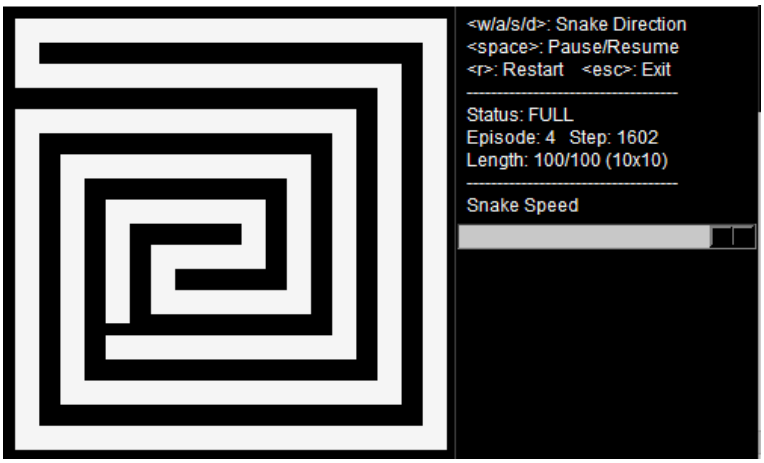
Figure 11: Greedy Approach Benchmarks

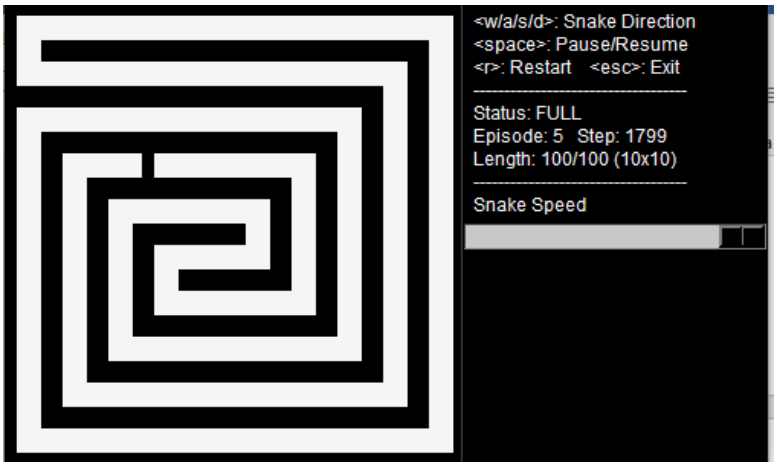
3.2. ENHANCED HAMILTON APPROACH












```
[Summary]
Average Length: 100.00
Average Steps: 1756.80
Total Runs: 5  Successful Runs: 5 (100.00%)
Avg Successful steps: 1756
```

Figure 11: Enhanced Hamilton Approach Benchmarks

4. CONCLUSION

This project implements and compares two different type of Approaches to Solve the Classic Snake Game Problem. The first approach, Greedy Approach, directs the snake to eat the food along the shortest path if it thinks the snake will be safe. Otherwise, it makes the snake wander around until a safe path can be found. The second approach, Enhanced Hamilton Approach builds a Hamiltonian cycle on the game map first and then directs the snake to eat the food along the cycle path. To reduce the average steps the snake takes to success, it enables the snake to take shortcuts if possible.

We compare both the approaches by calculating scores for each approach. The calculated score will compromise of two important factors.

Success Rate: Rate of success (i.e., the map is filled with the snake's bodies) after playing the game for N (say 100) times.
Average Steps: Average steps the snake has taken to success.

Our Comparison shows Hamilton Approach (100%) is clearly more accurate and successful than the Greedy Approach (40%). On the other hand, Total Average Steps taken in a successful Iteration shows that Whenever the Greedy Approach has been successful (Avg. Successful Steps = 1469), whereas, for Hamilton Approach (Avg. Successful Steps = 1756).

It is clear from the Comparison that our Enhanced Hamilton Approach is clearly more Accurate and Successful than Greedy Approach, but Success comes at the Cost of time Taken.

5. REFERENCES

[1] Yeh, Jia-Fong, et al. "Snake game AI: Movement rating functions and evolutionary algorithm-based optimization." *Technologies and Applications of Artificial Intelligence (TAAI), 2016 Conference on*. IEEE, 2016.

[2] https://johnflux.com/2015/05/02/nokia-6110-part-3-algorithms/