Artificial Intelligence Project Based Learning (PBL) Report on

Maze Solver

Submitted in partial fulfilment of the

Requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science and Engineering (AI&ML)

Submitted by

Y.Balchander Reddy	***22R15A6605***
A.Achuta Reddy	***21R11A6601***
B.Gowtham Chandra	***21R11A6607***

Under the esteemed guidance of

Mr. Shaik Akbar

Associate Professor, CSE(AI&ML) Department

Geethanjali College of Engineering and Technology

Geethanjali College of Engineering and Technology

Department of Computer Science and Engineering (AI&ML)

(UGC AUTONOMOUS INSTITUTION)

Accredited by NAAC with ‘A+’ Grade & NBA, Approved by AICTE and Affiliated to JNTUH

Cheeryal (V), Keesara (M), Medchal (Dist), Telangana – 501 301.

JANUARY- 2024

Geethanjali College of Engineering and Technology

Department of Computer Science and Engineering (AI&ML)

(UGC AUTONOMOUS INSTITUTION)

Accredited by NAAC with 'A+' Grade & NBA, Approved by AICTE and Affiliated to JNTUH

Cheeryal (V), Keesara (M), Medchal (Dist), Telangana – 501 301.

JANUARY- 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)

CERTIFICATE

This is to certify that the Artificial Intelligence Project Based Learning (PBL) Report entitled **"Maze Solver"** is a bonafide work done and submitted by **A.Achuta Reddy(21R11A6601),B.Gowtham Chandra (21R11A6607), Y.Balchander Reddy (22R15A6605)** during the academic year 2023 – 2024, in partial fulfilment of requirement for the award of Bachelor of Technology degree in **"Computer Science and Engineering (AI&ML)"** from Geethanjali College of Engineering and Technology (Accredited by NAAC with 'A+' Grade & NBA, Approved by AICTE and Affiliated to JNTUH), is a bonafide record of work carried out by them under guidance and supervision.

Certified further that to my best of the knowledge, the work in this project has not been submitted to any other institution for the award of any degree or diploma.

FACULTY GUIDE
Mr. Shaik Akbar
Associate Professor
CSE(AI&ML) Department

DECLARATION

We hereby declare that the project report entitled **“Rat Maze Solver”** is an original work done and submitted to CSE(AI&ML) Department, from Geethanjali College of Engineering and Technology (Accredited by NAAC with ‘A+’ Grade & NBA, Approved by AICTE and Affiliated to JNTUH), in partial fulfilment of the requirement for the award of Bachelor of Technology in **“Computer Science and Engineering (AI&ML)”** and it is a record of bonafide project work carried out by us under the guidance of **Mr. Shaik Akbar, Associate Professor, Department of CSE(AI&ML)**.

Signature of the Student
A.Achuta Reddy
(21R11A6601)

Signature of the Student
B.Gowtham Chandra
(21R11A6607)

Signature of the Student
Y.Balchander Reddy
(22R15A6605)

ACKNOWLEDGEMENT

This satisfaction of completing this project would be incomplete without mentioning our gratitude towards all the people who have supported us. Constant guidance and encouragement have been instrumental in the completion of this project.

First and foremost, we thank the chairman, Principal, Vice Principal for availing infrastructural facilities to complete the project in time.

We offer our sincere gratitude to our faculty guide **Mr. Shaik Akbar,** Associate Professor, CSE(AI&ML) Department, from Geethanjali College of Engineering and Technology (Accredited by NAAC with 'A+' Grade & NBA, Approved by AICTE and Affiliated to JNTUH), for his immense support, timely co-operation and valuable advice throughout the course of our project work.

We are thankful to all the project Coordinators for their supportive guidance and for having provided the necessary help for carrying forward this project without any obstacle and hindrances.

ABSTRACT

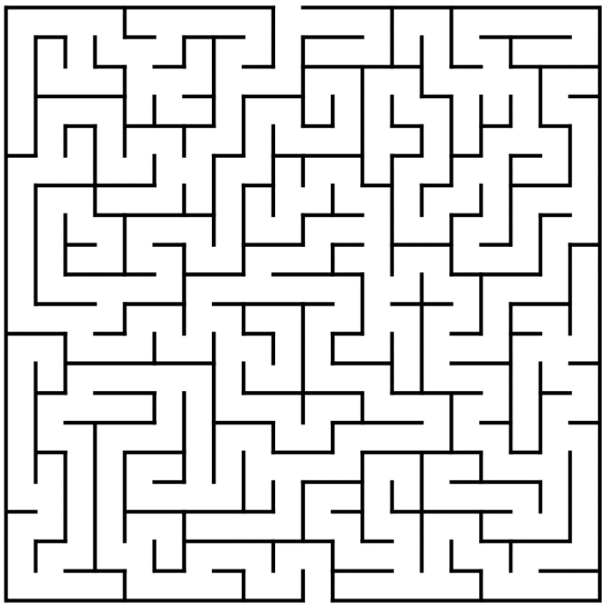
Mazes, characterized by intricate paths and dead ends, serve as fertile ground for investigating algorithms and strategies that enable efficient navigation. Researchers explore diverse methodologies, including heuristic search algorithms, machine learning techniques, and computational intelligence, to develop solutions that optimize pathfinding, minimize traversal time, and enhance decision-making. One key area of focus is the utilization of algorithms like Dijkstra's, A*, and various variants of depth-first and breadth-first searches to determine the shortest paths within mazes. The pointer or the way is programmed in a way that it identifies the blocks and the walls to avoid them and reaches the goal state.

Keywords: Maze, decision making, Dijkstra's algorithmm, A algorithm*

TABLE OF CONTENTS

1. INTRODUCTION

Maze Solver, a witching brain game, challenges suckers to unravel intricate paths within a square-structured block adorned with dead- ends and obstructive lines, all in pursuit of relating the most optimal route to reach a designated thing. The beauty of this game lies in its versatility, allowing the starting point to be deposited anywhere within the maze, be it at its core, its fringe, or any other strategic position. The ideal remains constant- to navigate through the complicated network and successfully pinpoint the destination. In the realm of artificial intelligence (AI), working the Maze Solver problem serves as a foundational exercise. The complications of this task make it a definitive standard for developing algorithms that pretend intelligent decision- making processes. numerous real- world scripts, from mapping technologies to robotics, draw alleviation from the Maze Solver problem to address challenges related to effective pathfinding and traversal optimization. In this pursuit of maze-working mastery, our design employs a sophisticated algorithm known as A *. A * stands out as a foundation in the sphere of Artificial Intelligence, famed for its effectiveness in navigating trees and graphs to uncover the optimal path to the thing. Unlike traditional hunt algorithms, A * incorporates a heuristic evaluation to intelligently guide its disquisition, icing that the algorithm not only finds a path but identifies the most effective bone . A * harnesses the power of heuristic values to estimate the cost of reaching the thing from any given point within the maze. This innovative approach enhances the algorithm's capability to make informed opinions, prioritizing paths that show pledge in terms of effectiveness. The use of A * in the environment of Maze Solver not only adds a subcaste of intelligence to the navigation process but also showcases the practical operations of AI algorithms in problem- working scripts. As we claw into the complications of A * and its operation in Maze Solver, we embark on a trip that not only challenges our problem- working chops but also highlights the symbiotic relationship between artificial intelligence and maze- working methodologies. Through this disquisition, we aim to unravel the eventuality of A * as a protean tool in creating intelligent results for pathfinding mystifications, setting the stage for a deeper understanding of AI operations in different fields.



1.2 Existing Implementations

- **Depth-First Search (DFS):** DFS is a simple algorithm that explores as far as possible along each branch before backtracking. In maze-solving, DFS can be used to explore different paths until a solution is found.
- **Breadth-First Search (BFS):** BFS explores all the neighbor nodes at the present depth prior to moving on to nodes at the next depth level. In maze solving, BFS can find the shortest path to the solution.

1.3 Limitations of Existing Implementations

Complexity of Maze Structure:

- Many maze-solving algorithms and robotic systems may struggle when faced with highly complex or irregular maze structures. Some algorithms may take a long time to find a solution or may not be able to handle certain types of mazes efficiently.

DFS:

Depth-First Search (DFS) is a simple and intuitive algorithm, but it has some limitations when used for maze solving. Here are a few drawbacks of using DFS:

Completeness:

DFS does not guarantee finding the shortest path to the goal. It may find a solution quickly, but this solution may not be optimal in terms of the number of steps or cost.

Memory Usage:

- In some cases, DFS can use a lot of memory, especially in large and complex mazes. This is because it needs to store the entire path from the start to the current node on the stack, which can lead to a stack overflow in certain situations.

BFS:

While Breadth-First Search (BFS) is a powerful algorithm for certain types of problems, including maze solving, it also has some limitations. Here are the drawbacks of using BFS for maze solving:

Memory Usage:

- BFS explores all nodes at a given depth level before moving on to the next level. In some cases, this can lead to high memory usage, especially in large mazes, as all nodes at a particular level need to be stored in memory.

1.4 Advantages of Proposed System

Optimality:

- A* guarantees finding the optimal path from the start to the goal if certain conditions are met. The optimality is achieved by using a heuristic function that estimates the cost of reaching the goal from a given state. If the heuristic is admissible (never overestimates the true cost), A* will always find the shortest path.

Efficiency:

- A* is often more efficient than uninformed search algorithms like Depth-First Search (DFS) or Breadth-First Search (BFS) when it comes to finding the shortest path. Its informed nature allows it to focus on more promising paths, reducing the exploration of unnecessary nodes.

Memory Efficiency:

- While A* may use more memory compared to some uninformed search algorithms, it is generally more memory-efficient than exhaustive search methods like DFS in most cases. Proper implementations can mitigate memory concerns.

2.AIM and OBJECTIVES

2.1 Aim of Proposed Project

The aim of a rat in a maze project is to create an algorithm or program that simulates a rat navigating through a maze to find its way to a specific destination or exit point. The project typically involves designing the maze, defining the rules for movement and pathfinding, and implementing the algorithm to guide the virtual rat through the maze. The project aims to showcase problem-solving and algorithmic thinking skills by creating an efficient and effective solution for the rat to reach its destination. The primary aim of a maze solver is to find the shortest path or a valid path from a designated start point to a target or goal point within a maze. Maze-solving algorithms are designed to navigate through a maze's corridors while avoiding obstacles or blocked paths, optimizing for the most efficient route. The goal is to automate the process of finding a solution to mazes, which can be time-consuming and challenging when done manually.

2.2 Objectives of Proposed Project

The objective of a rat in a maze project is to develop a program or algorithm that enables a rat to find its way from a starting point to a designated exit point in a maze.

The specific objectives may include:
1. Designing or generating a maze with various obstacles, walls, dead-ends, and corridors.
2. Implementing a systematic approach such as depth-first search, breadth-first search, or A* algorithm to determine the rat's path through the maze.
3. Ensuring that the rat follows a valid and efficient path towards the exit, avoiding obstacles and dead-ends.
4. Displaying the maze and the rat's movement visually, allowing for interactive user input or automated movement.
5. Providing feedback on the number of steps taken, time taken, or the shortest path found.
6. Allowing for customization of maze size, complexity, and rat's behavior.
7. Testing and evaluating the algorithm's performance based on factors like time complexity, space complexity, or the number of steps taken.
8. Optionally, implementing additional features such as multiple possible exits, maze generation algorithms, or different rat movement strategies.

Overall, the objective is to demonstrate problem-solving, logic, and algorithmic skills by creating a working solution for a rat to navigate a maze efficiently and successfully reach the exit point.

3.BACKGROUND

3.1 Background of Maze Solver

The background of maze solvers can be traced back to the field of graph theory and computer science.

One of the earliest and most well-known algorithms for solving mazes is the depth-first search (DFS) algorithm. DFS operates by exploring each path in the maze until it reaches a dead-end before backtracking and exploring the next available path. Although DFS is relatively simple to implement, it does not always find the shortest path to the exit.

Another common algorithm used for maze solving is the breadth-first search (BFS) algorithm. BFS explores all possible paths in a maze level by level, ensuring that the shortest path is found. BFS is generally considered more accurate than DFS in terms of finding the shortest path, but it may require more computational resources.

In addition to DFS and BFS, other more advanced algorithms like the A* algorithm and Dijkstra's algorithm can also be used to solve mazes. These algorithms use heuristics and prioritize certain paths based on their estimated cost or distance to the exit, allowing for more efficient and optimized maze solving.

Maze solvers serve as a practical application of algorithmic thinking and problem-solving skills. They are commonly used for educational purposes to introduce students to concepts like graph traversal, pathfinding, and computational thinking. They also have applications in real-world scenarios such as robotic navigation, game design, and optimization problems.

4.SOFTWARE REQUIRMENT SPECIFICATIONS

4.1. Software requirements

For developing the application, the following are the Software Requirements

1. **Python 3**

Fig. 4.3.1 Python 3 Logo

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas the other languages use punctuations. It has fewer syntactical constructions than other languages like it is Interpreted, Interactive, Object-Oriented, Beginners Language. In this project we use python language to develop the web application.

2. **Tkinter**

Fig. 4.3.2 TensorFlow Logo

Tkinter is the most commonly used library for developing GUI (Graphical User Interface) in Python. It is a standard Python interface to the Tk GUI toolkit shipped with Python. As Tk and Tkinter are available on most of the Unix platforms as well as on the Windows system, developing GUI applications with Tkinter becomes the fastest and easiest.

4.2. Hardware requirements

- **CPU:** A powerful CPU is required to run the machine learning algorithms used for image restoration. A CPU with at least four cores is recommended.
- **RAM:** At least 8GB of RAM is recommended.
- **Storage:** Sufficient storage is required to store the image restoration models and user data. At least 100GB of storage is recommended.

Technologies and Languages used to Develop.

- **Python**

5.IMPLEMENTATION

The implementation of a rat maze involves designing the maze structure, defining the rules and constraints, and developing an algorithm or program to navigate the maze. Here are the basic steps involved in implementing a rat maze:

1. Define the maze structure: Determine the size, shape, and complexity of the maze. This can be done using a grid-based representation, where each cell represents a location in the maze. Walls are typically represented as obstacles in certain cells.

2. Initialize the rat's position: Determine the starting position of the rat within the maze. This can be done by assigning specific coordinates or cell indices.

3. Define the maze rules: Determine the rules regarding movement within the maze. For example, specify if the rat can move horizontally, vertically, diagonally, or if certain paths are blocked by walls. Also, define the location of the exit or goal that the rat needs to reach.

4. Implement the algorithm: Choose an algorithm for solving the maze, such as depth-first search (DFS), breadth-first search (BFS), A* algorithm, or Dijkstra's algorithm. Implement the chosen algorithm in your preferred programming language.

5. Maze navigation: Start the algorithm from the rat's initial position and continuously move the rat according to the algorithm's rules. The algorithm will direct the rat to explore paths and find the way to the exit.

6. Path marking: Keep track of the rat's movement and mark the path taken to avoid re-visiting the same cells or getting trapped in infinite loops. This can be done by marking visited cells or maintaining a path history.

7. Exit condition: Define when the rat has successfully reached the exit. This could be when the rat reaches a specific cell, when it reaches a particular condition, or when it finds the shortest path to the exit.

8. Visualization: If desired, implement a graphical user interface (GUI) or visualization feature to display the maze, the rat's movement, and the explored paths.

9. Testing and optimization: Test your implementation with different maze configurations to ensure it functions correctly. If desired, optimize the algorithm to improve efficiency or incorporate additional features such as finding the shortest path.

By following these steps, you can implement a rat maze and develop a program or algorithm that navigates the maze in search of the exit.

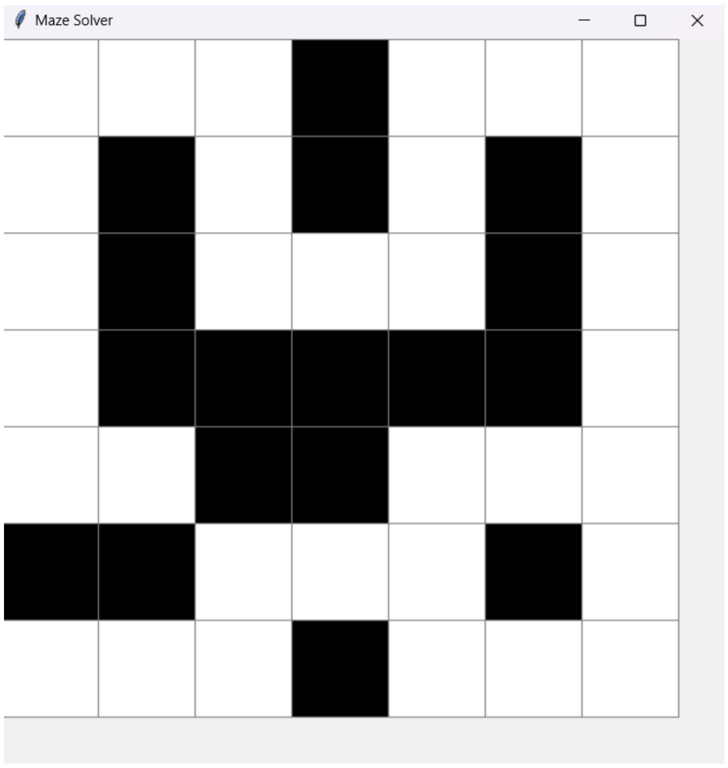
6.ALGORITHM

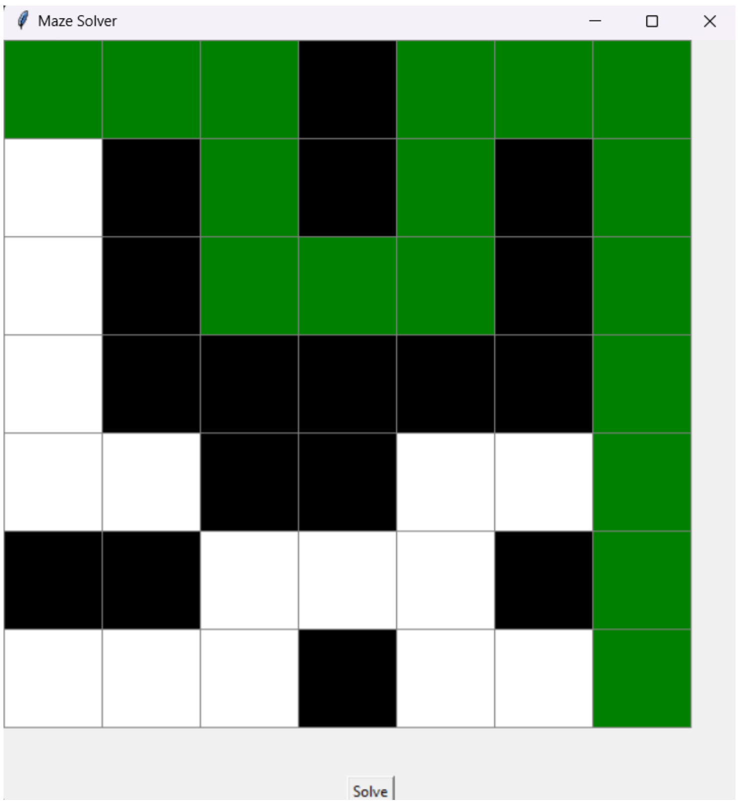
```
function AStar(maze, start, goal):
    rows, cols = dimensions of maze
    openSet = priority queue with initial node (start)
    closedSet = empty set

    while openSet is not empty:
        current = node with the lowest f = g + h value from openSet

        if current is goal:
            reconstruct and return path from start to goal

        add current to closedSet

        for each neighbor of current:
            if neighbor is not valid or in closedSet:
                skip to the next neighbor

            tentativeG = current.g + 1
            if tentativeG < neighbor.g or neighbor is not in openSet:
                update neighbor's g value to tentativeG
                update neighbor's h value to heuristic(neighbor, goal)
                update neighbor's parent to current
                if neighbor is not in openSet:
                    add neighbor to openSet

    return no path found
```

7. SAMPLE CODE

```python
import heapq
import tkinter as tk

class Node:
    def __init__(self, row, col, parent=None):
        self.row = row
        self.col = col
        self.parent = parent
        self.g = 0  # Cost from start to current node
        self.h = 0  # Heuristic (estimated cost from current node to goal)

    def __lt__(self, other):
        # Comparison function for priority queue
        return (self.g + self.h) < (other.g + other.h)

def heuristic(node, goal):
    # Manhattan distance heuristic
    return abs(node.row - goal.row) + abs(node.col - goal.col)

def astar(maze, start, goal):
    rows, cols = len(maze), len(maze[0])
    open_set = [Node(start[0], start[1])]
    closed_set = set()

    while open_set:
        current = heapq.heappop(open_set)

        if (current.row, current.col) == goal:
            path = []
            while current:
                path.append((current.row, current.col))
                current = current.parent
            return path[::-1]

        closed_set.add((current.row, current.col))

        for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            new_row, new_col = current.row + dr, current.col + dc

            if 0 <= new_row < rows and 0 <= new_col < cols and
maze[new_row][new_col] == 0 \
                    and (new_row, new_col) not in closed_set:
                neighbor = Node(new_row, new_col, current)
                neighbor.g = current.g + 1
                neighbor.h = heuristic(neighbor, Node(goal[0], goal[1]))
                if neighbor not in open_set:
                    heapq.heappush(open_set, neighbor)

    return None  # No path found

class MazeSolverApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Maze Solver")

        self.canvas = tk.Canvas(root, width=600, height=600, borderwidth=0,
highlightthickness=0)
        self.canvas.grid(row=0, column=0)

        self.maze = [
            [0, 0, 0, 1, 0, 0, 0],
            [0, 1, 0, 1, 0, 1, 0],
            [0, 1, 0, 0, 0, 1, 0],
            [0, 1, 1, 1, 1, 1, 0],
            [0, 0, 1, 1, 0, 0, 0],
            [1, 1, 0, 0, 0, 1, 0],
            [0, 0, 0, 1, 0, 0, 0]

        ]

        self.start_point = (0, 0)
        self.goal_point = (6, 6)

        self.draw_maze()
        self.solve_button = tk.Button(root, text="Solve", command=self.solve_maze)
        self.solve_button.grid(row=1, column=0)

    def draw_maze(self):
        cell_size = 80
        for row in range(len(self.maze)):
            for col in range(len(self.maze[row])):
                x0, y0 = col * cell_size, row * cell_size
                x1, y1 = x0 + cell_size, y0 + cell_size
                color = "white" if self.maze[row][col] == 0 else "black"
                self.canvas.create_rectangle(x0, y0, x1, y1, fill=color,
outline="gray")

    def solve_maze(self):
        result = astar(self.maze, self.start_point, self.goal_point)
        if result:
            self.highlight_path(result)
        else:
            self.show_message("No path found.")
    def highlight_path(self, path):
        cell_size = 80
        for row, col in path:
            x0, y0 = col * cell_size, row * cell_size
            x1, y1 = x0 + cell_size, y0 + cell_size
            self.canvas.create_rectangle(x0, y0, x1, y1, fill="green",
outline="gray")

    def show_message(self, message):
        popup = tk.Toplevel(self.root)
        popup.title("Message")
        label = tk.Label(popup, text=message)
        label.pack(pady=10)
        close_button = tk.Button(popup, text="Close", command=popup.destroy)
        close_button.pack(pady=10)

if __name__ == "__main__":
    root = tk.Tk()
    app = MazeSolverApp(root)
    root.mainloop()
```

8.OUTPUT SCREENS




Maze Solver
Solve

9. CONCLUSION

In conclusion, the development and implementation of a maze solver represent a significant achievement in the field of robotics, artificial intelligence, and computer science. The creation of an efficient maze-solving algorithm involves the integration of various concepts, such as pathfinding algorithms, sensor integration, and decision-making processes.

Maze solvers play a crucial role in various applications, ranging from robotics and autonomous vehicles to gaming and simulation environments. These algorithms showcase the ability to navigate complex and unpredictable environments, demonstrating adaptability and problem-solving capabilities.

The success of a maze solver depends on the chosen algorithm's effectiveness, taking into account factors like computation speed, resource utilization, and adaptability to diverse maze structures. Researchers and developers continue to explore and enhance existing algorithms, aiming to create more robust and versatile maze-solving solutions.

As technology advances, maze solvers are likely to become increasingly sophisticated, incorporating machine learning techniques and advanced sensors to further improve their performance. This evolution opens up new possibilities for applications in real-world scenarios, contributing to the development of smarter and more autonomous systems.

10. FUTURE SCOPE

The future scope for rat mazes extends beyond their current applications in research and algorithmic problem-solving. Here are some potential areas where rat mazes could have a significant impact:

1. Artificial intelligence training: Rat mazes can serve as a training ground for artificial intelligence (AI) algorithms. By using reinforcement learning techniques, AI models can be trained to navigate and solve complex rat mazes. This can contribute to the development of autonomous agents capable of successfully navigating unknown environments.

2. Robotics and automation: Rat mazes can be used to test and develop navigation algorithms for robots. By mimicking the rat's ability to navigate through complex mazes, robots can be trained to navigate real-world environments more efficiently and effectively.

3. Rehabilitation and therapy: Rat mazes could be adapted for therapeutic purposes, such as rehabilitation after neurological injuries or as a cognitive training tool for individuals with cognitive impairments. The interactive nature of rat mazes could be leveraged to enhance motor skills, memory, attention, and problem-solving abilities.

4. Educational tools: Rat mazes can serve as engaging educational tools for teaching complex concepts in neuroscience, biology, and computer science. By designing interactive virtual or physical rat mazes, students can learn about navigation strategies, algorithm development, and cognitive processes in a hands-on and interactive manner.

5. Gamification: Rat mazes can be gamified to create entertaining and challenging maze-solving puzzles. This could be done through the development of maze-solving games for mobile devices or integration of rat mazes into existing video game environments.

6. Behavioral analysis and pattern recognition: Rat mazes can be used as a tool for analyzing and understanding animal behavior. Advanced computer vision and machine learning techniques can be employed for automated tracking and analysis of rat movement patterns within the maze, providing insights into behavioral patterns and decision-making processes.

7. Virtual reality (VR) simulations: Virtual reality technology can be utilized to create immersive rat maze simulations. This would allow researchers and users to experience and interact with virtual rat mazes, providing a realistic and controlled environment for studying rat behavior and testing algorithms.

These are just a few potential areas where rat mazes could find future applications and contribute to advancements in various fields. As technology continues to evolve, the scope for innovation and utilization of rat mazes is likely to expand even further.

11. BIBILOGRAPHY

- Maze Solving Algorithms for Micro Mouse
- Intelligent Maze Solving Robot Based on Image Processing and Graph Theory Algorithms
- Pathfinding in Strategy Games and Maze Solving Using A^* Search Algorithm