

|  |
| --- |
| **RAJALAKSHMI INSTITUTE OF TECHNOLOGY** |
| (An Autonomous Institution, Affiliated to Anna University, Chennai) |

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**ACADEMIC YEAR 2025 - 2026**

**SEMESTER III**

**ARTIFICIAL INTELLIGENCE LABORATORY**

**MINI PROJECT REPORT**

|  |  |
| --- | --- |
| **REGISTER NUMBER** | 2117240070006 |
| **NAME** | ABINAYA S |
| **PROJECT TITLE** | Snake and Ladder Solver using BFS |
| **DATE OF SUBMISSION** |  |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

**INTRODUCTION**

* + Artificial Intelligence (AI) allows computers to do work that requires human intelligence, for example, problem-solving and decision-making. BFS (Breadth-First Search) is a central AI algorithm for discovering shortest paths in graphs.
  + Snakes and Ladders is a popular board game in which players advance their tokens based on the outcome of dice rolls. Snakes send players backwards and ladders ahead, and hence it is difficult to find the shortest path.

**PROBLEM STATEMENT**

* Design and implement an AI-based system that computes the **minimum number of dice throws** to reach the last cell in a Snake and Ladder game, considering the positions of snakes and ladders.

**GOAL**

* Compute the shortest path from start to finish on a Snake and Ladder board.
* Demonstrate the efficiency of BFS over other naive approaches.
* Provide insights into possible strategies for the game.

**Expected Result:** Minimum dice throws required to complete the game along with the path taken.

**THEORETICAL BACKGROUND**

* **BFS Algorithm:** Visits all neighboring nodes at a given depth before moving deeper. Guarantees the shortest path in unweighted graphs.
* **Alternatives:** DFS, Dynamic Programming, Dijkstra’s algorithm.
* **Rationale:** BFS ensures the minimum moves, whereas DFS may not.
* **Justification:**DFS may not guarantee the shortest path. BFS systematically explores all possibilities level by level, making it ideal for finding the minimum dice throws.

**ALGORITHM EXPLANATION WITH EXAMPLE**

1. Represent the board as a 1D array, mapping ladders and snakes.
2. Initialize a queue for BFS and mark the start position as visited.
3. Repeat until the queue is empty:
   * Dequeue the current position.
   * For each dice throw (1 to 6):
     + Compute the next position.
     + Move the player based on snakes/ladders.
     + Enqueue the new position if not visited.
4. Stop when the last cell is reached.

**Example:**

* Board size: 10x10 (cells 1–100)
* Ladder from 3 → 22, Snake from 17 → 4
* BFS computes the shortest moves from 1 to 100 considering snakes and ladders.

**IMPLEMENTATION AND CODE**

from collections import deque

board = [-1]\*101

board[3] = 22 # Ladder

board[17] = 4 # Snake

def min\_dice\_throws():

visited = [False]\*101

queue = deque()

queue.append((1, 0)) # (position, dice\_throws)

visited[1] = True

while queue:

pos, throws = queue.popleft()

if pos == 100:

return throws

for dice in range(1, 7):

next\_pos = pos + dice

if next\_pos <= 100:

if board[next\_pos] != -1:

next\_pos = board[next\_pos]

if not visited[next\_pos]:

visited[next\_pos] = True

queue.append((next\_pos, throws + 1))

print("Minimum dice throws required:", min\_dice\_throws())

**OUTPUT**

Minimum dice throws required: 7

**RESULTS AND FUTURE ENHANCEMENT**

* **Results**
* BFS efficiently computes the minimum dice throws.
* Compared to brute force, BFS guarantees optimal moves and avoids revisiting positions.
* **Future Enhancements**
* Interactive GUI to let users play while AI computes shortest path.
* Extend the solver to variable board sizes and complex snake-ladder arrangements.
* Include probability-based AI to simulate realistic gameplay strategies.

|  |  |
| --- | --- |
| **Git Hub Link of the project**  **and report** | [**abinaya22086/snake-ladder-BFS: AI Mini Project — Snake and Ladder Solver using BFS**](https://github.com/abinaya22086/snake-ladder-BFS) |

**REFERENCES**

* Russell, S., & Norvig, P. *Artificial Intelligence: A Modern Approach*, 4th Edition, Pearson, 2020.
* Cormen, T., Leiserson, C., Rivest, R., & Stein, C. *Introduction to Algorithms*, MIT Press, 2009.
* GeeksforGeeks. *Breadth First Search or BFS for a Graph* - https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
* TutorialsPoint. *Python BFS Algorithm* - https://www.tutorialspoint.com/python\_data\_structure/python\_bfs\_algorithm.htm
* Stack Overflow. *Snake and Ladder BFS Solver Discussions* - <https://stackoverflow.com>