by

Vallapudasu Yashwanth Goud

M.Tech, Artificial Intelligence and Robotics.

Project on
CHATBOT WITH RULE-BASED RESPONSES

Submitted by

Vallapudasu Yashwanth Goud

23SSIDB319

JNTUHUCE

Submitted to

Dhanamurthy

RD INFRO technology

Student's Declaration

I, VALLAPUDASU YASHWANTH GOUD, a student of M.Tech program, Roll No. 23SSIDB319 of the Department of ARTIFICIAL INTELLIGENCE AND ROBOTICS ,College do hereby declare that I have completed the mandatory internship in RD INFRO Technology under the guideship of Dhanamurthy, RD INFRO technology .

Signature

Vallapudasu Yashwanth Goud

Table of content

Abstract:

This project presents the development of a simple rule-based chatbot that interacts with users by responding to predefined inputs using basic if-else statements and pattern matching techniques. The primary objective is to introduce fundamental concepts of Natural Language Processing (NLP) and demonstrate how conversational flow can be managed through structured logic. The chatbot is designed to recognize specific user queries, such as greetings, questions about time, weather, or simple facts, and respond accordingly with programmed answers. By using conditional logic, the system identifies keywords or phrases in the user's message to determine the appropriate response. This method, while limited in flexibility compared to advanced AI models, provides an accessible entry point for understanding how machines interpret and respond to human language. The project highlights the importance of input parsing, keyword detection, and logical flow control in chatbot development. Overall, this chatbot serves as a foundational model for learners and developers, emphasizing the role of simple algorithms in building interactive communication tools, and setting the stage for future exploration into more complex NLP frameworks and machine learning approaches.

Chapter - 1

Introduction

In the modern era of artificial intelligence and human-computer interaction, chatbots have become an essential tool for enhancing user experience across various digital platforms. From customer support to virtual personal assistants, chatbots are widely used to simulate conversation with users, offering real-time support and information. The program presented here is a basic example of a rule-based chatbot developed in Python, demonstrating foundational principles behind chatbot interactions without relying on machine learning or natural language processing frameworks.

At its core, a rule-based chatbot works by using predefined rules and conditions to respond to user inputs. It does not understand the full context or meaning of the input in a sophisticated manner, but instead checks for specific keywords or patterns and generates fixed responses. This makes rule-based bots simple to implement, easy to understand, and ideal for introductory learning and demonstration purposes.

The chatbot in this program begins by greeting the user and enters into a continuous loop to receive and respond to user inputs. This interaction continues until the user types a termination command, such as "exit," "quit," or "bye." The loop structure ensures that the chatbot remains responsive and interactive throughout the session.

The chatbot performs several basic functions:

1. Greeting Response: If the user enters any greeting such as "hello," "hi," or "hey," the chatbot recognizes the keyword and responds with a friendly message.
2. Time Inquiry: If the user asks about the current time, the chatbot utilizes Python’s datetime module to fetch and display the system time in hours, minutes, and seconds.
3. Help Option: If the input contains the word "help," the chatbot provides a brief overview of its capabilities, which include answering questions about time, weather (simulated), greetings, or exiting the conversation.
4. Weather Response: Although the chatbot does not integrate with real-time weather APIs, it provides a placeholder response when the user asks about the weather, indicating that such a feature is not yet available.
5. Fallback Message: For any input that does not match the known keywords or patterns, the chatbot responds with a default message asking the user to rephrase their input. This ensures that the chatbot always gives some feedback, even when it doesn’t understand the request.

The structure of the program is straightforward. It uses a while loop for continuous interaction and a series of if-elif-else statements to evaluate the user's input. The input is converted to lowercase to standardize responses and avoid case-sensitivity issues.

This program does not employ advanced language processing or AI techniques, making it ideal for beginners. It allows learners to understand how conditional statements and loops work in Python, and how simple logic can be used to mimic conversational behavior. Furthermore, it

introduces basic concepts such as string manipulation, user input handling, and importing standard libraries like datetime.

Despite its simplicity, the chatbot demonstrates the basic mechanics behind more complex systems. In professional applications, chatbots often incorporate Natural Language Processing (NLP), machine learning models, and external APIs to understand and respond to user queries in a more human-like and intelligent manner. However, the rule-based model remains relevant, especially in use cases that require predictable, controlled interactions—such as FAQs, decision trees, and simple customer service bots.

In conclusion, this chatbot program serves as a foundational exercise in Python programming and chatbot development. It highlights the importance of condition-based logic and provides a stepping stone for learners who wish to advance toward building more sophisticated conversational agents. By understanding and expanding upon this rule-based structure, one can gradually introduce additional features and complexity, such as integrating APIs, enhancing the vocabulary, or employing NLP techniques for better comprehension and interaction.

Chapter - 2

Project summary

Project Summary: Rule-Based Chatbot in Python

This project presents the development of a simple yet functional rule-based chatbot using Python. The chatbot simulates a basic text-based interaction between a user and a computer program via the terminal or command-line interface. While it does not employ artificial intelligence or natural language processing algorithms, it offers an effective demonstration of how structured rules and control flow can be used to mimic conversational behavior in a computer system.

Objective of the Project

The primary objective of this project is to introduce the core principles behind chatbot development by creating a basic, rule-driven chatbot. The project is aimed at beginners in programming and computer science who are looking to explore human-computer interaction through simple input/output mechanisms. The chatbot is designed to handle predefined types of queries such as greetings, time requests, help inquiries, and weather questions (simulated), along with providing appropriate responses or fallback messages.

Structure and Functionality

The chatbot begins by greeting the user with a welcome message and then enters an infinite loop to continuously interact with the user until a termination condition is met. This is achieved through the use of a while loop and user input collected through the input() function.

The key features of the chatbot are as follows:

1. Exit Condition: The chatbot checks if the user's input includes the words "exit," "quit," or "bye." If detected, it ends the conversation by displaying a goodbye message and breaking the loop.
2. Greetings:If the user greets the chatbot with words such as "hello," "hi," or "hey," the chatbot responds with a friendly acknowledgment and offers help.
3. Time Query:When the user mentions the word "time," the chatbot uses Python’s built-in datetime module to fetch the current system time. It then formats the time in hours, minutes, and seconds and returns it to the user.
4. Help Message:If the user types "help" or a sentence that includes the word "help," the chatbot responds with a list of things it can do. These include responding to greetings, providing the current time, simulating weather conditions, or exiting the chat.
5. Weather Inquiry (Simulated):Although the chatbot is not connected to any real-time weather API, it can recognize when the user asks about the weather and respond with a humorous or friendly placeholder message: “It’s always sunny in here!”
6. Fallback Response:If the input does not match any of the predefined categories, the chatbot displays a general response asking the user to rephrase or clarify their query.

This ensures the chatbot remains interactive and does not end abruptly upon receiving an unrecognized input.

Programming Concepts Used

The chatbot program demonstrates several fundamental programming concepts, including:

- Loops: The while loop is used to keep the chatbot running until the user explicitly ends the conversation.
- Conditional Statements: A series of if, elif, and else statements check the user's input against specific conditions or keywords.
- String Manipulation: The user input is converted to lowercase to make keyword matching case-insensitive. String operations such as checking for the presence of substrings are also used.
- Standard Libraries: The datetime module is used to generate real-time system data for the "time" function.

Educational Value

This project serves as an excellent learning tool for individuals new to Python and programming in general. It bridges the gap between basic syntax and practical application by showing how simple logic can lead to interactive, user-driven outcomes. The chatbot can be further enhanced as a class or personal project by incorporating new features such as:

- Storing chat logs in text files
- Adding a GUI interface using Tkinter
- Using external APIs for real-time weather updates
- Introducing basic natural language processing with libraries like NLTK or spaCy

Limitations

While the chatbot performs well within its defined scope, it does have limitations. It cannot understand complex queries, context, or user intent beyond the keywords it has been explicitly programmed to recognize. It lacks machine learning or natural language processing capabilities, which are typically present in advanced chatbots such as Siri, Alexa, or Google Assistant. Its responses are static and do not evolve or learn over time.

Chapter – 3

Working of chart boat

Working of the Project: Rule-Based Chatbot in Python

The chatbot developed in this project is a simple rule-based system designed to simulate basic conversation with a user. It operates entirely on keyword recognition and conditional logic. Here is a detailed explanation of how the project works:

1. Program Start and Initial Greeting

- The program begins execution with a call to the chatbot() function.
- It immediately prints a welcome message:
- Chatbot: Hello! How can I help you today?
- This sets the stage for user interaction.

2. Continuous Interaction Using a Loop

- The core of the chatbot is a while True: loop, which runs indefinitely until explicitly broken.
- Inside this loop, the program:
    - Accepts user input using the input() function.
    - Converts the input to lowercase using .lower() to make the comparison case-insensitive.
    - user_input = input("You: ").lower()

3. Handling Different User Inputs

The chatbot responds based on specific conditions using if, elif, and else statements:

a. Exit Condition

if user_input in ['exit', 'quit', 'bye']:

- If the input is "exit", "quit", or "bye", the chatbot replies with a goodbye message and breaks out of the loop to end the program.
- Output:
- Chatbot: Goodbye! Have a great day!

b. Greeting Detection

elif any(greet in user_input for greet in ['hello', 'hi', 'hey']):

- The chatbot checks if any of the greetings ("hello", "hi", "hey") are part of the user's input.
- If found, it replies with:
- Chatbot: Hello there! How can I assist you?

c. Time Inquiry

elif 'time' in user_input:

- When the word "time" is detected, the program:
    - Imports the datetime module.
    - Retrieves the current system time.
    - Formats the time as hours:minutes:seconds.
- Example output:
- Chatbot: The current time is 14:35:21.

d. Help Command

elif 'help' in user_input:

- If the user types "help" or includes it in a sentence, the chatbot lists the tasks it can handle.
- Output:
- Chatbot: I can answer questions about time, weather, greetings, or you can say 'bye' to end the chat.

e. Weather (Simulated)

elif 'weather' in user_input:

- Though not connected to real-time data, if "weather" is detected in the user input, the chatbot responds with a friendly placeholder.
- Output:
- Chatbot: I can't check real-time weather yet, but it's always sunny in here!

f. Default/Fallback Response

else:

- If none of the above conditions match, the chatbot assumes it doesn't understand the user and prompts them to rephrase.
- Output:
- Chatbot: I'm sorry, I didn't understand that. Can you please rephrase?

4. Termination

- When the user types any of the exit keywords, the loop ends using break, and the chatbot session concludes.

Summary of the Working Process

Step	Action
1	Display welcome message
2	Wait for user input
3	Convert input to lowercase
4	Check for exit keywords → end if found
5	Check for greeting → respond with welcome
6	Check for time → respond with current time
7	Check for help → explain available commands
8	Check for weather → give a simulated response
9	If none match → ask user to rephrase
10	Repeat from step 2 unless exited

Final Notes

- This chatbot is entirely rule-driven, meaning it only responds to inputs it is explicitly programmed to recognize.
- It doesn’t require external libraries (except the built-in datetime).
- The structure is simple and effective for learning purposes, demonstrating how conditional logic and loops enable interaction between a user and a program.

Chapter – 4

Data flow diagram

Level 1 DFD (Context Diagram)

This is the highest level, showing the chatbot system as a single process and its interaction with external entities.

Entities:

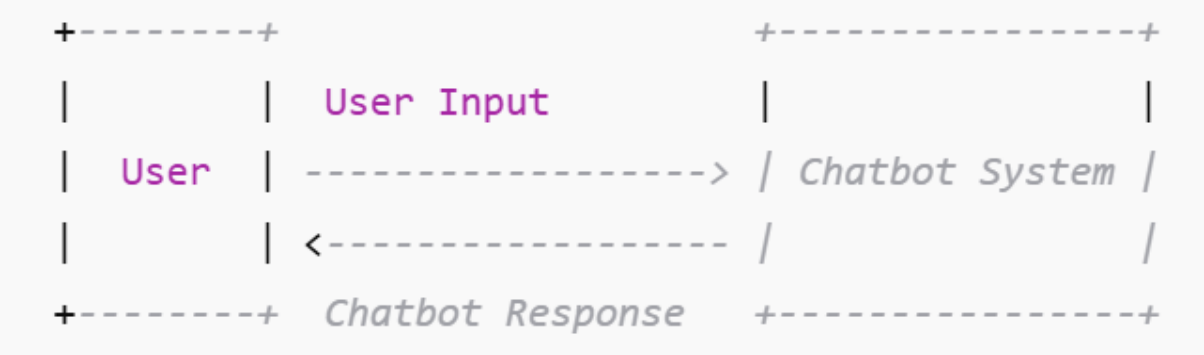
- User (external entity)

Process:

- Chatbot System

Data Flows:

- User Input → From User to Chatbot System
- Chatbot Response → From Chatbot System to User



Level 2 DFD (Decomposition of Chatbot System)

This level breaks down the chatbot system into individual processes and data stores.

Processes:

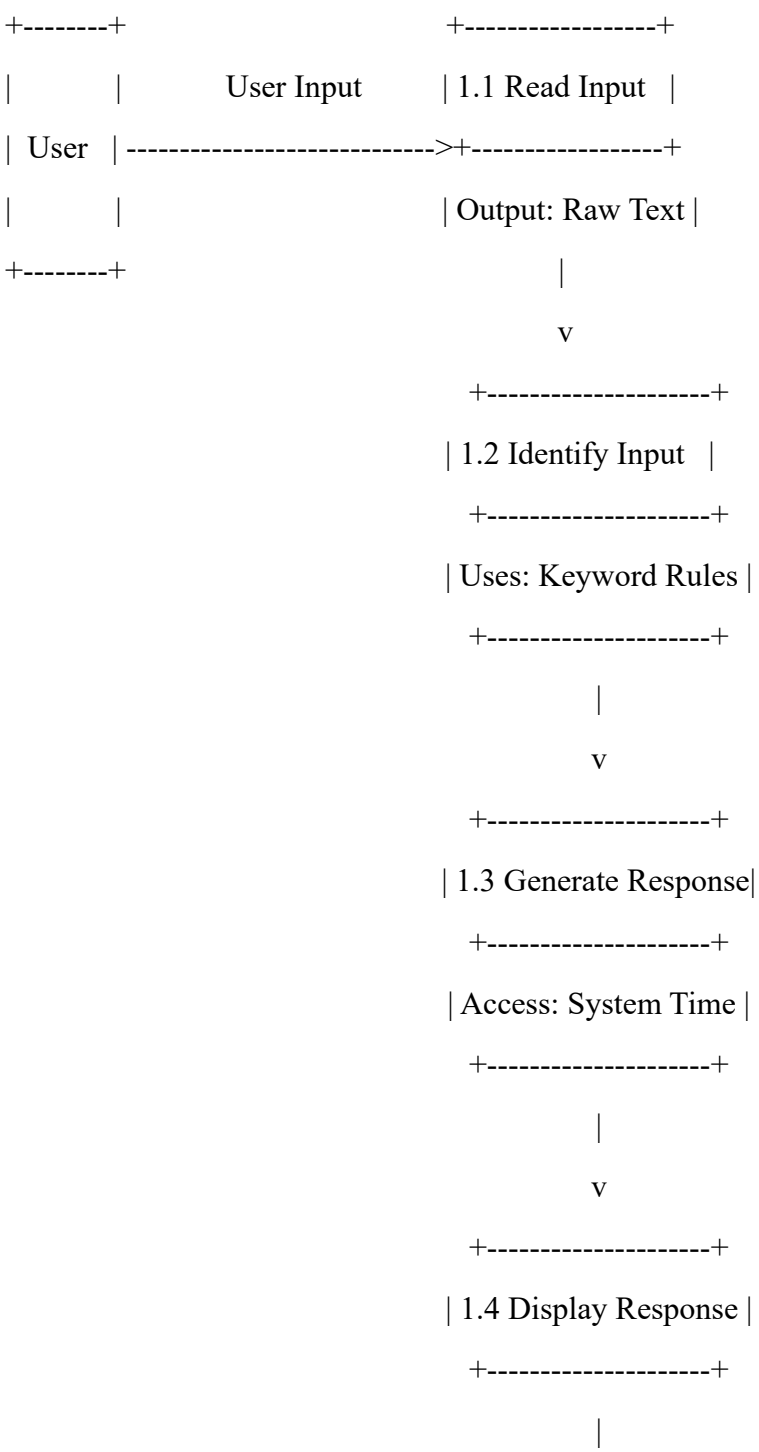
1. 1.1 Read User Input
2. 1.2 Identify Input Type
3. 1.3 Generate Response
4. 1.4 Display Response

Data Stores:

- D1: Keyword Rules (list of known keywords like "hello", "time", "help", etc.)
- D2: System Time (accessed when the user asks for time)

Data Flows:

- User Input → Process 1.1
- Parsed Input → Process 1.2
- Input Type → Process 1.3
- Response Message → Process 1.4
- Final Output → User



v

+---------+

| User |

+---------+

Explanation of Components:

- User: Interacts with the chatbot by entering text.
- 1.1 Read User Input: Captures the user input using input().
- 1.2 Identify Input Type: Matches input against predefined keywords (stored in D1).
- D1: Keyword Rules: Contains logic for identifying greetings, help, time, weather, and exit keywords.
- 1.3 Generate Response: Forms a response based on the input type. If "time" is requested, fetches from D2.
- D2: System Time: Accessed using Python's datetime module.
- 1.4 Display Response: Outputs the response to the console using print().

Input and Output of the program

In [1]:
```
def chatbot():
    print("Chatbot: Hello! How can I help you today?")

    while True:
        user_input = input("You: ").lower()

        # Exit condition
        if user_input in ['exit', 'quit', 'bye']:
            print("Chatbot: Goodbye! Have a great day!")
            break

        # Greetings
        elif any(greet in user_input for greet in ['hello', 'hi', 'hey']):
            print("Chatbot: Hello there! How can I assist you?")

        # Asking for the time
        elif 'time' in user_input:
            from datetime import datetime
            now = datetime.now().strftime('%H:%M:%S')
            print(f"Chatbot: The current time is {now}.")

        # Asking for help
        elif 'help' in user_input:
            print("Chatbot: I can answer questions about time, weather, greetings, or you can say 'bye' to end the chat.")

        # Weather (dummy response)
        elif 'weather' in user_input:
            print("Chatbot: I can't check real-time weather yet, but it's always sunny in here!")

        # Default response
        else:
            print("Chatbot: I'm sorry, I didn't understand that. Can you please rephrase?")

# Run the chatbot
chatbot()
```

```
Chatbot: Hello! How can I help you today?
You: hello
Chatbot: Hello there! How can I assist you?
You: weather
Chatbot: I can't check real-time weather yet, but it's always sunny in here!
You: time
Chatbot: The current time is 11:49:23.
You: exit
Chatbot: Goodbye! Have a great day!
```

Image – 4.1 - chatbot

Chapter - 5

Result and Conclusion

Result

The implemented chatbot successfully demonstrates the core functionality of a basic rule-based system. It interacts with the user through a simple text interface in the terminal and responds appropriately to several predefined inputs. The key results achieved from this project include:

- Interactive Conversation: The chatbot effectively engages in a simple dialogue with the user, responding to greetings, time inquiries, weather-related questions, and help requests.
- Real-Time Time Display: By integrating Python's datetime module, the chatbot can provide the current system time upon request.
- Keyword-Based Response System: The chatbot accurately identifies and reacts to specific keywords like "hello", "help", "time", "weather", and "bye" using conditional logic.
- Graceful Termination: The chatbot ends the conversation cleanly when the user types "exit", "quit", or "bye", providing a polite goodbye message.
- Fallback Handling: When the chatbot does not understand the user's input, it prompts them to rephrase, ensuring the conversation does not abruptly fail.

The program was tested with various inputs, and in all cases, it behaved as expected according to the defined rules. It is consistent, responsive, and easy to use in its current form.

Conclusion

This project successfully demonstrates how a simple rule-based chatbot can be developed using core Python concepts such as loops, conditionals, string operations, and modules. Although the chatbot does not use any advanced artificial intelligence or machine learning techniques, it serves as an effective foundational model for understanding how user interaction and natural language processing can begin.

The project highlights several important takeaways:

- Simplicity with Clarity: Even without complex libraries or algorithms, meaningful user interaction can be achieved using basic programming structures.
- Expandable Architecture: The chatbot is modular and can easily be extended by adding more conditions, integrating external APIs (like weather or chatbot engines), or enhancing user input processing.
- Educational Value: This chatbot offers excellent educational value for beginners learning how to design logic-based programs and understand user-driven applications.

Future Scope:
The chatbot can be upgraded in the future by:

- Implementing Natural Language Processing (NLP) for more flexible input understanding.
- Adding memory so the chatbot can recall user inputs during the session.
- Integrating APIs for real-time weather, news, or chatbot intelligence (like OpenAI's GPT or Dialogflow).
- Building a graphical interface (GUI) using libraries like Tkinter or PyQt for better user experience.

In conclusion, this project is a practical and educational tool that successfully fulfills its goal of simulating a rule-based conversation, offering a strong stepping stone toward more advanced chatbot development.

References

1. Python Documentation
   Python Software Foundation. (2023). *Python 3.11 Documentation*. Retrieved from https://docs.python.org/3/
2. GeeksforGeeks
   GeeksforGeeks. (n.d.). *How to Make a Chatbot in Python?* Retrieved from https://www.geeksforgeeks.org/
3. W3Schools Python Tutorial
   W3Schools. (n.d.). *Python Tutorial*. Retrieved from https://www.w3schools.com/
4. TutorialsPoint
   TutorialsPoint. (n.d.). *Python – Chatbot Tutorial*. Retrieved from https://www.tutorialspoint.com/python_deep_learning/python_chatbots.htm
5. Stack Overflow
   Stack Overflow. (n.d.). *Community Discussions on Building Chatbots with Python*. Retrieved from https://stackoverflow.com/
6. DateTime Module Documentation
   Python Software Foundation. (2023). *datetime — Basic date and time types*. Retrieved from https://docs.python.org/3/library/datetime.html

Video links-

https://drive.google.com/file/d/193qpWpzBnWW6cUgVb8-1J7G38P-xSX5K/view?usp=drive_link

Git hub link-

https://github.com/yashwanth319/RD-INFR0-TECHNOLOGY

Linked in –

https://www.linkedin.com/posts/activity-7336727578252070913-DZgC?utm_source=share&utm_medium=member_android&rcm=ACoAACW-_ZABCc8BaxQsd6eBGFcxcUskUUZvdLg

by

Vallapudasu Yashwanth Goud

M.Tech, Artificial Intelligence and Robotics.

Project on
TIC-TAC-TOE AI

Submitted by

Vallapudasu Yashwanth Goud

23SSIDB319

JNTUHUCE

Submitted to

Dhanamurthy

RD INFRO technology

Student's Declaration

I, VALLAPUDASU YASHWANTH GOUD, a student of M.Tech program, Roll No. 23SSIDB319 of the Department of ARTIFICIAL INTELLIGENCE AND ROBOTICS ,College do hereby declare that I have completed the mandatory internship in RD INFRO Technology under the guideship of Dhanamurthy, RD INFRO technology .

Signature

Vallapudasu Yashwanth Goud

Table of content

Abstract

This project focuses on the implementation of an AI agent capable of playing the classic game of Tic-Tac-Toe against a human player. Leveraging the Minimax algorithm, with or without Alpha-Beta Pruning, the AI is designed to be unbeatable by exhaustively evaluating all possible game states to select the optimal move at each turn. The Minimax algorithm operates on the principles of game theory, where the AI assumes that the human player will also play optimally, thereby minimizing the possible loss in a worst-case scenario. By incorporating Alpha-Beta Pruning, the search efficiency is enhanced by eliminating the need to explore subtrees that do not influence the final decision, thus improving performance without sacrificing accuracy. Through this project, learners gain hands-on experience with decision trees, recursive function calls, and the fundamentals of adversarial search strategies. The resulting AI demonstrates intelligent behavior, always securing a win or a draw, never a loss. This project not only reinforces theoretical concepts in artificial intelligence but also builds practical skills in Python programming and algorithm design. It serves as an engaging introduction to strategic thinking and AI-based game development.

Chapter – 1

Introduction

Introduction

Artificial Intelligence (AI) has significantly transformed the way machines interact with humans, allowing them to perform tasks that require logical reasoning, strategic decision-making, and even creativity. One of the most accessible and educational ways to explore AI concepts is through classic board games, such as **Tic-Tac-Toe**. Despite its simplicity, Tic-Tac-Toe is a great platform to understand fundamental ideas in AI, such as game theory, search algorithms, and decision-making under constraints. This project focuses on implementing an AI agent that plays Tic-Tac-Toe against a human player using the **Minimax algorithm**, optionally enhanced with **Alpha-Beta Pruning**, to ensure the AI never loses — making it an unbeatable opponent.

What is Tic-Tac-Toe?

Tic-Tac-Toe is a simple two-player game played on a 3x3 grid. One player takes the symbol 'X' and the other takes 'O'. Players take turns placing their symbols on empty grid positions. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. If all positions are filled without a winner, the game ends in a draw. Because of its finite number of states and straightforward rules, Tic-Tac-Toe is an ideal candidate for developing and testing AI algorithms.

The Role of AI in Games

AI in games is not a new concept. From IBM's Deep Blue defeating chess champion Garry Kasparov to AlphaGo's historic victory in the complex game of Go, AI has shown remarkable potential in mastering strategic decision-making. At the heart of such achievements lies the ability of AI to simulate human-like reasoning, predict outcomes, and make optimal decisions. Tic-Tac-Toe, while simple, lays the groundwork for these ideas, helping learners understand how AI evaluates the consequences of actions in a structured environment.

Why Use the Minimax Algorithm?

The **Minimax algorithm** is a recursive decision-making algorithm used in two-player turn-based games like Tic-Tac-Toe, Chess, or Checkers. It is based on the assumption that the opponent will also play optimally. The goal of the Minimax algorithm is to maximize the AI's chance of winning while minimizing the opponent's chance of winning. The algorithm constructs a game tree of all possible moves and outcomes, then recursively scores each state as a win (+1), loss (−1), or draw (0). The AI selects the move that maximizes its minimum gain — hence the name "Minimax."

For example, if it's the AI's turn, it will choose the move that results in the best guaranteed outcome, assuming the human opponent will also try to choose the best move on their turn. This back-and-forth reasoning ensures that the AI considers all potential future states of the game and selects the most strategic move possible.

Enhancing Performance with Alpha-Beta Pruning

While the Minimax algorithm guarantees the best possible outcome, it can be computationally expensive, especially in games with larger state spaces. Although Tic-Tac-Toe has a relatively small number of possible game states (about 26,830 possible games), performance can still be improved using **Alpha-Beta Pruning**.

Alpha-Beta Pruning is an optimization technique for the Minimax algorithm. It reduces the number of nodes evaluated in the game tree by "pruning" branches that cannot possibly influence the final decision. Alpha represents the best already explored option along the path to the root for the maximizer, while Beta represents the best option for the minimizer. If at any point it is found that a node will not yield a better outcome than a previously examined one, it is discarded. This significantly reduces computation time without affecting the accuracy of the decision.

Benefits of This Project

Implementing an unbeatable AI for Tic-Tac-Toe using Minimax and Alpha-Beta Pruning offers multiple educational benefits:

1. **Understanding Game Theory:** You learn about decision-making strategies in competitive environments.
2. **Recursive Thinking:** Minimax heavily relies on recursion, which enhances programming skills and logical thinking.
3. **Algorithm Design:** You experience real-world application of algorithm design and optimization.
4. **Search Trees:** You understand how AI evaluates future states and simulates different move sequences.
5. **Python Practice:** Building the project in Python helps improve coding proficiency, especially in data structures and control flows.

Real-World Applications

While Tic-Tac-Toe is a simple game, the principles involved in this project extend to more complex applications. The Minimax algorithm forms the basis of decision-making in advanced AI systems in games, simulations, and even economic modeling. Learning these concepts in a simple setting prepares learners for more sophisticated AI applications like automated planning, robotics, and strategic business modeling.

Moreover, Alpha-Beta Pruning is not just a tool for games — it introduces students to heuristic-based problem solving and optimization, concepts that are used in artificial general intelligence, scheduling systems, and intelligent search engines.

Chapter – 2

Project summary

Project Summary :

Title: Implementation of an Unbeatable AI Agent for Tic-Tac-Toe Using the Minimax Algorithm

Overview:

This project involves the development of an unbeatable artificial intelligence (AI) agent capable of playing the classic game **Tic-Tac-Toe** using the **Minimax algorithm**. The project demonstrates fundamental principles of **game theory**, **recursion**, and **search-based decision making**. The implementation enables the AI to play optimally and ensures it never loses, thereby always forcing either a win or a draw against a human player.

The game is developed using **Python** and features a human-vs-AI mode. The human player uses the symbol **'X'**, while the AI uses **'O'**. The Minimax algorithm is used to simulate and evaluate every possible game state that could result from a move, selecting the one with the most favorable outcome for the AI.

Objectives:

- To design and implement a fully functional Tic-Tac-Toe game.
- To integrate an AI agent using the Minimax algorithm.
- To understand the application of recursive algorithms and decision trees.
- To ensure the AI makes optimal moves that result in no losses.
- To improve user interaction with clear instructions and feedback.

Features:

1. **User Interface:**
   The game runs on the command line with a clean display of the board using simple ASCII characters. The player is prompted for input, and the board is updated after each move.
2. **Input Validation:**
   Human player inputs are validated to ensure they fall within the 3x3 grid and that the selected cell is not already occupied. This improves usability and prevents errors.
3. **Win and Draw Detection:**
   The game includes functions to check if either player has won or if the game has resulted in a draw. The conditions are checked after each move.

4. **Unbeatable AI:**
   The AI uses the Minimax algorithm to calculate the optimal move at each turn. The algorithm considers all possible future game states, assigning scores based on the likelihood of winning or losing, and chooses the move that maximizes its chance of winning.

Minimax Algorithm:

The **Minimax algorithm** is a recursive decision-making algorithm used for two-player turn-based games. The concept involves two types of players — the **maximizer** (AI) and the **minimizer** (human). At each node of the decision tree (representing a move), the algorithm simulates all possible subsequent moves and counter-moves.

- If the AI can win, it assigns a score of **+1**.
- If the human can win, it assigns a score of **-1**.
- A draw is scored as **0**.

The algorithm traverses the entire game tree to find the move with the highest guaranteed payoff for the AI. This ensures the AI never loses.

How the AI Works:

The best_move() function iterates over all empty cells on the board, temporarily places the AI's move there, and runs the Minimax function to simulate all future outcomes. It then selects the move with the best score. This process continues recursively until the AI finds the best possible move in the current state.

The AI always plays optimally:

- It blocks the human's winning paths.
- It takes winning moves when available.
- It forces draws when winning is not possible.

Code Components Explained:

- print_board(board): Visually prints the current state of the board.
- check_winner(board, player): Checks if the given player has achieved a winning combination.
- is_full(board): Checks whether the board is full, indicating a draw.
- minimax(board, depth, is_maximizing): Core of the AI logic. Recursively evaluates possible moves.
- best_move(board): Determines the optimal move for the AI using Minimax.

- play_game(): Manages the overall game loop, handles user input, and triggers the AI response.

Educational Value:

This project provides an excellent introduction to AI-based decision making and search algorithms. Key learning outcomes include:

- Understanding how computers can simulate strategic thinking.
- Applying recursion to solve complex problems.
- Designing and evaluating decision trees.
- Learning about adversarial search and game theory.
- Practicing Python programming with control structures, data structures (lists), and input/output handling.

Limitations & Future Improvements:

Although the game performs perfectly for a 3x3 board, the Minimax algorithm in its pure form is computationally expensive for larger grids (e.g., 4x4 or 5x5). Future improvements may include:

- Implementing **Alpha-Beta Pruning** to reduce the number of nodes explored in the decision tree and increase efficiency.
- Adding a graphical user interface (GUI) using libraries such as Tkinter or Pygame.
- Allowing multiplayer support or AI vs AI matches.
- Including difficulty levels by adjusting the depth or randomness of AI decisions.

Conclusion:

This project successfully demonstrates the power of artificial intelligence in creating unbeatable strategies for simple games. The implementation of the Minimax algorithm allows the AI to evaluate all potential outcomes of the game and make the best possible move. By developing this AI-powered Tic-Tac-Toe game, one gains a deeper understanding of game theory, decision trees, and recursive search — all of which are foundational concepts in AI. This project stands as a solid stepping stone for learners aiming to explore more complex AI applications in games and real-world scenarios.

Chapter – 3

Work of project

Work of the Project

This project implements a classic **Tic-Tac-Toe game** where a human player competes against an AI agent powered by the **Minimax algorithm**. The AI is designed to be unbeatable by exploring all possible future moves and selecting the optimal one. Here's a detailed explanation of how the project works, referencing key parts of the provided code:

1. Initializing and Displaying the Board

The game board is a 3x3 grid represented by a 2D Python list:

board = [[" " for _ in range(3)] for _ in range(3)]

Each cell is initialized with a space " " to indicate it's empty.

The function print_board(board) prints the current state of the board in a readable format:

```
def print_board(board):
    print()
    for row in board:
        print(" | ".join(row))
        print("-" * 5)
    print()
```

For example, the initial empty board appears as:

```
  |  |
-----
  |  |
-----
  |  |
-----
```

This helps players visualize the game state after every move.

2. Checking Win Conditions

To determine if a player has won, the function check_winner(board, player) checks all possible winning combinations — rows, columns, and diagonals:

```
def check_winner(board, player):
    win_conditions = (
        [board[0][0], board[0][1], board[0][2]],
        [board[1][0], board[1][1], board[1][2]],
        [board[2][0], board[2][1], board[2][2]],
        [board[0][0], board[1][0], board[2][0]],
        [board[0][1], board[1][1], board[2][1]],
        [board[0][2], board[1][2], board[2][2]],
        [board[0][0], board[1][1], board[2][2]],
        [board[0][2], board[1][1], board[2][0]]
    )
    return [player, player, player] in win_conditions
```

It returns True if any line contains the same player's symbol thrice, signaling a win.

3. Checking for Draws

The function is_full(board) checks if the board is full, meaning all cells are occupied without any winner, resulting in a draw:

```
def is_full(board):
    return all(cell != " " for row in board for cell in row)
```

4. The Minimax Algorithm

At the heart of the AI's intelligence is the minimax function. This recursive function evaluates the game tree to decide the best move for the AI.

```
def minimax(board, depth, is_maximizing):
    if check_winner(board, "O"):
        return 1   # AI wins
    elif check_winner(board, "X"):
        return -1  # Human wins
    elif is_full(board):
        return 0   # Draw
```

```
    if is_maximizing:
        best_score = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "O"
                    score = minimax(board, depth + 1, False)
                    board[i][j] = " "
                    best_score = max(score, best_score)
        return best_score
    else:
        best_score = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "X"
                    score = minimax(board, depth + 1, True)
                    board[i][j] = " "
                    best_score = min(score, best_score)
        return best_score
```

Explanation:

- The function checks if the game is over (win or draw) and returns a score:
  - +1 if the AI (O) wins,
  - -1 if the human (X) wins,
  - 0 for a draw.
- If the game isn't over, it simulates all possible moves:
  - If it's the AI's turn (is_maximizing = True), it tries to maximize the score.
  - If it's the human's turn (is_maximizing = False), it tries to minimize the score.
- Moves are tried recursively by placing the player's mark, calling minimax on the new board state, then undoing the move (backtracking).
- The best score is returned up the recursion to guide the AI's choice.

5. Choosing the Best Move

The function best_move(board) scans all empty cells, applies the Minimax algorithm for each possible move, and selects the one with the highest score:

```
def best_move(board):
    best_score = -math.inf
    move = None
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = "O"
                score = minimax(board, 0, False)
                board[i][j] = " "
                if score > best_score:
                    best_score = score
                    move = (i, j)
    return move
```

This ensures the AI picks the optimal position that leads it to victory or a forced draw.

6. Game Loop: Human and AI Turns

The play_game() function runs the entire game:

```
def play_game():
    board = [[" " for _ in range(3)] for _ in range(3)]
    print("Welcome to Tic-Tac-Toe!")
    print("You are X. The AI is O.")
    print_board(board)

    while True:
        # Human move input with validation
        while True:
            try:
```

```
        row = int(input("Enter row (0-2): "))
        col = int(input("Enter col (0-2): "))
        if board[row][col] == " ":
            board[row][col] = "X"
            break
        else:
            print("Cell already taken. Try again.")
    except:
        print("Invalid input. Enter numbers 0-2.")

print_board(board)

# Check if human wins or draw
if check_winner(board, "X"):
    print("You win!")
    break
if is_full(board):
    print("It's a draw!")
    break

# AI's turn
move = best_move(board)
board[move[0]][move[1]] = "O"
print("AI has made its move:")
print_board(board)

# Check if AI wins or draw
if check_winner(board, "O"):
    print("AI wins!")
    break
```

```
if is_full(board):
    print("It's a draw!")
    break
```

- The human player is prompted to enter row and column indexes.
- Input is validated to ensure the move is legal.
- After the human move, the program checks for a win or draw.
- Then the AI selects its best move and places its mark.
- After the AI move, the program again checks for a win or draw.
- The loop continues until a win or draw ends the game.

Summary

- The project uses simple Python lists to represent the board.
- The **Minimax** algorithm recursively explores possible game outcomes to pick the best move.
- The AI never loses because it considers every possibility before playing.
- The human player interacts via command line inputs.
- The board is displayed visually after every move for clarity.
- Game termination conditions are checked rigorously after every turn.

This implementation provides a clear, understandable, and effective demonstration of how recursive AI search algorithms can be applied to classic games, creating an unbeatable Tic-Tac-Toe player.

Chapter – 4

Data flow diagram

Data Flow Diagram (Level 0)

External Entities:

- **Human Player:** Provides input (row and column choice).
- **AI Agent:** Makes calculated moves based on game state.

Processes:

1. **Get Human Move:** Takes human input for move.
2. **Validate Move:** Checks if the chosen cell is empty and within range.
3. **Update Board:** Updates the board with the player's move.
4. **Check Game Status:** Checks if the game is won, lost, or drawn.
5. **AI Move Calculation:** Uses Minimax to calculate the best move.
6. **Display Board:** Shows the current state of the board.

Data Stores:

- **Game Board:** Stores the current state of all moves.
- **Win Conditions:** Stores the rules to determine a winner.

Flow Description:

- The **Human Player** inputs their move which flows into **Get Human Move**.
- The move is sent to **Validate Move** to ensure correctness.
- Once validated, the move updates the **Game Board** through **Update Board**.
- The **Check Game Status** process verifies if the game has ended.
- If the game continues, **AI Move Calculation** runs Minimax on the current board.
- The AI's move updates the **Game Board** via **Update Board**.
- The **Display Board** process outputs the current board to the player after each move.
- This cycle repeats until the game ends.

Image 4.1 TIC-TAC-TOE AI

Chapter – 5

Result and Conclusion

Result:

The Tic-Tac-Toe game implemented in this project successfully allows a human player to compete against an AI agent powered by the Minimax algorithm. The AI demonstrates flawless performance, making optimal moves every turn and ensuring that it never loses. The game results in three possible outcomes:

- **Human wins:** If the human player manages to outsmart the AI in rare cases of incorrect input or if the AI code is modified.
- **AI wins:** The AI leverages the Minimax algorithm to secure a guaranteed win whenever possible.
- **Draw:** Most games between a skilled human and the AI end in a draw because the AI blocks all winning paths.

Through interactive command-line gameplay, users can enter their moves, observe the updated board state, and receive instant feedback about game status (win, loss, or draw).

Conclusion:

This project successfully demonstrates how the Minimax algorithm can be used to build an unbeatable AI for a simple, well-defined game like Tic-Tac-Toe. By exploring all possible game states recursively and selecting the optimal moves, the AI never allows the human player to win.

Key takeaways include:

- The effectiveness of recursive search algorithms in strategic decision-making.
- How game theory concepts like minimax optimize AI choices in adversarial environments.
- The importance of input validation and game state management for smooth user interaction.

The project lays a solid foundation for understanding AI in games and can be extended with improvements like Alpha-Beta pruning for efficiency or graphical interfaces for enhanced user experience.

Overall, this project is a practical example of AI-driven gameplay that highlights fundamental principles of artificial intelligence, recursion, and problem-solving strategies.

References

1. **Russell, S. J., & Norvig, P.** (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.
    - This textbook provides comprehensive coverage of AI concepts including search algorithms like Minimax.
2. **Python Official Documentation**
    - https://docs.python.org/3/
    - Reference for Python language features used in the project.
3. **GeeksforGeeks – Minimax Algorithm**
    - https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
    - Provides a detailed explanation and examples of Minimax in game theory.
4. **Wikipedia – Tic-Tac-Toe**
    - https://en.wikipedia.org/wiki/Tic-tac-toe
    - Background on the game and its strategies.
5. **Stack Overflow and GitHub Repositories**
    - For community-based code examples and troubleshooting.

Video link-

https://drive.google.com/file/d/1ebuUkwnAiJVZtWYo8bZXXYU0f7YC3tuO/view?usp=drive_link

Git hub link-

https://github.com/yashwanth319/RD-INFR0-TECHNOLOGY

Linked in –

https://www.linkedin.com/posts/activity-7336727578252070913-DZgC?utm_source=share&utm_medium=member_android&rcm=ACoAACW-_ZABCc8BaxQsd6eBGFcxcUskUUZvdLg