MID TERM REPORT

ON

SOLVING PUZZLES USING ALGORITHM

PROJECT-I

Department of Artificial Intelligence & Machine Learning
CHANDIGARH ENGINEERING COLLEGE JHANJERI,MOHALI

In partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Artificial Intelligence & Machine Learning

SUBMITTED BY:
VIBHUTI KANWAR (2330663)
VISHAL SUPPAHIYA (2330665)
VISHESH KUMAR(2330666)
ABRAR SHABIR DAR (2420626)
AJAY KUMAR (2420627)
AMAN THAKUR (2420628)

Under the Guidance of:
Tanya Gupta
Assistant Professor

Table of Content

Chapter 1.Introduction

1.1 Introduction

Algorithms are the backbone of artificial intelligence, helping solve complex puzzles and games with precision. Take Sudoku and Tic-Tac-Toe, for example—two classic games that highlight the power of smart strategies. Sudoku challenges players to fill a grid using logic and number placement, while Tic-Tac-Toe is a simple yet strategic game where every move counts. Despite their differences, both require critical thinking and a solid game plan.

In this project, we explore how algorithms can tackle these puzzles efficiently. We will use the Minimax algorithm to create an unbeatable Tic-Tac-Toe solver and apply Backtracking with Constraint Propagation to crack even the toughest Sudoku grids.

1.2 Objectives

- Analyzing Optimized Algorithms
- Implementation of Minimax for Tic-Tac-Toe
- Utilization of Backtracking for Sudoku
- Performance Evaluation
- Algorithmic Enhancements and Broader Applications

1.3 Tools and Technology used

Tools and Technologies Used

(A) **Programming Language:** Python

(B) **Libraries & Frameworks:**

- NumPy (Efficient array operations)
- Pygame (GUI visualization for Tic-Tac-Toe)
- Tkinter (Interactive Sudoku interface)

(C) **Algorithms:**

- Minimax with Alpha-Beta Pruning (Tic-Tac-Toe AI)
- Backtracking with Constraint Propagation (Sudoku Solver)

(D) **Development Environment:**

- Jupyter Notebook / VS Code / PyCharm

(E) **Performance Evaluation Tools:**

- Python Profiling Tools (cProfile, timeit)
- Memory Management

Chapter 2. System Requirements

2.1 Software Requirements

(A) Operating System: Windows, Linux, or macOS

(B) Programming Language: Python 3.x

(C) Libraries & Frameworks: Tkinter, numpy

2.2 Hardware Requirements

(A) **Processor:** Intel i3 or higher

(B) **RAM:** Minimum 4GB

(C) **Storage:** At least 50GB

Chapter 3. Software Requirement Analysis

3.1 Problem Definition

Artificial Intelligence (AI) has revolutionized problem-solving across multiple domains, including puzzle-solving and strategic games. Traditional approaches to solving puzzles like Tic-Tac-Toe and Sudoku relied on brute-force methods, which were computationally expensive and inefficient for large-scale problems. With the advancement of AI, machine learning algorithms, Monte Carlo Tree Search (MCTS), and deep reinforcement learning techniques have enabled AI to solve complex puzzles efficiently.

This project aims to analyze and implement various AI-driven techniques to enhance problem-solving capabilities in puzzles. The objective is to compare different algorithms, optimize decision-making strategies, and create an AI system capable of solving Tic-Tac-Toe, Sudoku, and similar puzzles with high efficiency and accuracy.

3.2 System modules and Functionalities

1. Tic-Tac-Toe Solver Module

- **Objective:** Develop an AI-driven Tic-Tac-Toe solver that ensures optimal play.
- **Functionalities:**
  - Implements Minimax algorithm with Alpha-Beta pruning.
  - Analyzes possible game states and selects the best move.
  - Provides a real-time decision-making AI opponent.
  - GUI integration for user interaction.

2. Sudoku Solver Module

- **Objective:** Implement an efficient Sudoku-solving algorithm.
- **Functionalities:**
  - Uses Backtracking with Constraint Propagation.
  - Analyzes puzzle constraints to eliminate invalid choices.
  - Heuristic optimizations for faster solution finding.
  - Interactive interface for puzzle input and output display.

3. Input Handling System

- **Objective:** Ensure seamless user interaction with the solvers.
- **Functionalities:**
  - Accepts puzzle states from user input.
  - Provides an interactive UI for data entry.
  - Validates input formats and prevents errors.

4. Algorithm Execution Engine

- **Objective:** Process the puzzle-solving logic efficiently.
- **Functionalities:**
  - Implements core logic for Minimax and Backtracking algorithms.
  - Optimizes computational complexity with pruning and heuristics.
  - Dynamically adapts execution speed based on puzzle difficulty.

5. Performance Evaluation System

- **Objective:** Analyze and compare algorithm efficiency.
- **Functionalities:**
  - Measures execution time, accuracy, and memory usage.
  - Compares optimized algorithms against naive implementations.
  - Provides statistical insights into solver effectiveness.

6. User Interface and Visualization

- **Objective:** Enhance user experience with graphical representations.
- **Functionalities:**
  - Uses Pygame for visualizing Tic-Tac-Toe moves.
  - Uses Tkinter for interactive Sudoku grid display.
  - Provides real-time feedback and solution visualization

7. Result Validation and Reporting

- **Objective:** Ensure correctness and efficiency of solutions.
- **Functionalities:**
  - Verifies Tic-Tac-Toe AI decisions against winning strategies.
  - Checks Sudoku solutions for correctness and completeness.
  - Generates reports on algorithm performance and accuracy.

8. Future Enhancements Module

- **Objective:** Provide extensibility for additional features.
- **Functionalities:**
  - Supports integration with deep learning for advanced game AI.
  - Allows cloud-based implementation for remote accessibility.
  - Plans for expanding AI techniques to more complex games.

Chapter 4. Software Design

4.1 System Architecture

The system is divided into two main modules: the Tic-Tac-Toe Solver and the Sudoku Solver, each with a structured computational pipeline to ensure efficiency and accuracy.

4.2 Design Principles

- Modularity: Each component is developed as an independent module.
- Scalability: The system supports future enhancements and additional features.
- Efficiency: Optimized algorithms ensure minimal computation time.
- User-Friendly Interface: Provides interactive elements for a seamless experience.

4.3 System Components

4.3.1 Input Handling

- Accepts user input for Tic-Tac-Toe and Sudoku.
- Validates input format and structure.
- Provides real-time feedback on input errors.

4.3.2 Algorithm Execution Engine

- Implements Minimax with Alpha-Beta Pruning for Tic-Tac-Toe.
- Implements Backtracking with Constraint Propagation for Sudoku.
- Optimizes execution speed using heuristic techniques.

4.3.3 User Interface and Visualization

- Tic-Tac-Toe: GUI with real-time game board using Tkinter.
- Sudoku: Interactive grid-based input using Tkinter.
- Visual Feedback: Displays solutions dynamically with step-by-step updates.

4.3.4 Performance Analysis Module

- Evaluates computational efficiency.
- Measures execution time, accuracy, and memory usage.
- Compares optimized algorithms against traditional approaches.

4.4 Software Implementation

4.4.1 Programming Language and Frameworks

- Language: Python
- Libraries Used:
  - NumPy (Efficient computations)
  - Tkinter (Sudoku UI , Tic-Tac-Toe UI)

4.5 Data Flow Diagram (DFD)

- Level 0: User inputs puzzle state → System processes input → Outputs solution.
- Level 1:
  - User selects puzzle type.
  - Input validated and processed.
  - Algorithm executes with optimizations.
  - Results displayed through UI.

4.6 Summary

This chapter outlines the architecture, principles, and implementation details of the project. The structured approach ensures a balance between computational efficiency and usability.

Chapter 5. Implementation

5.1 Code implementation

```
import tkinter as tk
from tkinter import messagebox
import numpy as np

def show_main_menu():
    root.deiconify()

def hide_main_menu():
    root.withdraw()

def start_tic_tac_toe():
    hide_main_menu()
    tic_tac_toe_window = tk.Toplevel(root)
    tic_tac_toe_window.title("Tic Tac Toe")
    tic_tac_toe_window.geometry("300x400")
    tic_tac_toe_window.configure(bg="#FCE38A")

    playerX, playerO = "X", "O"
    curr_player = playerX
    board = [[None for _ in range(3)] for _ in range(3)]
    turns, game_over = 0, False

    def set_tile(row, column):
        nonlocal curr_player, turns, game_over
        if game_over or board[row][column]["text"]:
            return
        board[row][column]["text"] = curr_player
        curr_player = playerO if curr_player == playerX else playerX
        label["text"] = curr_player + "'s turn"
        check_winner()

    def check_winner():
        nonlocal turns, game_over
        turns += 1

        for row in range(3):
            if board[row][0]["text"] == board[row][1]["text"] ==
board[row][2]["text"] and board[row][0]["text"]:
```

```
                label.config(text=board[row][0]["text"] + " wins!", fg="red")
                game_over = True
                return

        for col in range(3):
            if board[0][col]["text"] == board[1][col]["text"] ==
board[2][col]["text"] and board[0][col]["text"]:
                label.config(text=board[0][col]["text"] + " wins!", fg="red")
                game_over = True
                return

        if (board[0][0]["text"] == board[1][1]["text"] == board[2][2]["text"]
and board[0][0]["text"]) or \
           (board[0][2]["text"] == board[1][1]["text"] == board[2][0]["text"]
and board[0][2]["text"]):
            label.config(text=board[1][1]["text"] + " wins!", fg="red")
            game_over = True
            return

        if turns == 9:
            label.config(text="It's a tie!", fg="blue")
            game_over = True

    def new_game():
        nonlocal turns, game_over, curr_player
        turns, game_over, curr_player = 0, False, playerX
        label.config(text=curr_player + "'s turn", fg="black")
        for r in range(3):
            for c in range(3):
                board[r][c].config(text="")

    def return_to_main():
        tic_tac_toe_window.destroy()
        show_main_menu()

    frame = tk.Frame(tic_tac_toe_window, bg="#FCE38A")
    label = tk.Label(frame, text=curr_player + "'s turn", font=("Arial", 16,
"bold"), bg="#FCE38A")
    label.grid(row=0, column=0, columnspan=3, pady=10)

    for r in range(3):
        for c in range(3):
```

```
            board[r][c] = tk.Button(frame, text="", font=("Arial", 20),
width=5, height=2, bg="white",
                                    command=lambda row=r, col=c:
set_tile(row, col))
            board[r][c].grid(row=r+1, column=c, padx=5, pady=5)

    tk.Button(frame, text="Restart", font=("Arial", 14), bg="#FF6363",
fg="white", command=new_game).grid(row=4, column=0, columnspan=3, pady=10)
    tk.Button(frame, text="Return to Main Menu", font=("Arial", 14),
bg="gray", fg="white", command=return_to_main).grid(row=5, column=0,
columnspan=3, pady=10)
    frame.pack(pady=10)

def start_sudoku():
    hide_main_menu()
    sudoku_window = tk.Toplevel(root)
    sudoku_window.title("Sudoku Solver")
    sudoku_window.configure(bg="#A8E6CF")

    entries = []

    def solve():
        messagebox.showinfo("Sudoku", "Sudoku Solver coming soon!")

    def return_to_main():
        sudoku_window.destroy()
        show_main_menu()

    for i in range(9):
        row_entries = []
        for j in range(9):
            entry = tk.Entry(sudoku_window, width=3, font=('Arial', 14),
justify='center', bg="white")
            entry.grid(row=i, column=j, padx=2, pady=2)
            row_entries.append(entry)
        entries.append(row_entries)

    tk.Button(sudoku_window, text="Solve", command=solve, font=('Arial', 14),
bg="#FF6363", fg="white").grid(row=9, column=0, columnspan=9, pady=10)
    tk.Button(sudoku_window, text="Return to Main Menu", font=('Arial', 14),
bg="gray", fg="white", command=return_to_main).grid(row=10, column=0,
columnspan=9, pady=10)
```

```python
root = tk.Tk()
root.title("Puzzle Hub")
root.geometry("400x300")
root.configure(bg="#FFD3B6")

frame = tk.Frame(root, bg="#FFD3B6")
frame.pack(pady=20)

tk.Label(frame, text="🧩Choose a Puzzle 🎮🧩", font=("Arial", 18, "bold"),
bg="#FFD3B6", fg="#3D348B").pack(pady=10)

tk.Button(frame, text="Solve Tic-Tac-Toe", font=("Arial", 14), bg="#3D348B",
fg="white", width=20, command=start_tic_tac_toe).pack(pady=5)

tk.Button(frame, text="Solve Sudoku", font=("Arial", 14), bg="#3D348B",
fg="white", width=20, command=start_sudoku).pack(pady=5)

tk.Button(frame, text="Exit", font=("Arial", 14), bg="#FF6363", fg="white",
width=20, command=root.quit).pack(pady=10)

root.mainloop()
```

Chapter 6 :References

- GeeksforGeeks: Minimax Algorithm in Game Theory (https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory/)
- Towards Data Science: Solving Sudoku with Backtracking (https://towardsdatascience.com)
- Robert A. Hearn and Erik D. Demaine (2009): Games, Puzzles, and Computation.
- A K Peters. Rémi Coulom (2006): "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." Proceedings of the 5th International Conference on Computers and Games, pp. 72–83.
- L. Kocsis and Cs. Szepesvári (2006): "Bandit Based Monte-Carlo Planning." Proceedings of the 17th European Conference on Machine Learning, pp. 282–293. S. Gelly et al. (2006): "Modifications of UCT with Patterns in Monte-Carlo Go." Technical Report, INRIA.
- Bruce Abramson (1987): "The Expected-Outcome Model of Two-Player Games." Ph.D. Dissertation, Columbia University.
- Bhavuk Kalra (2022): "Enhancing Tic-Tac-Toe Strategies on Larger Boards Using Neural Networks and Genetic Algorithms." Journal of Artificial Intelligence Research, Vol. 75, pp. 123–145.
- Nazneen Rajani et al. (2011): "Optimising Game Move Selection Using Hamming Distance-Based Pattern Matching." International Journal of Game Theory and Technology, Vol. 10, No. 3, pp. 210–225.
- Iaakov Exman and Avinoam Alfia (2014): "A Knowledge-Based Approach to Solving Sudoku Puzzles." International Journal of Advanced Computer Science, Vol. 5, No. 7, pp. 356–362.
- Abhishake Kundu (2021): "Pattern Recognition Techniques for Efficient Sudoku Solving." Journal of Computational Puzzles and Games, Vol. 12, No. 1, pp. 45–59.
- S. Jain (2015): "Design and Implementation of a Tic-Tac-Toe Playing Robot Using Lego Mindstorms." Proceedings of the International Conference on Robotics and Automation, pp. 1123–1128.