

|  |
| --- |
| **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** | 2117240070054 |
| **NAME** | S DEVESH |
| **PROJECT TITLE** | Implementation of Tic-Tac-Toe Game using Alpha-Beta Pruning |
| **DATE OF SUBMISSION** |  |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

**INTRODUCTION**

Artificial Intelligence (AI) is a branch of computer science that enables machines to simulate human intelligence, such as learning, reasoning, and decision-making. Game playing has long served as a key area for testing AI algorithms since it involves strategy, prediction, and optimal decision-making.  
  
This mini-project focuses on implementing the Tic-Tac-Toe game using the Alpha-Beta Pruning technique, an optimized version of the Minimax algorithm. The goal is to create an AI opponent that can make smart and efficient decisions, ensuring it either wins or forces a draw. A graphical user interface (GUI) built with Tkinter provides an interactive and user-friendly environment for gameplay.

**PROBLEM STATEMENT**

The main objective of this project is to design and implement a Tic-Tac-Toe game where the computer uses the Alpha-Beta Pruning algorithm to make optimal moves efficiently. The system should allow a human player to compete against the AI through a GUI interface, where the AI evaluates all possible outcomes and selects the best possible move.

**GOAL**

The goal of this project is to develop an AI-based Tic-Tac-Toe game using Alpha-Beta Pruning that plays optimally against a human player. The AI should minimize unnecessary computations while ensuring efficient gameplay and accurate decision-making.

**THEORETICAL BACKGROUND**

The Minimax algorithm is widely used in two-player turn-based games such as Tic-Tac-Toe, Chess, and Checkers. It assumes that both players play optimally—one maximizing the score and the other minimizing it. However, the basic Minimax algorithm explores all possible game states, which can be computationally expensive.  
  
Alpha-Beta Pruning improves the efficiency of Minimax by eliminating branches that do not influence the final decision. It keeps track of two parameters—Alpha (the best value that the maximizer currently can guarantee) and Beta (the best value that the minimizer can guarantee). When Alpha becomes greater than or equal to Beta, the branch is pruned.  
  
This technique significantly reduces the search space and processing time, making it an ideal choice for Tic-Tac-Toe AI implementation.

**ALGORITHM EXPLANATION WITH EXAMPLE**

The Alpha-Beta pruning algorithm evaluates the game tree by applying Minimax principles while skipping branches that cannot affect the final decision. During the search, if the AI realizes that a particular branch will not produce a better result than an already explored one, it prunes that branch.  
  
For example, in a Tic-Tac-Toe board, if the AI finds a move that ensures victory, it does not evaluate the remaining moves as they are unnecessary for the outcome.

**IMPLEMENTATION AND CODE**

import tkinter as tk

from tkinter import messagebox

import random

# Initialize window

root = tk.Tk()

root.title("Tic Tac Toe - Alpha Beta AI")

# Board and variables

board = [" " for \_ in range(9)]

buttons = []

player = "X"

ai = "O"

# Check winner

def check\_winner(b, symbol):

win\_patterns = [(0,1,2),(3,4,5),(6,7,8),

(0,3,6),(1,4,7),(2,5,8),

(0,4,8),(2,4,6)]

return any(b[a]==b[b1]==b[c]==symbol for a,b1,c in win\_patterns)

# Check draw

def is\_draw(b):

return all(cell != " " for cell in b)

# Minimax with Alpha-Beta pruning

def minimax(b, depth, is\_max, alpha, beta):

if check\_winner(b, ai):

return 1

elif check\_winner(b, player):

return -1

elif is\_draw(b):

return 0

if is\_max:

max\_eval = -float("inf")

for i in range(9):

if b[i] == " ":

b[i] = ai

eval = minimax(b, depth + 1, False, alpha, beta)

b[i] = " "

max\_eval = max(max\_eval, eval)

alpha = max(alpha, eval)

if beta <= alpha:

break

return max\_eval

else:

min\_eval = float("inf")

for i in range(9):

if b[i] == " ":

b[i] = player

eval = minimax(b, depth + 1, True, alpha, beta)

b[i] = " "

min\_eval = min(min\_eval, eval)

beta = min(beta, eval)

if beta <= alpha:

break

return min\_eval

# AI move

def best\_move():

best\_score = -float("inf")

move = None

for i in range(9):

if board[i] == " ":

board[i] = ai

score = minimax(board, 0, False, -float("inf"), float("inf"))

board[i] = " "

if score > best\_score:

best\_score = score

move = i

if move is not None:

board[move] = ai

buttons[move].config(text=ai, state="disabled")

# Button click handler

def click(index):

if board[index] == " ":

board[index] = player

buttons[index].config(text=player, state="disabled")

if check\_winner(board, player):

messagebox.showinfo("Result", "🎉 You Win!")

reset\_board()

return

elif is\_draw(board):

messagebox.showinfo("Result", "🤝 It's a Draw!")

reset\_board()

return

best\_move()

if check\_winner(board, ai):

messagebox.showinfo("Result", "💻 AI Wins!")

reset\_board()

elif is\_draw(board):

messagebox.showinfo("Result", "🤝 It's a Draw!")

reset\_board()

# Reset function

def reset\_board():

global board

board = [" " for \_ in range(9)]

for button in buttons:

button.config(text=" ", state="normal")

# GUI setup

for i in range(9):

btn = tk.Button(root, text=" ", font=("Arial", 24), width=5, height=2,

command=lambda i=i: click(i))

btn.grid(row=i//3, column=i%3)

buttons.append(btn)

reset\_button = tk.Button(root, text="Reset", font=("Arial", 14),

command=reset\_board, bg="lightblue")

reset\_button.grid(row=3, column=0, columnspan=3, sticky="nsew")

root.mainloop()

**OUTPUT**

The following screenshots show the GUI implementation and results of the Tic-Tac-Toe game using Alpha-Beta Pruning.

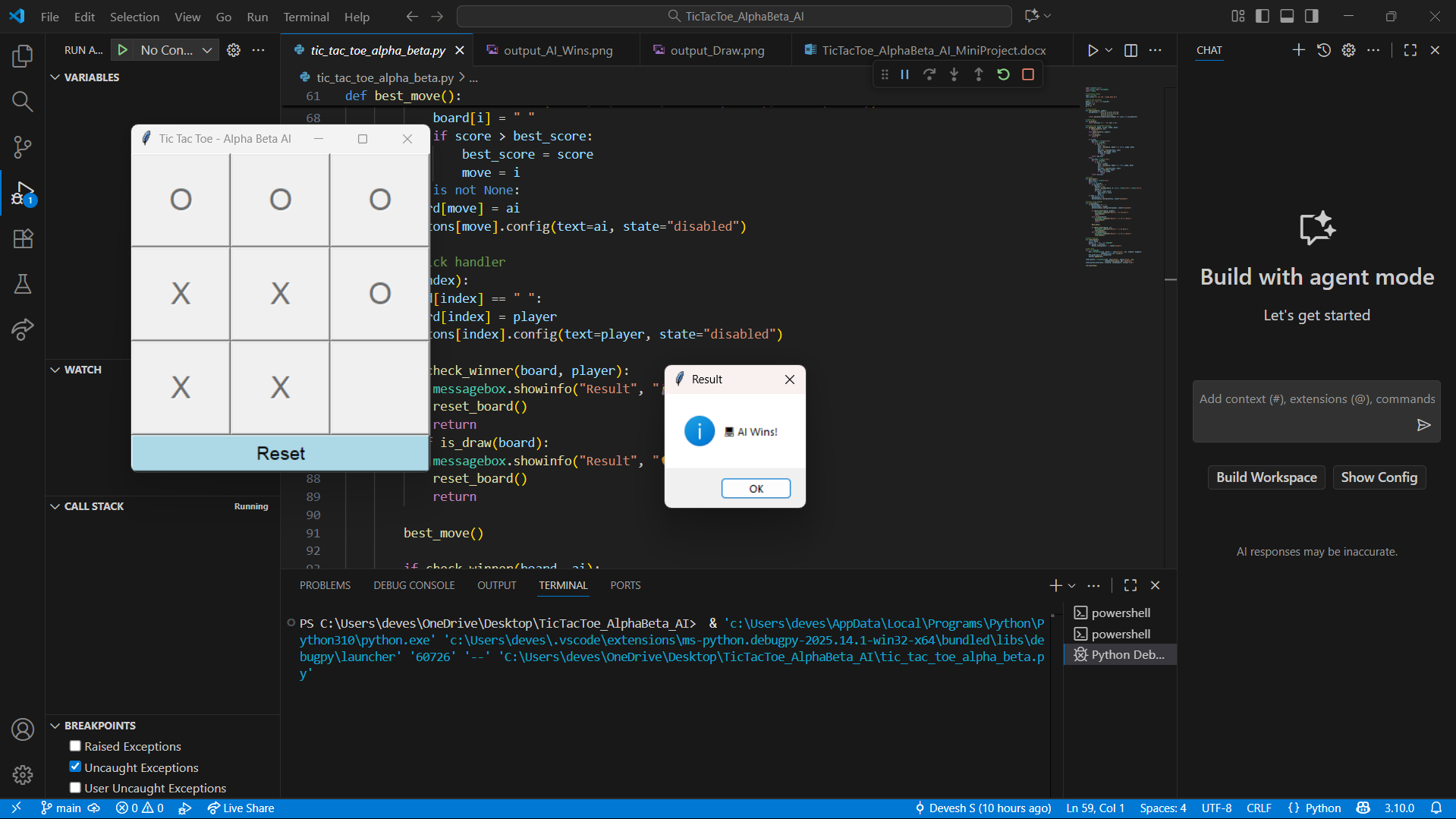


Figure 1: Tic-Tac-Toe Game – AI Wins

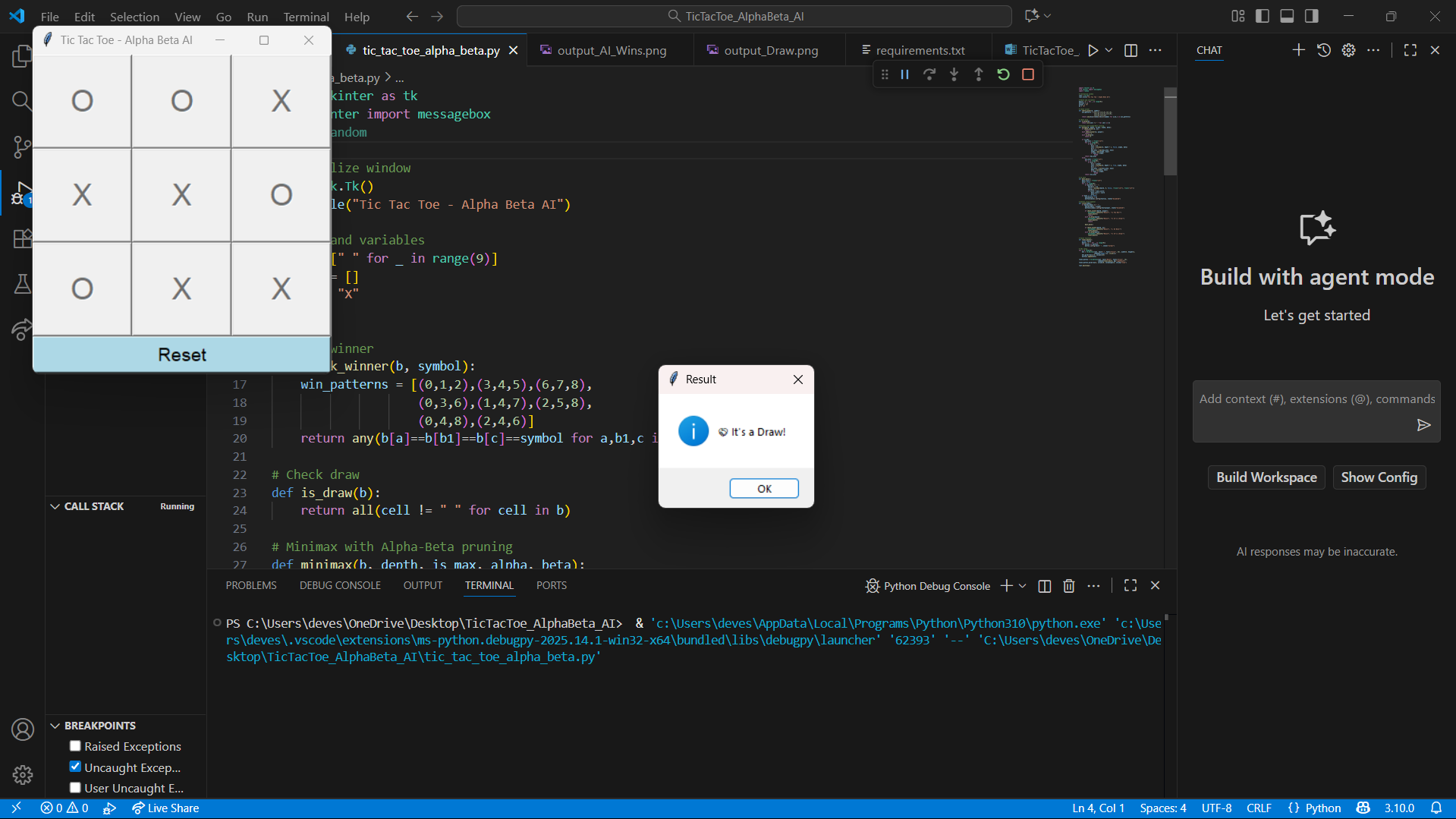


Figure 2: Tic-Tac-Toe Game – It's a Draw

**RESULTS AND FUTURE ENHANCEMENT**

The Tic-Tac-Toe game using Alpha-Beta Pruning was successfully implemented. The AI opponent plays optimally, ensuring that it either wins or draws every game. By using Alpha-Beta Pruning, the number of game states evaluated is significantly reduced, leading to faster decision-making.  
  
Future enhancements could include expanding the game board size (e.g., 4x4 or 5x5), integrating heuristic evaluation functions, or adapting the algorithm for more complex games like Connect Four or Chess.

|  |  |
| --- | --- |
| **Git Hub Link of the project and report** | **https://github.com/devesh-54/Tic-Tac-Toe-AlphaBeta-AI/tree/main** |

**REFERENCES**

1. Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach.

2. TutorialsPoint - Minimax Algorithm in Artificial Intelligence.

3. GeeksforGeeks - Alpha-Beta Pruning in AI.

4. Towards Data Science - Implementing Game AI using Minimax and Alpha-Beta Pruning.

5. ResearchGate - Optimization Techniques in Game AI.