AI Project: Tic-Tac-Toe Winning Move Detection Using Forward Chaining in Python

Tustee Mazumdar
Id- 210054, B.Sc in Computer Science.
Asian University For Women, Chittagong, Bangladesh.

Submitted to: Nihad Karim Chowdhury
Submission date: December 5, 2024

1 Introduction

Logical reasoning has been one of the strong pillars of Artificial Intelligence [1]. The evaluation of Artificial Intelligence(AI) is integrated and developed with symbolic reasoning, emphasizing the importance of logical reasoning in order to achieve robust AI systems [2]. Tic Tac Toe, usually known as Noughts and Crosses, is played in a 3X3 grid with two players. There are two usual symbols of this game - "X" and "O". At first, the players among them decide who will use X and who will go first. After that, both of the players take turns by placing their symbols on the vacant cell. If any of the players put his marks on - vertically, diagonally or horizontally; then that respective player wins. On the other hand, if all the cells are filled with symbols along with none of the players has marks three in a row, the game ends in a draw.

There has been many research going on in this AI field even with a simple game Tic Tac Toe which has explored various computational strategies and theoretical analyses. For example, (Gontukkala et al., 2022) discuss designing the game using automate theory, which actually highlights its computational aspects [3]. Additionally, "Reinforcement Learning: Playing Tic-Tac-Toe" explores by using which learning technique an AI agent to learn and improve its game-play over time [4]. However, in this project, we aim to design an AI agent that can detect the winning move in the "Tic-Tac-Toe" by using the Forward Chaining rule based on first-order logic (FOL). In the particular given configuration given, player X occupies the positions P1 and P2(the first row) on the board. Along with that, P3 is completely empty which means, player X is one move away from winning. By using the Forward Chaining rule, we will determine whether placing X in P3 would result in a win for X. The key objective of this project is to explain how logical reasoning can be implemented programmatically, and also can be used to solve structured problems like identifying winning strategies in games.

2 Methodology

According to the project problem statement, to detect the winning move of the AI agent, two significant method has been used. The method includes, Forward Chainining mechanism as well as the First-Order-Logic is used to get the output.

The predicated the given below:

• ***Occupied(Position, Player)***: Indicates that a position is occupied by a player. For example, Occupied(P1,X).

• ***WinningMove(Player, Position)***: Indicates that placing a move at a certain position will result in a win for the player

. •***Empty(Pn)***: Here n is 1 to 9 is empty initially.

•***Draw():*** Denotes the game has finished by being tie

The knowledge base (KB) is represented in FOL as follows:

Empty(P1), Empty(P2), Empty(P3), Empty(P4)..., Empty(P9)

Initially, all positions P1 through P9 on the board were empty. The knowledge base keeps getting updated after each move.

The inference rule for detecting a winning move is as follows:

$$\forall \text{combo} \in \text{win\_combinations},\ \text{Occupied}(\text{combo}[0], X) \land \text{Occupied}(\text{combo}[1], X) \land \text{Occupied}(\text{combo}[2], X) \Rightarrow \text{Winning}(X)$$

This rule indicates that if all positions in the current combination are occupied by X which declares the winning move for X.

3 Implementation

In this project, the Tic-Tac-Toe game is played by two human players (human vs. human). First-order logic and forward chaining are used to run this game successfully. Moreover, for a great visualization, a GUI has been set up.

1. GUI setup: To create the graphical interface through the game, the Tkinter library has been used. Tk() has been used to create the window and title method(title()) to set the game title. In the GUI setup, buttons along with their font size, and font name are created and placed in a 3X3 grid which reflects the Tic-Tac-Toe game board.

2.Initializing the Knowledge Base: Firstly, as the same has not been started yet, the board is set up as empty. This results in initializing the Knowledge Base(KB) to all the board positions(p1-p9) as empty. **3.Updating the Knowledge Base:** The function `update_kB()` updates the knowledge Base after every single move. In the presented 9 positions on the board, each of which

```python
# Step 5: GUI Setup
window = tk.Tk() #creates the main window of the GUI using Tkinter #window is the .
#This window is the container for all the graphical elements (like buttons, labels, etc.)
window.title("AI Tic-Tac-Toe Game")

# Create 3x3 grid of buttons for the Tic-Tac-Toe board
buttons = []  # List to store buttons
for i in range(9):
    btn = tk.Button(window, text="", font=("Times New Roman", 24), width=5, height=2,
                    command=lambda i=i: human_move(i))  # Link to human_move function
    btn.grid(row=i // 3, column=i % 3)  # Arrange in 3x3 grid
    buttons.append(btn)  # Add button to list
```

Figure 1: GUI Setup

```python
# Knowledge base: predicates for the board state
knowledge_base = {f"Empty(P{i+1})": True for i in range(9)}
#it means all the positions are initially empty
#it is a dictory - stores value in pairs
#F-string will evaluate what's inside {} and insert the result into string
```

Figure 2: Initialization Knowledge Base

can be occupied by either the player's mark "X"or "O" or empty. If one of the positions is filled with "O" or "X"then this function stores two facts: marking the current filled position as occupied(e.g., not empty (Empty(P1) = False) and Occupied(P1, X) = True). If the current position is vacant, then it is marked as empty (Empty(P1) = True). This function tracks each cell in each move to ensure the correct reflection of the updated knowledge base in the board configuration.

```python
# Function to update the knowledge base after each move
def update_kB():
    for i in range(9): #loop through all 9 positions
        if board[i] == "X" or board[i] == "O": #Check if the current cell is occupied by X or O
            knowledge_base[f"Occupied(P{i+1}, {board[i]})"] = True #Mark the cell as occupied by X or O in th
            knowledge_base[f"Empty(P{i+1})"] = False #mark false those are occupied now.
            #For position P1 (board[0] == "X"), the knowledge base will be updated:
#Occupied(P1, X) = True
#Empty(P1) = False
        else:
            knowledge_base[f"Empty(P{i+1})"] = True #otherwise empty = true
```

Figure 3: Updated Knowledge Base

4.Checking Winning Move: The `is_winning_move()` function verifies if the current player has won by evaluating all the possible winning moves which include the columns, rows, and diagonals. In this specific process, first-order logic has been used. The function checks whether all three positions by one single player(either "X"or "O") in any eight-winning combination or not. If yes, then the occupied buttons in the winning combination are marked in yellow to denote the win. If not, there are unfortunately no winners.

5.Human Player Move: The `human_move()` function handles the human player's move by giving the symbols ("X"or "O")in the selected cells, updating the board then checking if the game is ended or not. If `game_over` is true, then it shows "The Game is Over *_*!" message in the window. After each move, the function checks for win by using the function `is_winning_move()` and finishes the game once the winner is found. If the board is marked full by both of the

```python
# Step 2: Check for a winner using first-order logic
def is_winning_move(): # checks if the current player (either "X" or "O") has won the game.
    # Define winning combinations (rows, columns, diagonals)
    win_combinations = [
        [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 combo in win_combinations:
        # Check if all positions in a combination are occupied by the current player
        if board[combo[0]] == board[combo[1]] == board[combo[2]] != "":
            # Color the winning cells in green
            buttons[combo[0]].config(bg="Yellow", state="normal")
            buttons[combo[1]].config(bg="Yellow", state="normal")
            buttons[combo[2]].config(bg="Yellow", state="normal")
            return True  # Winning condition met

    return False  # No winner
```

Figure 4: Winning Move

players then it results in the game being tied.

```python
# Step 3: Human player's move
def human_move(pos):
    global current_player, game_over  # Use the global variables to update current_player and game_over
    #global Current_player keeps tract which symbol I want to use as a current player variable(X or O)
    #global game over which keeps track of whether the game is still ongoing or has ended.
    if game_over:  # If the game is over, no more moves allowed
        result_label.config(text="The Game is Over *_*!")
        return

    if board[pos] == "":  # Check if the cell is empty
        board[pos] = current_player  # Place the current player's mark in the cell
        buttons[pos].config(text=current_player, state="disabled")  # Update button
        update_kB()  # Update knowledge base with the new move

        if is_winning_move():  ## Check for a winner
            result_label.config(text=f"yohooo!{current_player} Wins!")
            game_over = True  # Set game over flag to True
            return

        if "" not in board: #  # Check for a tie
            result_label.config(text="DRAWWWWWW!")
            game_over = True  # Set game over flag to True
            return
```

Figure 5: Human Player Move

4 Result

In this project, the tic-tac-toe game has been successfully implemented which includes both logical reasoning and a graphical GUI setup. Each of the moves of both players has been precisely tracked by the game in order to check for a draw or a win after each move. Highlighting the cells in yellow that create a winning combination, with a message showing who won the game. If the board is completely filled and doesn't satisfy any winning combination, it's a draw. The knowledge base gets updated after every move. Once the winner is found, no player can give any further move.

5 Conclusion

This project successfully demonstrated the application of Forward chaining and First Order Logic (FOL) to detect the winning moves of an AI agent in a

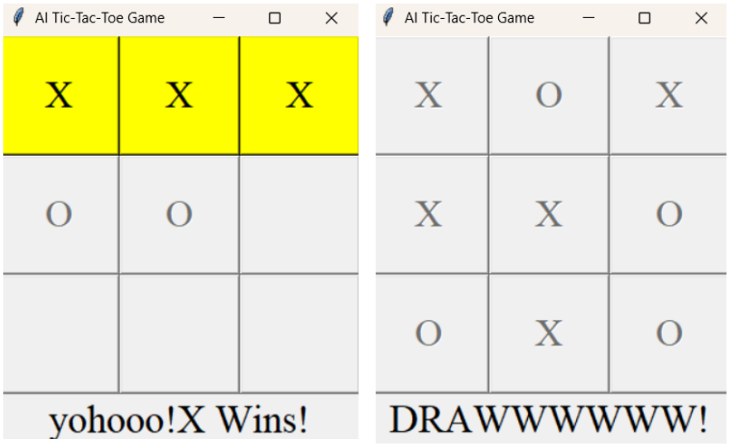

Figure 6: Possible outputs

structured game like Tic-Tac-Toe. By focusing on the initial knowledge base, updating it, and checking the winning move, the AI agent has successfully found the winning which ensures the power of logical reasoning in the field of AI. The graphical implementation has demonstrated the interaction between two players very effectively. The result label has played a vital role as the immediate feedback on the game's outcome. Still, there is room for future improvement e.g. adding buttons for restarting the game, some other GUI features, the game between a human vs. AI or AI vs. AI, and exploring additional logic rules.

References

[1] Vijay Ganesh, Sanjit A. Seshia, and Somesh Jha. Machine learning and logic: a new frontier in artificial intelligence. *Formal Methods in System Design*, 60:426–451, 2023.

[2] d'Avila Garcez, Artur and Luís C. Lamb. Neurosymbolic ai: the 3rd wave. *Artificial Intelligence Review*, 56:12387–12406, 2023.

[3] Sai Surya Teja Gontumukkala, Yogeshwara Sai Varun Godavarthi, Bhanu Rama Ravi Teja Gonugunta, and Supriya M. Implementation of tic tac toe game using multi-tape turing machine. In *2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)*, pages 381–386, 2022.

[4] Jocelyn Ho, Jeffrey Huang, Benjamin Chang, Allison Liu, and Zoe Liu. Reinforcement learning: Playing tic-tac-toe. *Preprint*, October 2023. Affiliations partially available.