Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)

AI Algorithm Visualizer

Course Title: Artificial Intelligence Lab
Course Code: CSE 316
Section: 221 D8

<u>Students Details</u>

Name	ID
Md. Zahidul Islam	221902091

Submission Date: 27 April, 2025
Course Teacher's Name: Farjana Akter Jui

[For teachers use only: Don't write anything inside this box]

Project Report Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

This project aims to develop a Python-based AI Algorithm Visualizer with a graphical interface to help users understand AI techniques through interactive visualization. It covers fundamental AI concepts such as search algorithms (BFS, DFS, A*), game-playing (Minimax), constraint satisfaction (Backtracking), knowledge representation, and fuzzy logic. The system allows users to select algorithms, input data, and observe step-by-step execution, making AI concepts more accessible and engaging.

1.2 Motivation

The motivation for this project is to provide a practical and interactive way to understand AI concepts. Many students struggle to grasp AI techniques through theory alone. By visualizing how algorithms work in real time, this project will help users build a strong conceptual understanding of AI.

1.3 Problem Definition

Understanding AI concepts through traditional methods can be difficult, as theoretical explanations often lack interactive elements. Many students and beginners struggle to visualize how AI algorithms work in practice. This project addresses this issue by providing an AI Algorithm Visualizer that allows users to see step-by-step execution of various AI techniques. By incorporating graphical representations and animations, the system enhances learning and provides a practical way to interact with AI concepts.

1.3.1 Problem Statement

The key challenge is to develop an AI Algorithm Visualizer that effectively addresses the following issues:

- **Lack of Interactive Learning** – Traditional learning methods rely on textbooks and static explanations, making it difficult for students to visualize how AI algorithms work. [1]

- **Complexity of AI Algorithms** – Many AI techniques, such as search algorithms, game-playing strategies, and constraint satisfaction methods, involve intricate step-by-step computations that are hard to follow without visualization. [2]

- **Absence of Real-Time Execution Feedback** – Without an interactive tool, learners cannot observe the intermediate steps of AI algorithms, leading to gaps in understanding.

- **Difficulty in Debugging AI Logic** – Developers and researchers often struggle to analyze AI behavior due to the absence of tools that provide graphical insights into algorithm execution.

- **Performance vs. Usability Trade-Off** – A challenge exists in balancing smooth system performance with detailed visual representation, ensuring a user-friendly experience.

1.3.2 Complex Engineering Problem

The design and implementation of the AI Algorithm Visualizer involve addressing several complex engineering challenges. The table below summarizes the critical aspects of the project:

Table 1.1: Summary of the project's complex engineering aspects

Aspect	**Explanation**
P1: Depth of knowledge required	Understanding AI techniques such as search algorithms, game playing, constraint satisfaction, and knowledge representation.
P2: Range of conflicting requirements	Balancing detailed visualizations with smooth system performance while ensuring a user-friendly interface.
P3: Depth of analysis required	Implementing efficient algorithms to provide accurate step-by-step execution without lag or performance issues.
P4: Familiarity with issues	Handling real-time user interactions, debugging algorithmic inconsistencies, and ensuring correctness in AI visualizations.
P5: Extent of applicable codes	Using Python-based GUI development with Tkinter or other libraries and efficient AI implementations for accurate and interactive learning.
P6: Extent of stakeholder involvement	Designing the system to meet the needs of students, researchers, and educators while ensuring an intuitive interface.
P7: Interdependence	Ensuring seamless integration of multiple AI techniques within the same system without conflicts or performance degradation.

1.4 Objectives

The primary objectives of this project are:

- To develop a GUI-based system for interactive AI algorithm visualization.
- To implement various AI algorithms and display their execution step by step.
- To help users understand AI concepts through graphical representations and animations.
- To provide an easy-to-use interface for selecting and running different AI techniques.

1.5 Applications

The AI Algorithm Visualizer can be useful in many areas, such as:

- **Education:** Helps students and learners understand AI concepts interactively.
- **Research:** Provides a tool for testing AI algorithms and understanding their behavior.
- **Software Development:** Assists developers in debugging and optimizing AI-based applications.

1.6 Implementation Plan

This project utilizes Python along with GUI frameworks such as Tkinter (or optionally PyQt/Pygame for visual enhancements). The main features include:

- **Search Algorithm Visualization:** BFS, DFS, and A* Search. [3]
- **Game Playing Algorithm:** Minimax Algorithm for Tic-Tac-Toe.
- **Constraint Satisfaction Problems:** Backtracking Algorithm for Sudoku Solver.
- **Knowledge Representation:** Rule-based decision system.
- **Fuzzy Logic Demonstration:** Simple fuzzy control system.

Chapter 2

Implementation of the Project

2.1 Introduction

The **AI Algorithm Visualizer** is a Python-based educational tool designed to demonstrate core AI algorithms through interactive visualizations [4]. Using libraries like `Tkinter` for GUI and custom Python logic for algorithms, the project simplifies complex AI concepts such as BFS, DFS, A*, Minimax, and Backtracking. The aim is to enhance learning by providing users with a hands-on way to observe how these algorithms function step by step. This chapter outlines the system's structure, key components, and implementation process.

2.2 Project Details

The AI Algorithm Visualizer project is designed to simulate and visualize various AI algorithms in real time. Built entirely in Python, the system provides a user-friendly interface and graphical feedback that enhances understanding and engagement. The following subsections provide a detailed overview of the components and features of the project.

2.2.1 Implemented AI Algorithms

- **Search Algorithms**
  - **Breadth-First Search (BFS):** Visualizes level-wise expansion in a graph/-grid.
  - **Depth-First Search (DFS):** Shows deep traversal with backtracking.
  - **A*:** Demonstrates heuristic-based optimal pathfinding.
- **Game Playing Algorithm**
  - **Minimax (Tic-Tac-Toe):** Simulates turn-based decision-making with AI evaluation.

- **Constraint Satisfaction**
  - **Backtracking (Sudoku Solver):** Visualizes recursive backtracking on a puzzle.
- **Knowledge Representation**
  - **Rule-Based System:** Evaluates if-then rules on user input.
- **Fuzzy Logic**
  - **Fuzzy Controller Example:** Demonstrates fuzzy logic in simple control systems. [5]

2.2.2 Methodology

Here's a project methodology based **Figure 2.1**, outlining the development of several AI applications:

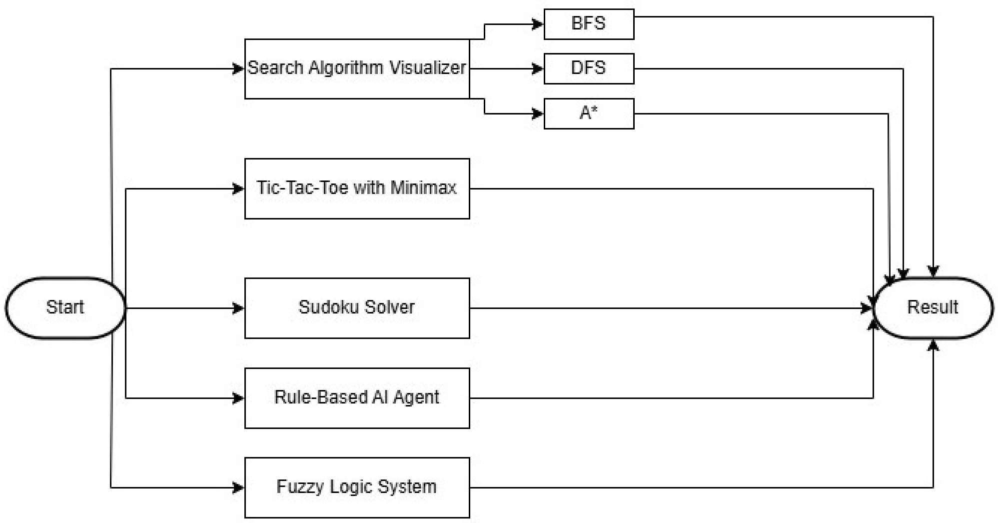

Figure 2.1: Methodology of AI Algorithm Visualizer

2.2.3 GUI Design

The interface was built using `Tkinter` and includes the following elements:

- Dropdown menus for selecting algorithms
- Input fields or random data generation options
- Buttons for visualization control (e.g., Start, Next Step)
- Canvas or plot area for graphical output
- Text areas for step-by-step status messages

2.2.4 Modularity

Each AI algorithm is implemented as a separate Python module or class. This modular design ensures:

- Easy maintenance and debugging
- Clear separation of logic and GUI
- Smooth integration of future algorithms

2.3 Implementation

2.3.1 Workflow

1. User selects an algorithm from the menu.
2. Input is provided manually or generated randomly.
3. Clicking "Visualize" starts the algorithm.
4. Each step is visually updated with a delay.
5. Final results are displayed at the end of execution.

2.3.2 Programming Language and Tools

- **Language:** Python
- **GUI Framework:** Tkinter
- **Visualization Libraries:** `matplotlib`, `networkx`, and custom canvas drawings
- **Development Environment:** Visual Studio Code / PyCharm
- **Other Libraries:** `time`, `random`, optional `pygame` for advanced visuals

2.3.3 Source Code

Source code for main.py

Listing 2.1: Source code for main.py

```
import tkinter as tk
from tkinter import messagebox
import subprocess
import os

# Function to open other Python files
def open_module(filename):
```

```python
    try:
        subprocess.Popen(["python", filename])
    except Exception as e:
        messagebox.showerror("Error", f"Failed to open
        {filename}\n{e}")

# Create main window
root = tk.Tk()
root.title("AI Project - Main Menu")
root.geometry("400x400")
root.configure(bg="#f0f0f0")

tk.Label(root, text="AI Project Modules", font=("Helvetica",
16, "bold"), bg="#f0f0f0").pack(pady=20)

modules = [
    ("Search Algorithm Visualizer", "search_visualizer.py"),
    ("Tic-Tac-Toe with Minimax", "tictactoe.py"),
    ("Sudoku Solver", "sudoku_solver.py"),
    ("Rule-based System", "rule_based_system.py"),
    ("Fuzzy Logic System", "fuzzy_logic.py")
]

for text, file in modules:
    tk.Button(
        root, text=text, width=30, height=2,
        command=lambda f=file: open_module(f),
        bg="#007acc", fg="white", font=("Helvetica", 10, "bold")
    ).pack(pady=10)

root.mainloop()
```

*****Source code for tempCodeRunnerFile.py*****

Listing 2.2: Source code for tempCodeRunnerFile.py

```python
import tkinter as tk
from tkinter import messagebox
import subprocess
import os

# Function to open other Python files
def open_module(filename):
    try:
        subprocess.Popen(["python", filename])
    except Exception as e:
        messagebox.showerror("Error", f"Failed to open {filename}\
            n{e}")

# Create main window
root = tk.Tk()
root.title("AI Project - Main Menu")
root.geometry("400x400")
root.configure(bg="#f0f0f0")

tk.Label(root, text="AI Project Modules", font=("Helvetica",
```

```
16, "bold"), bg="#f0f0f0").pack(pady=20)

modules = [
    ("Search Algorithm Visualizer", "search_visualizer.py"),
    ("Tic-Tac-Toe with Minimax", "tictactoe.py"),
    ("Sudoku Solver", "sudoku_solver.py"),
    ("Rule-based System", "rule_based_system.py"),
    ("Fuzzy Logic System", "fuzzy_logic.py")
]

for text, file in modules:
    tk.Button(
        root, text=text, width=30, height=2,
        command=lambda f=file: open_module(f),
        bg="#007acc", fg="white", font=("Helvetica", 10, "bold")
    ).pack(pady=10)

root.mainloop()
```

*****Source code for search_visualizer.py*****

Listing 2.3: Source code for search_visualizer.py

```
import tkinter as tk
from tkinter import messagebox
import time
from queue import Queue, LifoQueue, PriorityQueue
from itertools import count


GRID_SIZE = 20
CELL_SIZE = 30

class Cell:
    def __init__(self, x, y, button):
        self.x = x
        self.y = y
        self.button = button
        self.state = "empty"  # empty, wall, start,
        end, visited, path

class SearchVisualizer:
    def __init__(self, master):
        self.master = master
        self.master.title("Search Algorithm Visualizer")
        self.grid = []
        self.start = None
        self.end = None
        self.build_grid()

        algo_frame = tk.Frame(master)
        algo_frame.pack(pady=10)

        tk.Button(algo_frame, text="BFS", command=self.run_bfs)
        .pack(side=tk.LEFT, padx=10)
        tk.Button(algo_frame, text="DFS", command=self.run_dfs)
```

```
        .pack(side=tk.LEFT, padx=10)
        tk.Button(algo_frame, text="A*", command=self.run_astar)
        .pack(side=tk.LEFT, padx=10)

        tk.Button(master, text="Reset", command=self.reset_grid)
        .pack(pady=10)

    def build_grid(self):
        frame = tk.Frame(self.master)
        frame.pack()
        for i in range(GRID_SIZE):
            row = []
            for j in range(GRID_SIZE):
                btn = tk.Button(frame, width=2, height=1, bg="white",
                                command=lambda x=i, y=j: self
                                .cell_clicked(x, y))
                btn.grid(row=i, column=j)
                row.append(Cell(i, j, btn))
            self.grid.append(row)

    def cell_clicked(self, x, y):
        cell = self.grid[x][y]
        if not self.start:
            cell.state = "start"
            cell.button.config(bg="green")
            self.start = cell
        elif not self.end and cell != self.start:
            cell.state = "end"
            cell.button.config(bg="red")
            self.end = cell
        elif cell.state == "empty":
            cell.state = "wall"
            cell.button.config(bg="black")

    def reset_grid(self):
        self.grid = []
        self.start = None
        self.end = None
        for widget in self.master.winfo_children():
            if isinstance(widget, tk.Frame):
                widget.destroy()
        self.build_grid()

    def run_bfs(self):
        if not self.start or not self.end:
            messagebox.showwarning("Warning",
            "Please select a start and end point.")
            return

        for row in self.grid:
            for cell in row:
                if cell.state == "visited" or cell.state == "path":
                    cell.state = "empty"
                    cell.button.config(bg="white")

        q = Queue()
```

```
        q.put((self.start, []))
        visited = set()
        visited.add((self.start.x, self.start.y))

        while not q.empty():
            current, path = q.get()

            if current == self.end:
                for cell in path:
                    if cell not in [self.start, self.end]:
                        cell.state = "path"
                        cell.button.config(bg="yellow")
                return

            for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
                nx, ny = current.x + dx, current.y + dy
                if 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE:
                    neighbor = self.grid[nx][ny]
                    if neighbor.state in ["empty", "end"] and (nx,
                    ny) not in visited:
                        visited.add((nx, ny))
                        if neighbor != self.end:
                            neighbor.state = "visited"
                            neighbor.button.config(bg="blue")
                        q.put((neighbor, path + [current]))
                        self.master.update()
                        time.sleep(0.01)

        messagebox.showinfo("Result", "No path found!")


    def run_dfs(self):
        if not self.start or not self.end:
            messagebox.showwarning("Warning",
            "Please select a start and end point.")
            return

        for row in self.grid:
            for cell in row:
                if cell.state == "visited" or cell.state == "path":
                    cell.state = "empty"
                    cell.button.config(bg="white")

        stack = LifoQueue()
        stack.put((self.start, []))
        visited = set()
        visited.add((self.start.x, self.start.y))

        while not stack.empty():
            current, path = stack.get()

            if current == self.end:
                for cell in path:
                    if cell not in [self.start, self.end]:
                        cell.state = "path"
                        cell.button.config(bg="yellow")
                return
```

```
            for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
                nx, ny = current.x + dx, current.y + dy
                if 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE:
                    neighbor = self.grid[nx][ny]
                    if neighbor.state in ["empty", "end"] and
                    (nx, ny) not in visited:
                        visited.add((nx, ny))
                        if neighbor != self.end:
                            neighbor.state = "visited"
                            neighbor.button.config(bg="blue")
                        stack.put((neighbor, path + [current]))
                        self.master.update()
                        time.sleep(0.01)

        messagebox.showinfo("Result", "No path found!")


    def run_astar(self):
        if not self.start or not self.end:
            messagebox.showwarning("Warning",
            "Please select a start and end point.")
            return

        for row in self.grid:
            for cell in row:
                if cell.state == "visited" or cell.state == "path":
                    cell.state = "empty"
                    cell.button.config(bg="white")

        def heuristic(a, b):
            return abs(a.x - b.x) + abs(a.y - b.y)

        counter = count()
        open_set = PriorityQueue()
        open_set.put((0, next(counter), self.start, []))
        g_score = { (self.start.x, self.start.y): 0 }
        visited = set()

        while not open_set.empty():
            _, _, current, path = open_set.get()

            if current == self.end:
                for cell in path:
                    if cell not in [self.start, self.end]:
                        cell.state = "path"
                        cell.button.config(bg="yellow")
                return

            visited.add((current.x, current.y))

            for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
                nx, ny = current.x + dx, current.y + dy
                if 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE:
                    neighbor = self.grid[nx][ny]
                    if neighbor.state not in ["wall"] and (nx, NY)
                    not in visited:
```

```python
                        temp_g = g_score[(current.x, current.y)] + 1
                        if (nx, ny) not in g_score or temp_g <
                        g_score[(nx, ny)]:
                            g_score[(nx, ny)] = temp_g
                            f_score = temp_g + heuristic(neighbor,
                            self.end)
                            open_set.put((f_score, next(counter),
                            neighbor, path + [current]))
                            if neighbor.state != "end":
                                neighbor.state = "visited"
                                neighbor.button.config(bg="blue")
                            self.master.update()
                            time.sleep(0.01)

        messagebox.showinfo("Result", "No path found!")



# Run the app
if __name__ == "__main__":
    root = tk.Tk()
    app = SearchVisualizer(root)
    root.mainloop()
```

*****Source code for tictactoe.py*****

Listing 2.4: Source code for tictactoe.py

```python
import tkinter as tk
from tkinter import messagebox

class TicTacToe:
    def __init__(self, master):
        self.master = master
        self.master.title("Tic-Tac-Toe")

        self.board = [" " for _ in range(9)]  # 9 cells on the board
        self.current_player = "X"  # X starts
        self.game_over = False

        self.buttons = []
        for i in range(9):
            button = tk.Button(master, text=" ", width=10, height=3,
            font=('normal', 20),
                                   command=lambda i=i: self.make_move(i))
            button.grid(row=i // 3, column=i % 3)
            self.buttons.append(button)

        self.reset_button = tk.Button(master, text="Reset",
        command=self.reset_game)
        self.reset_button.grid(row=3, column=0, columnspan=3, pady=10)
```

```

    def make_move(self, index):
        if self.board[index] == " " and not self.game_over:
            self.board[index] = self.current_player
            self.buttons[index].config(text=self.current_player)

            if self.check_winner(self.current_player):
                self.game_over = True
                messagebox.showinfo("Game Over", f"Player
                {self.current_player} wins!")
                return

            if " " not in self.board:
                self.game_over = True
                messagebox.showinfo("Game Over", "It's a draw!")
                return

            # Switch player
            self.current_player = "O" if self.current_player ==
            "X" else "X"
            if self.current_player == "O":
                self.computer_move()

    def check_winner(self, player):
        win_conditions = [(0, 1, 2), (3, 4, 5), (6, 7, 8),  # Rows
                          (0, 3, 6), (1, 4, 7), (2, 5, 8),  # Columns
                          (0, 4, 8), (2, 4, 6)]  # Diagonals
        for condition in win_conditions:
            if all(self.board[i] == player for i in condition):
                return True
        return False

    def computer_move(self):
        best_move = self.minimax(self.board, "O")
        self.make_move(best_move)

    def minimax(self, board, player, alpha=-float("inf"), beta=float("inf")):
        opponent = "O" if player == "X" else "X"

        # Base case: Check if the game is over
        if self.check_winner("O"):
            return 1
        elif self.check_winner("X"):
            return -1
        elif " " not in board:
            return 0

        # Possible moves
        moves = [i for i, x in enumerate(board) if x == " "]

        best_move = None

        if player == "O":  # Maximizing player (computer)
            best_score = -float("inf")
            for move in moves:
                board[move] = player
```

```
                score = self.minimax(board, opponent, alpha, beta)
                board[move] = " "  # Undo move

                if score > best_score:
                    best_score = score
                    best_move = move

                alpha = max(alpha, best_score)
                if beta <= alpha:
                    break  # Beta cut-off

        else:  # Minimizing player (human)
            best_score = float("inf")
            for move in moves:
                board[move] = player
                score = self.minimax(board, opponent, alpha, beta)
                board[move] = " "  # Undo move

                if score < best_score:
                    best_score = score
                    best_move = move

                beta = min(beta, best_score)
                if beta <= alpha:
                    break  # Alpha cut-off

        return best_move


    def reset_game(self):
        self.board = [" " for _ in range(9)]
        self.current_player = "X"
        self.game_over = False
        for button in self.buttons:
            button.config(text=" ")

# Run the app
if __name__ == "__main__":
    root = tk.Tk()
    app = TicTacToe(root)
    root.mainloop()
```

****Source Code for sudoku_solver.py****

Listing 2.5: Source Code for sudoku_solver.py

```
import tkinter as tk
from tkinter import messagebox

class SudokuSolver:
    def __init__(self, master):
        self.master = master
        self.master.title("Sudoku Solver")

        self.board = [[0 for _ in range(9)] for _ in range(9)]  # Initialize 9x9 grid
```

```python
        self.cells = [[None for _ in range(9)] for _ in range(9)]  # Grid for displaying UI
        self.create_grid()

        self.solve_button = tk.Button(master, text="Solve Sudoku", command=self.solve_sudoku)
        self.solve_button.grid(row=9, column=0, columnspan=9)

    def create_grid(self):
        """Create a 9x9 grid of entry fields."""
        for r in range(9):
            for c in range(9):
                entry = tk.Entry(self.master, width=3, font=('Arial', 18), borderwidth=2, relief="solid",
                                 justify="center")
                entry.grid(row=r, column=c, padx=5, pady=5)
                self.cells[r][c] = entry
                self.cells[r][c].bind("<FocusOut>", self.update_board)

    def update_board(self, event=None):
        """Update the board array based on the entries in the grid."""
        for r in range(9):
            for c in range(9):
                value = self.cells[r][c].get()
                if value.isdigit() and 1 <= int(value) <= 9:
                    self.board[r][c] = int(value)
                else:
                    self.board[r][c] = 0

    def solve_sudoku(self):
        """Solve the Sudoku puzzle using the backtracking algorithm."""
        if self.solve():
            self.update_grid()
            messagebox.showinfo("Success", "Sudoku Solved!")
        else:
            messagebox.showerror("Error", "No solution exists")

    def solve(self):
        """Backtracking algorithm to solve Sudoku."""
        for r in range(9):
            for c in range(9):
                if self.board[r][c] == 0:  # Find an empty cell
                    for num in range(1, 10):
                        if self.is_valid(r, c, num):  # Check if num is valid
                            self.board[r][c] = num
                            if self.solve():  # Recursively try to solve with this number
                                return True
                            self.board[r][c] = 0  # Backtrack if it didn't work
                    return False
        return True

    def is_valid(self, r, c, num):
```

```python
        """Check if placing num at board[r][c] is valid."""
        # Check row
        for i in range(9):
            if self.board[r][i] == num:
                return False
        # Check column
        for i in range(9):
            if self.board[i][c] == num:
                return False
        # Check 3x3 sub-grid
        start_row, start_col = 3 * (r // 3), 3 * (c // 3)
        for i in range(start_row, start_row + 3):
            for j in range(start_col, start_col + 3):
                if self.board[i][j] == num:
                    return False
        return True

    def update_grid(self):
        """Update the grid UI to reflect the solved board."""
        for r in range(9):
            for c in range(9):
                self.cells[r][c].delete(0, tk.END)
                if self.board[r][c] != 0:
                    self.cells[r][c].insert(tk.END, str(self.board[r][c]))


# Run the app
if __name__ == "__main__":
    root = tk.Tk()
    app = SudokuSolver(root)
    root.mainloop()
```

*****Source code for rule_based_system.py*****

Listing 2.6: Source code for rule_based_system.py

```python
import tkinter as tk
import subprocess
import sys
import os

class RuleBasedSystem:
    def __init__(self):
        self.rules = []
        self.facts = {}

    def add_rule(self, condition, action):
        self.rules.append((condition, action))

    def add_fact(self, fact, value):
        self.facts[fact] = value

    def evaluate(self):
        for condition, action in self.rules:
            if condition(self.facts):
```

```
                return action()
        return "No specific action recommended."


# Rules
def rule_rainy(facts):
    return facts.get("rain", False)

def rule_cold_and_cloudy(facts):
    return facts.get("temperature", 0) < 20 and facts.get("cloudy", False)

def rule_sunny(facts):
    return facts.get("temperature", 0) > 25 and not facts.get("rain", False)

# Actions
def action_carry_umbrella():
    return "Carry an umbrella    "

def action_no_umbrella():
    return "No umbrella needed       "


# GUI App
class UmbrellaDecisionApp:
    def __init__(self, master):
        self.master = master
        self.master.title("Rule-Based System: Umbrella Decision")
        self.master.geometry("500x400")
        self.master.configure(bg="#f0f0f0")

        self.setup_rule_system()
        self.create_widgets()

    def setup_rule_system(self):
        self.system = RuleBasedSystem()
        self.system.add_rule(rule_rainy, action_carry_umbrella)
        self.system.add_rule(rule_cold_and_cloudy, action_carry_umbrella)
        self.system.add_rule(rule_sunny, action_no_umbrella)

    def create_widgets(self):
        tk.Label(self.master, text="Enter Weather Conditions", font=("Helvetica", 14, "bold"), bg="#f0f0f0").pack(pady=10)

        tk.Label(self.master, text="Is it raining? (yes/no):", bg="#f0f0f0").pack()
        self.rain_entry = tk.Entry(self.master)
        self.rain_entry.pack()

        tk.Label(self.master, text="Temperature ( C ):", bg="#f0f0f0").pack()
        self.temp_entry = tk.Entry(self.master)
        self.temp_entry.pack()

```

```python
        tk.Label(self.master, text="Is it cloudy? (yes/no):", bg="#f0f0f0").pack()
        self.cloud_entry = tk.Entry(self.master)
        self.cloud_entry.pack()

        self.result_label = tk.Label(self.master, text="", font=("Helvetica", 12), bg="#f0f0f0", fg="blue")
        self.result_label.pack(pady=15)

        tk.Button(self.master, text="Get Decision", bg="#007acc", fg="white",
                  font=("Helvetica", 10, "bold"), command=self.evaluate_input).pack(pady=10)

        tk.Button(self.master, text="Back to Main Menu", bg="gray", fg="white",
                  font=("Helvetica", 10), command=self.back_to_main).pack(pady=5)

    def evaluate_input(self):
        rain = self.rain_entry.get().strip().lower() == "yes"
        cloudy = self.cloud_entry.get().strip().lower() == "yes"
        try:
            temperature = float(self.temp_entry.get().strip())
        except ValueError:
            self.result_label.config(text="Invalid temperature! Enter a number.")
            return

        self.system.facts = {
            "rain": rain,
            "cloudy": cloudy,
            "temperature": temperature
        }

        result = self.system.evaluate()
        self.result_label.config(text=result)

    def back_to_main(self):
        self.master.destroy()
        subprocess.Popen([sys.executable, "main.py"])


# Run from main.py
def run_rule_based_gui():
    root = tk.Tk()
    app = UmbrellaDecisionApp(root)
    root.mainloop()

# Direct execution
if __name__ == "__main__":
    run_rule_based_gui()
```

*****Source code for fuzzy_logic.py*****

Listing 2.7: Source code for fuzzy_logic.py

```python
import tkinter as tk
from tkinter import messagebox
import skfuzzy as fuzz
import numpy as np

# Main GUI Window
root = tk.Tk()
root.title("Fuzzy Logic System")
root.geometry("500x400")
root.configure(bg="#f9f9f9")

# Menu logic handler
def show_menu(selection):
    # Remove all widgets except the menu
    for widget in root.winfo_children():
        if not isinstance(widget, tk.Menu):
            widget.destroy()

    if selection == "Food Suggestion":
        show_food_suggestion()

# Food Suggestion System
def show_food_suggestion():
    title = tk.Label(root, text="Fuzzy Food Suggestion", font=("Helvetica", 14, "bold"), bg="#f9f9f9")
    title.pack(pady=10)

    tk.Label(root, text="Hunger Level (0-10)", bg="#f9f9f9").pack()
    hunger_scale = tk.Scale(root, from_=0, to=10, orient=tk.HORIZONTAL)
    hunger_scale.pack()

    tk.Label(root, text="Spicy Preference (0-10)", bg="#f9f9f9").pack()
    spicy_scale = tk.Scale(root, from_=0, to=10, orient=tk.HORIZONTAL)
    spicy_scale.pack()

    def suggest_food():
        hunger = hunger_scale.get()
        spice = spicy_scale.get()

        # Defining fuzzy sets for hunger and spice preferences
        x = np.arange(0, 11, 1)  # Fuzzy range from 0 to 10
        hunger_high = fuzz.trimf(x, [5, 10, 10])  # High hunger membership function
        spice_high = fuzz.trimf(x, [5, 10, 10])  # High spice membership function

        # Evaluating fuzzy values for hunger and spice
        h_val = fuzz.interp_membership(x, hunger_high, hunger)  # Membership for hunger
        s_val = fuzz.interp_membership(x, spice_high, spice)  # Membership for spice

        # Decision-making for food suggestion
```

```
        if h_val > 0.7 and s_val > 0.7:
            msg = " Suggestion: Biriyani or Kacchi"
        elif h_val > 0.7:
            msg = " Suggestion: Rice with Curry"
        elif s_val > 0.7:
            msg = " Suggestion: Fuchka or Chatpati"
        else:
            msg = " Suggestion: Light Snacks"

        messagebox.showinfo("Food Suggestion", msg)

    tk.Button(root, text="Get Suggestion", command=suggest_food,
        bg="#007acc", fg="white").pack(pady=10)

# Create a menu for navigation
menu_bar = tk.Menu(root)
option_menu = tk.Menu(menu_bar, tearoff=0)
option_menu.add_command(label="Food Suggestion", command=lambda:
    show_menu("Food Suggestion"))
menu_bar.add_cascade(label="Select System", menu=option_menu)
root.config(menu=menu_bar)

# Initial view
show_menu("Food Suggestion")

root.mainloop()
```

Chapter 3

Performance Evaluation

3.1 Simulation Environment

The project was developed and tested in a Python 3 environment using standard libraries. Tkinter was used for GUI-based interaction. The simulation was performed on a local machine running Windows 10 with Python 3.11 installed. No additional packages were required beyond Python's standard library. Each module (e.g., fuzzy logic, rule-based system, search visualizer, game AI) was executed via a unified Tkinter interface, allowing users to interact with and observe the AI systems in real time. The environment ensured smooth visualization and accurate output generation for each AI concept. [6]

3.2 Results Analysis

Each AI module was individually tested for correctness and efficiency.

- The **fuzzy logic system (Figure 3.2)** produced accurate outputs based on varying input ranges.
- The **rule-based system (Figure 3.8)** successfully inferred conclusions from defined rules. [7]
- The **Sudoku solver (Figure 3.6)** consistently solved puzzles without error using backtracking.
- The **search visualizer** clearly demonstrated BFS **(Figure 3.3)** , DFS **(Figure 3.4)** and A* **(Figure 3.5)** pathfinding behaviors.
- The **Tic Tac Toe AI (Figure 3.7)** performed flawlessly, never losing a game due to the Minimax algorithm.

Overall, the system performed reliably, with accurate outcomes across all test scenarios.

3.3 Results Overall Discussion

The results were achieved by implementing each AI algorithm (BFS, DFS, A*, Minimax, Backtracking, Rule-Based System, and Fuzzy Logic) in modular Python scripts and visualizing their behaviors using Tkinter-based GUI. User inputs and real-time feedback helped validate correct algorithm execution. Problems detected included occasional minor GUI delays during large grid searches and slight complexity when handling simultaneous events. Overall, the project met its objectives with smooth, accurate, and interactive performance across all modules.

3.3.1 Complex Engineering Problem Discussion

The AI Algorithm Visualizer addressed multiple complex engineering aspects. Depth of knowledge was applied through the implementation of AI algorithms like BFS, DFS, A*, Minimax, and Backtracking [8] . Conflicting requirements, such as balancing smooth GUI performance with detailed visual feedback, were solved by modular code design and efficient use of Tkinter. Real-time interaction and visualization challenges were overcome by carefully managing event handling and update delays. Integration of diverse AI techniques into a unified, user-friendly system ensured that the interdependence and stakeholder needs were successfully met.

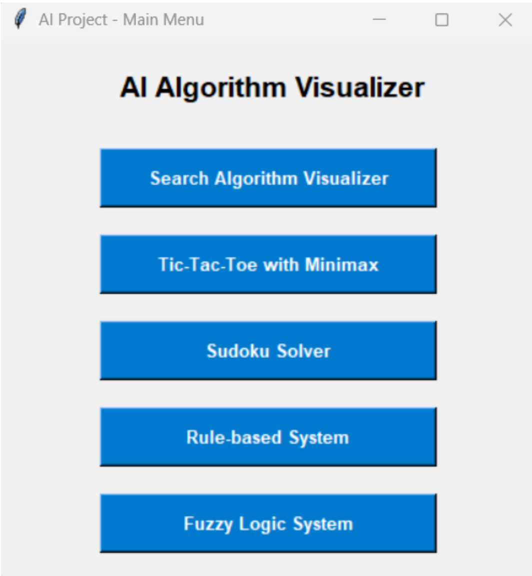

Figure 3.1: Home page of AI Algorithm Visualizer

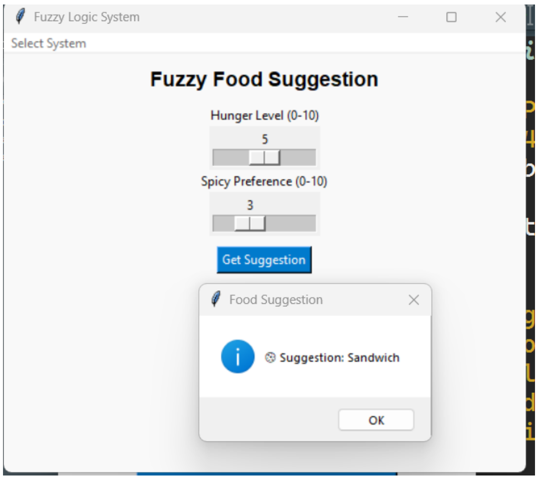

Figure 3.2: Visual Representation of Fuzzy Logic

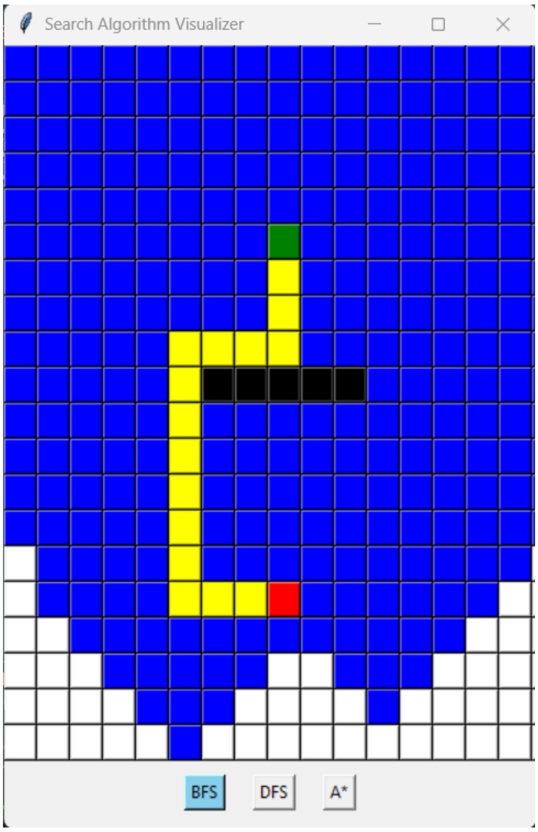

Figure 3.3: Visual Representation of BFS

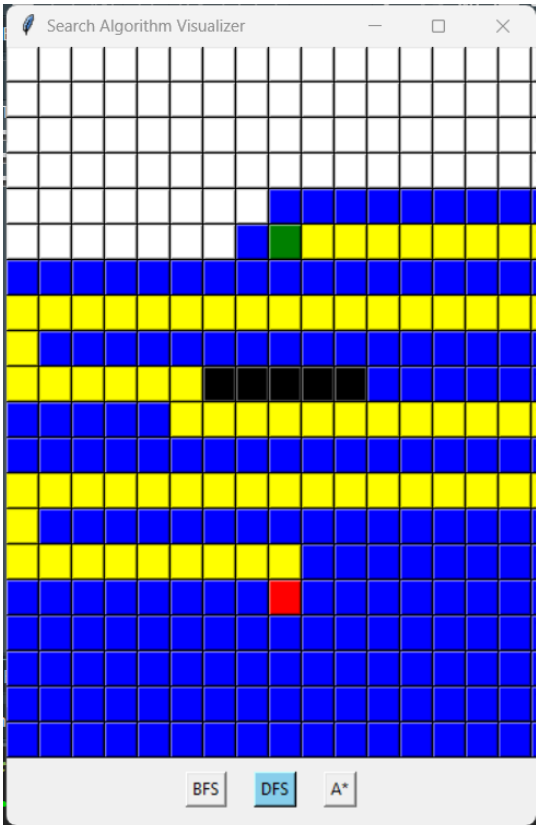

Figure 3.4: Visual Representation of DFS

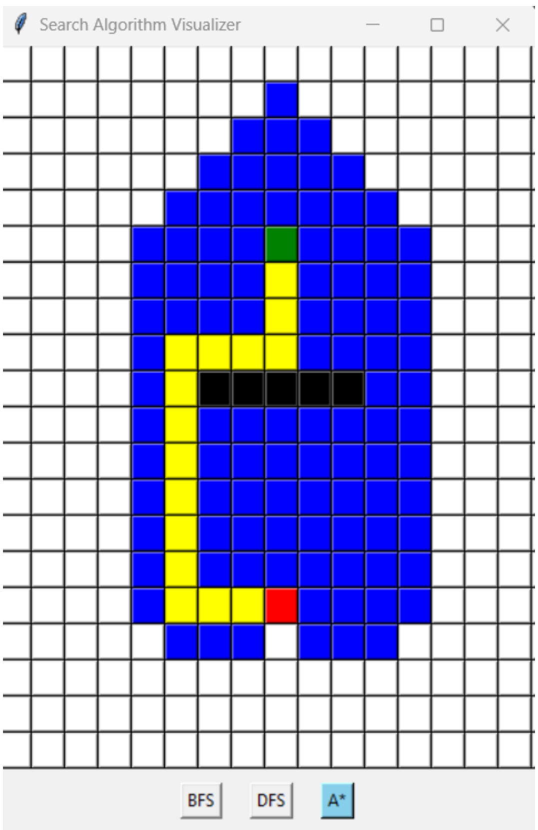

Figure 3.5: Visual Representation of A*

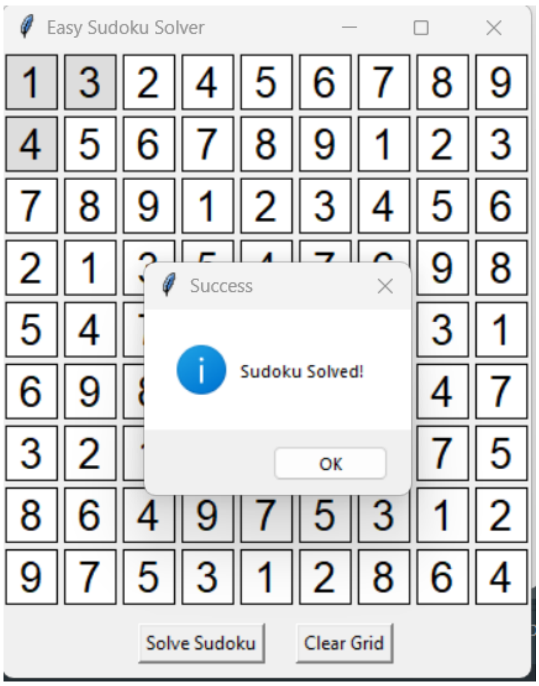

Figure 3.6: Visual Representation of Sudoku Solver

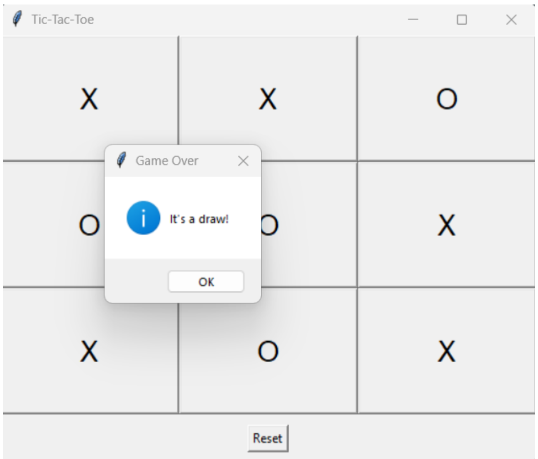

Figure 3.7: Visual Representation of Tic Tac Toe

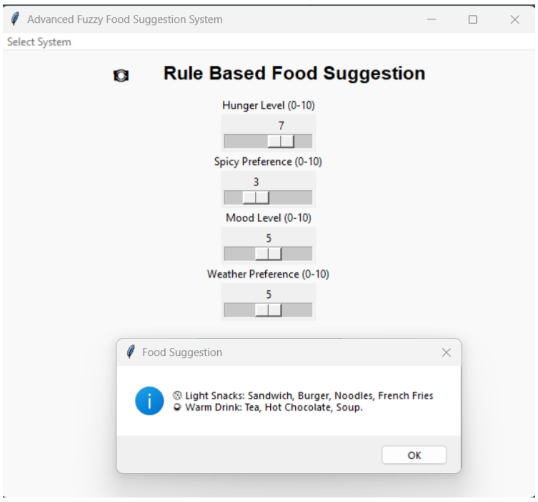

Figure 3.8: Visual Representation of Rule Based Food Suggestion

Chapter 4

Conclusion

4.1 Discussion

This project successfully developed an AI Algorithm Visualizer **(Figure 3.1)** that implemented various AI techniques with real-time visualization. The work enhanced understanding of AI concepts through modular, interactive GUI-based simulations, and the results matched expected algorithm behaviors across all modules.

4.2 Limitations

The main limitations include slight GUI lag during large search visualizations, basic design compared to advanced visualizers, and limited flexibility in handling highly complex inputs. Additionally, scalability to larger datasets was not optimized due to focus on educational clarity.

4.3 Scope of Future Work

Future improvements include optimizing performance for larger grids, adding more AI algorithms (e.g., Genetic Algorithm, Reinforcement Learning), enhancing the GUI with PyQt for better user experience, and enabling step-by-step debugging features for deeper analysis.

References

[1] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[2] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited, 2016.

[3] Donald E Knuth and Ronald W Moore. An analysis of alpha-beta pruning. In *Artificial Intelligence*, volume 6, pages 293–326. Elsevier, 1975.

[4] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[5] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[6] Allen Newell, J.C. Shaw, and Herbert A Simon. Report on a general problem-solving program. *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959.

[7] Matthew L Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[8] Rina Dechter. Constraint processing. In *The Morgan Kaufmann Series in Artificial Intelligence*. Morgan Kaufmann, 2003.